14th ICCRTS

“C2 and Agility”

“Using Web Services and XML Security to Increase Agility in an Operational Experiment Featuring Cooperative ESM Operations”

Paper Id: 019

Topic 5: Experimentation and Analysis

Trude Hafsøe, Frank T. Johnsen, Nils A. Nordbotten, Espen Skjervold

Point of contact:

Trude Hafsøe
Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller
Norway

Telephone: +47 63 80 74 31
E-mail: trude.hafsoe@ffi.no

Using Web Services and XML Security to Increase Agility in an Operational Experiment Featuring Cooperative ESM Operations

Abstract
Agility is a key property for Command and Control (C2). Legacy applications perform special-purpose tasks often using binary and proprietary formats for data exchange. Since military forces work with an increasing number and variety of partners, we need to accommodate this by basing our C2 systems on standards. Web services technology is an enabler for the future in NEC. We have used Web services in conjunction with a legacy system for cooperative ESM (CESMO). Our prototype allowed both legacy systems and our new Web services based system to co-exist and operate together. Our experiments have shown how we can take this COTS and XML-based technology and apply it to a military scenario to gain added value over that of the legacy applications. In addition to supporting the legacy CESMO application and data format, we used XML technology to convert the data and disseminate it to systems requiring input in other formats. For example, we provided location data via NATO Friendly Force Information (NFFI) to Norway's operational tactical C2 system, which would not have been feasible without our Web services based prototype.

Introduction

Service Oriented Architecture (SOA) has been indentified as one of the key enabling technologies for realizing the Network Enabled Capabilities (NEC) vision of seamless information exchange encompassing all operational levels and also between nations. In order to verify the usability of Web services as a means to implement SOA in military networks, we participated in a large national experiment in late 2008. The focus of this experiment was to connect different communication systems from all the military services into one common network. This includes interconnections tried earlier, as well as some interconnections never tried before.

This national experiment consisted of a number of trials, one of which focused on Electronic Support Measures (ESM). ESM sensors are special radio receivers for the detection of emissions from radars and communication systems. The ESM sensors measure the radar frequencies and other characteristic parameters of the signal, and the direction or bearing to the emitter. By tracking emitters over time based on bearings from one ESM sensor, it can take a long time to establish target solutions, and the solutions may have limited accuracy.

With cooperating ESM sensors one can process simultaneous observations of an emitter from several ESM sensors, and obtain solutions more rapid and more accurate than is possible with one sensor, and one can use other and better methods that produce a more accurate emitter location (see Figure 1). Operations where the main concern is the detection and localization of radar emitters by cooperation between many sensor platforms are called Cooperative ESM operations (CESMO).

CESMO require the platforms equipped with ESM sensors to communicate with each other, often over narrow bandwidth network connections. The communication consists of short, structured messages which are exchanged between platforms. There are several messages types, and the different platforms involved in CESMO may have different needs when it comes to which message types they consume.

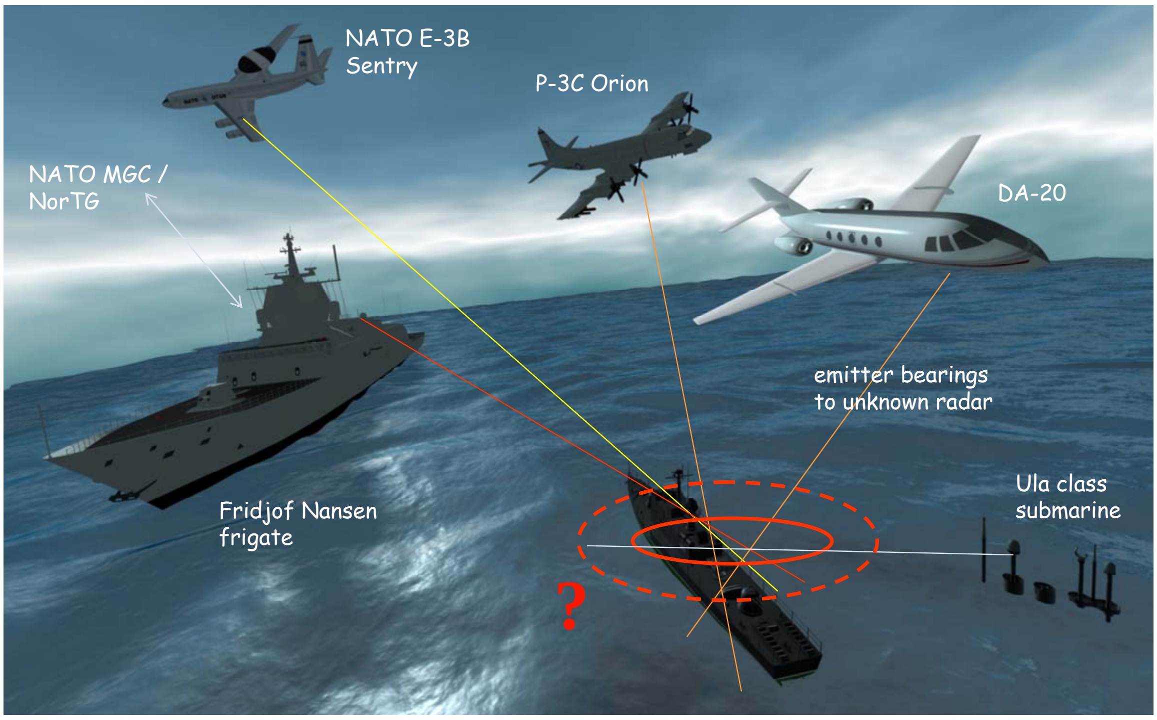

Figure 1 The CESMO concept

CESMO is a good example of operations where sharing of data is essential for the success of the operation. Traditionally signal intelligence and Electronic Warfare are areas where the sharing of data even between close allies has been limited. By using the interfaces and infrastructure of Web services, CESMO could benefit favorably from a SOA with accompanying flexible security solutions, and controlled data dissemination.

Motivation

The main motivation for using CESMO as a case for SOA was to show that it is feasible to SOA-enable existing operational systems, which brings substantial benefits. It was also important to demonstrate these benefits for operational personnel, in order to, hopefully, trigger some ideas around how SOA can be used.

One of our goals was to show that introducing a SOA infrastructure into the CESMO network can contribute to automate and speed up cooperative ESM operations. We added a Web service front-end to each platform, which made them appear as services. Each platform could then discover, and subscribe to, the services of other platforms.

Another goal of the SOA-CESMO experiments was to demonstrate how SOA enables information exchange between stove pipe systems, possibly in different military services. In other words, how SOA can function as an integrator, making it possible to find information and make it available across tactical and strategic systems in different military services. This was demonstrated by connecting the CESMO network to an army C2 system, which received geolocations of target emitters from the CESMO system. This army C2 system would in turn make this information available to others, in particular to a strategic C2 system called NORCCIS-II [1]. By doing this, we demonstrated how information could be gathered by both air force and naval units, processed in a central location and then distributed to interested parties within a different military service.

Another illustration of information exchange across operational levels is the fact that we connected the CESMO network to a Joint Electronic Warfare Coordination Cell (JEWCC). The JEWCC is responsible for coordinating all activity related to electronic warfare, and as such it is dependent on receiving information about all radio activity in the operational area. Today, the JEWCC usually receives such information late, and relies on analyzing the information after the fact. The information gathered from such analysis is then used in operational planning. However, by using the SOA infrastructure, we were able to provide the JEWCC with live information from the operational area using publish/subscribe Web services technology.

Since the planned CESMO network was radio-based, this was also a good opportunity to test SOA on a real, disadvantaged grid. Thus, it was also a goal to show that Web Services can be employed on radio networks with relatively low bandwidth, using compression.

Experiment setup

According to the present CESMO operational concept, an ESM sensor platform can have two roles:

- ordinary ESM sensor platform,
- Sigint (Signal Intelligence) Identity Authority (SIA), which is responsible for coordinating the observations made, and calculating the geolocations of detected emitters.

All platforms can have the role of SIA, but the role is often given to the platform with the best resources, both sensor and operator wise. During the experiment the SIA was collocated with the experiment authorities, without any locally connected sensors.

Planned Network Setup

The planned experiments had four main entities that were actively participating in the CESMO network: A group of (i.e. two) DA-20 aircraft, a group of (i.e. two) frigates, an experiment authority which functioned as SIA and an information consumer, i.e. a JEWCC. In addition to these four CESMO participants, a fifth entity, an army C2 system, received information from the CESMO network via an NFFI service, but did not actively participate in the operations. Figure 2 shows the participants and the planned connections between them.

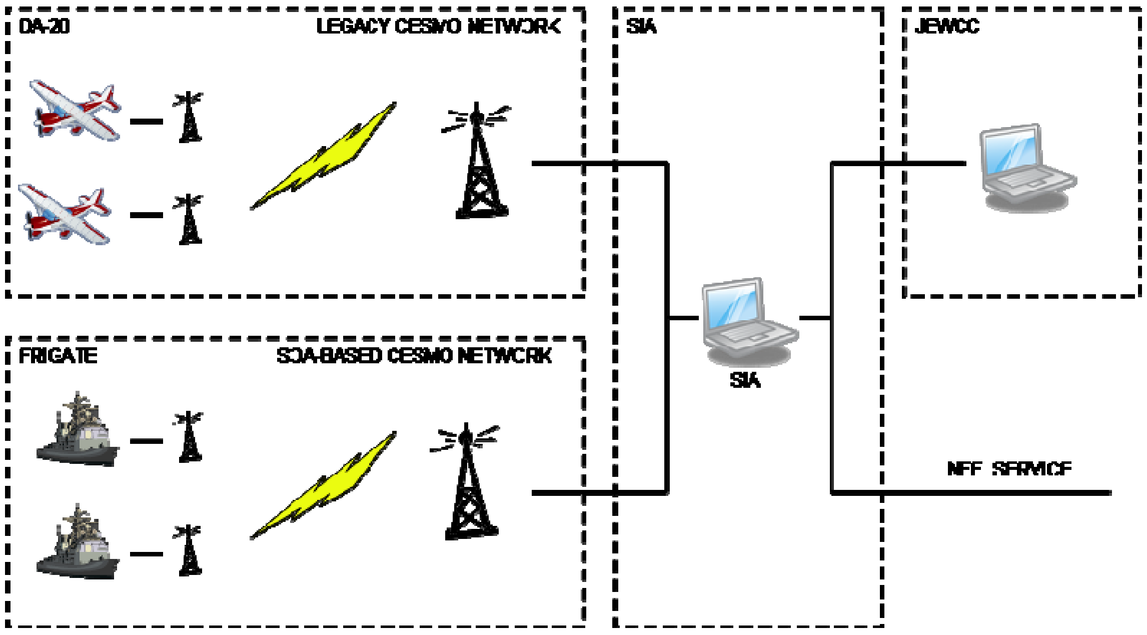

Figure 2 Planned network setup

Participants

The DA-20s were equipped with ESM sensors, and were running the above described CESMO software. However, due to the limited capabilities of the on board computer hardware, they were unable to run the SOA software. In order to connect these aircraft to the other participants, we wrapped the entire network as a Web Service. This wrapping did not only allow for the interconnection of a SOA-enabled and a non-SOA enabled network, but it also provided the opportunity to test and demonstrate how SOA can be used to bridge the gap between systems with different capabilities. The SIA, being the connection point for all the entities, performed this wrapping function.

The frigates were, according to experiment plans, going to run the same CESMO software as the DA-20s, but additionally also run a SOA software that would SOA enable these platforms. In other words, the CESMO application provided the user with a familiar interface, while the SOA software intercepted all network traffic and transmitted it using Web Services technology.

The role of SIA is often given to the platform with the best sensor and operator capabilities. In these experiments however, the role of SIA was given to the stationary experiment authority responsible for coordinating the entire experiment. This meant that the SIA did not have a local ESM sensor, and would thus have to rely on the other participants to provide it with ESM data. The SIA used the information it received for a number of purposes:

- it graphically displayed the information it received to an ESM operator that could use this information to coordinate operations,
- calculated the location of targets,
- distributed the information generated by the CESMO network to the JEWCC, and
- made CESMO information available to other systems as an NFFI service

The JEWCC, being a coordinator, understands native CESMO messages. But, it does not have a sensor of its own. This entity can gather and observe CESMO information, and can choose to interact with the active CESMO entities, i.e. the entities with ESM sensors. Using just a legacy CESMO application, the JEWCC would have to receive and visualize all tip, tune, and geolocation messages. However, it is not necessary for the JEWCC to receive all this, for example, in a particular setting maybe only the geolocations are of interest to the operators at the JEWCC. Thus, our SOA software wraps the CESMO application at the JEWCC, enabling it to selectively subscribe to only the information it sees fit, and thus reducing bandwidth usage and reducing the possibility of information overload. Furthermore, the SOA software at the JEWCC also provided an added-value function not previously available; it exported the received information into a comma-separated values (CSV) file as well as passing it on to the CESMO application for visualization. This allowed the operators at the JEWCC to use the received data in other applications as well without having to enter the data manually in order to do so.

NorTac [2] is a tactical army C2 application which is totally separate from the CESMO network. It does neither understand nor consume CESMO messages. However, thanks to our SOA solution, we can provide parts of the information from the CESMO network to NorTac through an NFFI service. Thus, NorTac can receive tracks and positions from CESMO. CESMO specific data is first received and translated to NFFI format in the SOA solution, before the information finally is presented to NorTac, which can visualize it by parsing the NFFI message. The NFFI service was provided as a push service at regular intervals.

The remaining entities, i.e. the SIA, SOA-based CESMO network and the JEWCC were fully SOA capable and were set to use the full functionality of our SOA software. In other words, a complete Web services based solution using publish/subscribe for disseminating the CESMO information, as well as performing a value-added service by providing NFFI.

Network Connections

The participants of the CESMO experiments were to be connected to the SIA with the network technologies shown in Table 1 below. The DA-20s and the frigates should use their respective installed radios, but with UIDM modems running on top of the voice channel to provide IP functionality.

Entity	**Connection to SIA**
DA-20	Radio network, UIDM IP modem, approximately 16 Kbps
Frigate	Radio network, UIDM IP modem, approximately 16 Kbps
JEWCC	2Mbit secure wired connection
NorTac	100 Mbit Ethernet[1] (local connection within the experiment authority location)

Table 1 Network connection type and bandwidth

[1] Please note that 100 Mbit is not a usual operating speed for NORTaC, but in this experiment the SIA was located in the experiment authority container complex, where NORTaC was being operated from. Thus, we used an Ethernet connection from SIA to NORTaC, which yielded a local area connection with 100 Mbit.

The UIDM adds IP packet support to the network, and can be used together with existing voice radios. The modem has several limitations that need to be taken into account when using it;

- There is limited support for IP fragments, so the application should not send very large packets (or should fragment its packets).
- The maximum transfer unit (MTU) is much lower than that which is usual for an Ethernet, so the messages/fragments must reflect this.
- There is limited buffer space in the modem. During testing we crashed the modem by sending too many packets to it in rapid succession. Thus, it is necessary to control the communication burstiness of the applications.

Final Network Setup

Due to unforeseen circumstances, the frigates were unable to take part in the UIDM-based network during the experiments. Thus, we had to change our configuration, and connected the two laptops that were supposed to be on the two frigates to the local network at the SIA location instead. That meant replacing the radio connection between these nodes with a wired connection, as shown in Table 2.

Entity	**Connection to SIA**
DA-20	Radio network, UIDM IP modem, approximately 16 Kbps
Frigate	100 Mbit Ethernet (local connection within the experiment authority location)
JEWCC	2Mbit secure wired connection
NorTac	100 Mbit Ethernet (local connection within the experiment authority location)

Table 2 Final network connection overview

This change of configuration meant that we would not be able to test the SOA solution over an actual disadvantaged grid. The SOA solution would now use only 2 Mbit and 100 Mbit connections, and the ESM information that was supposed to be gathered and provided to the SIA by the frigates was now unavailable. However, after some discussion we came up with an ad-hoc solution to this problem: One of the frigates would collect and send information to us using a manual system, whereas a local operator at the SIA would have to read these messages and input the data into the CESMO software. While not ideal, this was better than receiving no information from the frigates whatsoever. As an added bonus, we were also able to receive ESM information from an additional aircraft and a submarine that were participating in the CESMO experiments. They reported their findings via voice to a frigate, which in turn included this information in its messages to us. So, all in all, we received ESM data from quite a few platforms, even if the method of delivery did not allow us to showcase the full functionality of the SOA software.

SOA Software

The SOA software used during the experiments consisted of several components. These components are, where possible, based on existing Web service standards. The reason for this choice is twofold; the use of standards based software is critical for interoperability, and it is therefore important for us to gain experience with these standards. However, each standard used introduces overhead, often in the form of header information that must be included in messages sent and/or additional processing of information. If we are to extend Web service to all tactical levels, we must determine how well these standards perform in disadvantaged grids.

SOA Coordinator

The SOA Coordinator is the main component of our SOA software. It is written in Java, and presents the user with a graphical user interface, allowing him to control and monitor the flow of information in the SOA network. In order to enable communication between legacy software components residing on different nodes, third-party software components as well as tailor-made software were integrated with the existing software.

While the SOA Coordinator has several functions, its key role is to wrap the legacy software and connect it to a third party publish/subscribe middleware called WS-Messenger. This middleware enables publish/subscribe mechanisms in Web Service oriented systems, and implements two different standards associated with publish/subscribe and Web services, namely WS-Eventing and WS-Notification [11]. As interoperability is central to this experiment, support for both standards was one of our main motivations behind the choice of framework. By supporting both standards, WS-Messenger is capable of translating between the two, enabling messages to be published in different formats to different nodes, depending on which formats the nodes support. WS-Messenger has store-and-forward functionality, and offers retransmission of messages that failed to reach the intended subscribers.

The SOA Coordinator serves as a proxy between the CESMO software and the SOA network, and enables the user to subscribe to SOA message topics from both local and remote nodes. It also offers services to the network on behalf of the local CESMO software. The CESMO software communicates with other systems by sending XML messages. The CESMO software broadcasts these messages on the local network using UDP multicast, and the SOA Coordinator intercepts the messages and converts them to SOA messages. This process includes determining which topic they sort under, after which they are published to the local WS-Messenger. One benefit of doing this is that while the original CESMO messages are only available within one local network, the SOA software can be used to reach nodes in other networks as well.

The SOA Coordinator has a number of tabs, which provide the user with a number of options. Through the *Messaging* tab, users can monitor ingoing and outgoing SOA messages, and are provided information about their topic and content. While the *CESMO* tab similarly monitors the messages going to and from the CESMO software, the Messaging tab may also show messages going to and from other systems. The *Chat* tab enables SOA Coordinator users to see other active users on the network running the same application, and allows users to send instant messaging messages to each other.

Out of the box, WS-Messenger only supports the HTTP-protocol over TCP/IP for sending and receiving messages. While the TCP-protocol offers reliable point-to-point communication, it relies on acknowledgements, and the TCP header introduces relatively large amounts of overhead. As the system is required to perform in disadvantaged grids offering narrow bandwidths, another transmission protocol was required. Despite the lack of reliability and ordering, UDP was chosen for transporting SOA messages between the nodes. Not being dependent on ACKs/NACKs makes it suitable for low bandwidth environments. XML messages sent over UDP were first compressed using the GZIP algorithm, and encoded into

Base64 to avoid special characters and possible hardware incompatibilities. Because the SOA system was developed for use in radio networks, and because in a radio network all information is broadcasted to all participants, support for UDP multicast was added to WS-Messenger/SOA Coordinator.

Service Discovery

Service discovery was implemented into the SOA Coordinator software using a custom Java library. The library, called UdpDiscovery, was optimized for disadvantaged grids, and generates very little network traffic.

It adheres to the following principles: Small, portable Java-objects are used as information-carriers over the network. Compression of the objects was used to ensure compactness. All nodes send keep-alive messages at a specified interval (here, once every minute). However, piggybacked on the keep-alive messages is information about the nodes' perceived view of the network in the form of a list of active nodes. When nodes B and C receive a keep-alive message from A, they know the node is alive and active and update their keep-alive timestamp for node A. They also compare the received node-list with their own network information, and, if the received information is newer, update their own information about active and inactive nodes. This mechanism ensures that all nodes have updated information about the network, even in environments where packets are lost and information fails to reach all nodes. The more nodes that are part of the service discovery overlay network, the more redundancy is added to the network, increasing the robustness of the system. When a node joins the network, instead of waiting for incoming keep-alives it sends a request, which is responded to with a keep-alive message. However, in order to minimize the number of duplicate responses, the nodes that receive the request wait for a random period of time before responding, and skip the response if a response is sent by another node.
The implementation of service discovery provided the SOA Coordinator operators with information about active services in the network, enabling them to manually subscribe to services and topics they were interested in.

NFFI implementation

The NATO Friendly Force Information (NFFI) Standard for Interoperability of Force Tracking Systems (FTS), to become STANAG 5527, was used for the delivery of sensor platform locations, signal observations and emitter locations to the army's NorTac C2 system.

ESM-specific information in the signal observations and emitter locations, such as information about signal frequencies, was lost in the translation to NFFI. However, all the location information contained in the CESMO messages, such as the bearing to the emitters and platform locations were successfully translated to NFFI, and could be displayed by NorTac.

Security Software

Considering that the SOA software was based on Web services and XML, this provided a good opportunity to experiment with security standards for Web services and XML in a military environment. The following XML and Web services security standards were therefore used in conjunction with the SOA software:

- XML Signature [3]

- WS-Security [4]
- The Security Assertion Markup Language (SAML) [5]
- The eXtensible Access Control Markup Language (XACML) [6]

While confidentiality was already provided by the underlying network infrastructure, the above listed standards provided the additional benefits of end-to-end integrity, authentication, and policy-based access control.

It should be noted that the main intention of the experiment with regard to security has been to gain experience on the use of the mentioned XML and Web Services security standards in a military environment. Thus, securing the system has not been a primary goal.

A SOA Coordinator may subscribe to one or more topics from a WS-Messenger instance. In order to perform access control, we require all subscription requests to contain a SAML assertion specifying the role of the requester. The SAML assertion is obtained by the SOA Coordinator from a SAML issuer (i.e., identity provider). The SOA Coordinator includes the SAML assertion within the request message using the security header of WS-Security. The request message is also signed using XML Signature, in compliance with WS-Security.

The use of digital signatures obviously requires some type of key management to be in place. In our case, this was solved through pre-distributed keys, where the keys were stored in a Java key store at each node.

Upon reception of a subscription request at the WS-Messenger side, the request message is first subject to signature verification. If the signature verification succeeds, further access control is performed based on the role specified in the attribute statement of the SAML assertion.

The access control decision itself is taken by a XACML policy decision point (PDP) based on the applicable XACML security policy. Both the SAML identity provider and the XACML policy decision point (PDP) were run as services on a WSO2 Web Services Application Server [7].

Digital signatures were also applied to notification messages published through WS-Messenger ensuring the integrity and authenticity of these messages.

The above described security functionality was made available to WS-Messenger and the SOA Coordinator through a Java jar-file. In particular, this jar-file contained one class for outbound security and one class for inbound security. The outbound security class contains a method to add a signature to an outbound message and a method to obtain a SAML assertion and add this assertion to an outbound message. For signature verification, the inbound security class contains a method to verify the signature of an incoming message. The inbound security class also contains a method to determine if access (i.e., a subscription) should be granted or not. This method works by first obtaining the role of the requester from the SAML assertion, contained in the subscription request message, and then sends a request to the PDP point whether access should be granted or not. The XACML PDP then makes the access decision, based on the supplied role and the access control policy for the given subscription service.

Experiment Evaluation

The experimentation with the SOA solution eventually went as planned, and we were able to achieve most of the goals we had set. The only thing we were unable to test was the performance of the SOA solution over disadvantaged grids, since the frigates did not participate in the UIDM network.

Network use

We employed a publish/subscribe based system in order to reduce network load. In such a system you send one message initially to start your subscription, and after that you will receive new information in the form of notification messages later (See Table 3 for message sizes). Thus, there is no longer any need to poll the system to check if there is new data available. Eliminating the need for polling eliminates all unnecessary traffic on the network, as only useful data will be sent.

	Subscription message	Notification message envelope	Example notification message (compressed body)
Size	985 bytes	584 bytes	652 bytes
Size w/security	5074 bytes	2509 bytes	2577 bytes

Table 3 Publish/subscribe message sizes without and with the security mechanisms enabled

For an example subscription message, see Figure 3 - these messages were always the same size. The notification messages, on the other hand, could change in size from message to message depending on the payload, i.e. the size of the compressed data that was included. See Figure 4 for an example notification message. It should be noted that we only compressed the payload and not the entire message to retain compatibility with the publish/subscribe software library used. Naturally, it would be more bandwidth efficient to compress the entire message, and not just the payload. In tactical systems, one could consider using a tactical transport protocol to run Web services over, for example DMP from STANAG 4406 Annex E. Previously we have experimented with Thales' implementation of STANAG 4406 (XOMail) as a carrier for Web services in disadvantaged grids [12]. Because the messages sent with the tactical protocol are compressed before being put on the wire, this is a simple and efficient means of exchanging data in disadvantaged grids. We did not use that approach for this experiment, since there is no standardized binding from SOAP to DMP, and we wanted to utilize current Web services standards if possible.

```
<?xml version="1.0"?><S:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:wa48="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
        xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Header><wa48:To>http://193.156.33.75:5555</wa48:T
        o><wa48:MessageID>uuid:c44de0f0-0fe1-11de-9383-
        f638c3b58f08</wa48:MessageID><wa48:Action>http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe</wa
        48:Action></S:Header><S:Body><wse:Subscribe
        xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"><wse:Delivery><wse:NotifyTo><wa48:Address>ud
        p://193.156.33.75:18888</wa48:Address></wse:NotifyTo></wse:Delivery><wse:Filter
        Dialect="http://www.ibm.com/xmlns/stdwip/web-services/WS-Topics/TopicExpression/simple"
        xmlns:widget="http://widgets.com"
        xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing">widget:GeolocationType</wse:Filter></wse:Subscr
        ibe></S:Body></S:Envelope>
```

Figure 3 An example subscription message (without security)

In addition to the initial subscription messages and the recurring notification messages, there was also some traffic necessary to keep our service discovery solution up to date. The service discovery protocol needed to exchange information with all the nodes in the network at regular intervals in order to maintain an up to date view of the available services on the network. This protocol needed 500 bytes per message, and would send one such message from every SOA node every 60 seconds. Having four SOA nodes (one at the SIA, one at the JEWCC and one at each frigate) this amounted to 2000 bytes every minute to maintain the service information.

```
<?xml version="1.0" encoding="utf-8"?><S:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:wa48="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
        xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Header><wa48:To>http://193.156.33.75:18888</wa48:
        To><wa48:MessageID>test</wa48:MessageID><wa48:Action>test</wa48:Action></S:Header><S:Body><extreme
        :StringNotification
        xmlns:extreme="http://extreme.indiana.edu">PHdzbWdjbGllbnQgdG9waWM9J1BsYXRmb3JtVHlwZSc+dGVzdDw
        vd3NtZ2NsaWVudD4=</extreme:StringNotification></S:Body></S:Envelope>
```

Figure 4 An example notification message (without security). The payload is shown in red.

Considering these numbers, we can estimate the maximum amount of throughput of messages per second for our network. We base this estimate on our example notification message of 652 bytes, assuming the subscriptions have already been set up. This yields a bandwidth use with increasing number of notifications as shown in Figure 5. Considering our UIDM based network, which has a data rate 16 Kbps, we can sustain a data rate of 3 notifications per second[2]. Had we been using a 64 Kbps connection, we could have sustained 12 notifications per second. Using two lines we could have had double that, 24 notifications per second. However, in practice this number will be lower, because once every minute the service discovery mechanism will require some of the bandwidth, thus inducing some delay and requiring some notifications to be buffered for a short while. In fact, if all nodes trigger their

[2] This can be calculated as 652 bytes * 8 bits/byte = 5216 bits. UIDM supports 16 Kbps = 16000 bits per second. 16000 bits per second / 5216 bits per message ≈ 3 messages per second.

service discovery protocol at the same time, then the entire bandwidth will be consumed for the next second by service discovery traffic alone[3].

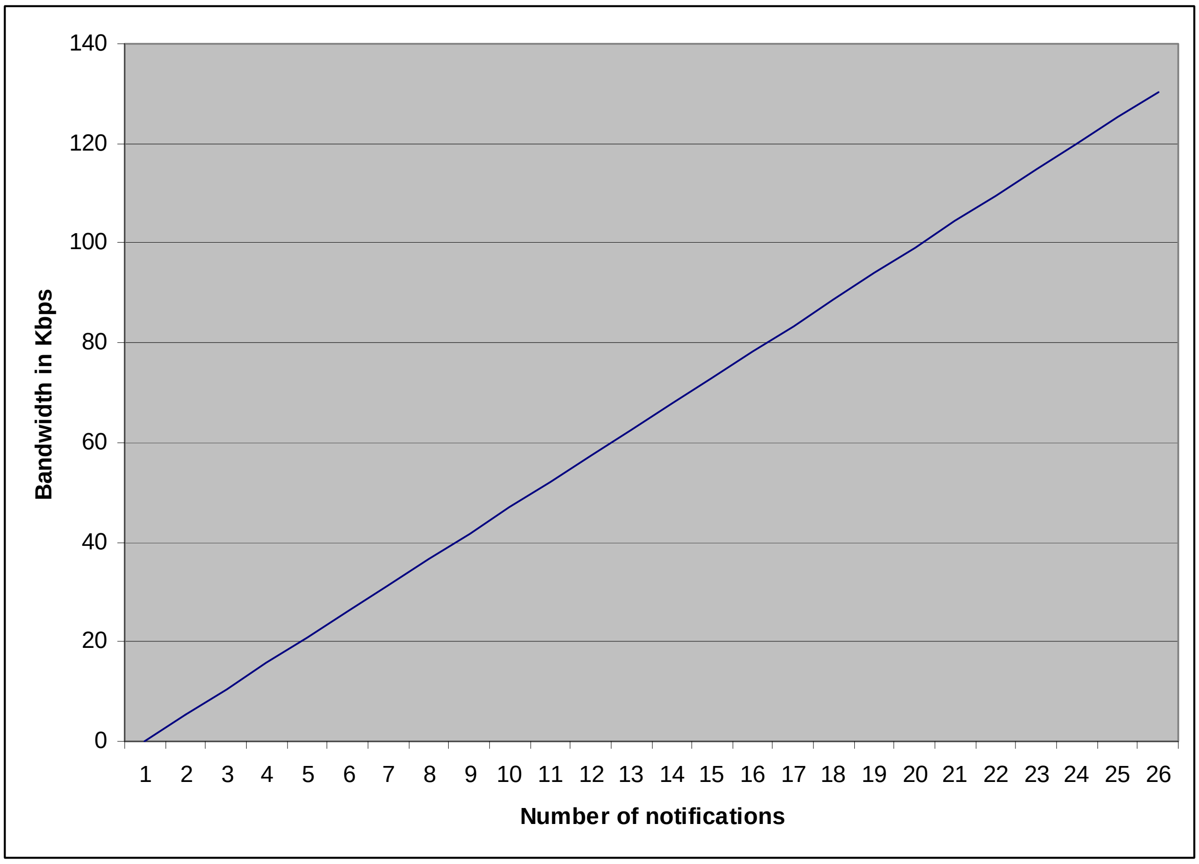

Figure 5 Estimated bandwidth use for notifications

This estimate is valid for our network configuration. For larger networks you would have a linear increase in bandwidth usage for the service discovery solution, requiring 500 bytes extra per node. The estimate regarding notifications per second would still be valid provided you use a multicast based dissemination of notifications (as we did). Using a point-to-point dissemination (i.e. using unicast to all notification recipients) would not scale well, as you would then need one notification per receiving client, instead of one notification being multicast to all clients (i.e. only one notification on the network for all clients).

As shown in Table 3, the security mechanisms add significantly to the size of such small messages. This is particularly the case for the subscription messages, where both a SAML assertion and an XML signature are included. For the notification messages the overhead is somewhat smaller, as only a single XML signature is included. Although it is possible to reduce the induced overhead by removing optional information (e.g., the KeyInfo element in XML signature) this information would then have to be known through some other means. Compressing the security header could therefore provide a more flexible solution, although this would increase the processing overhead for inspecting the security header.

[3] We have 500 bytes per service discovery message and four nodes, which amounts to 4 nodes * 500 bytes/node * 8 bits/byte = 16000 bits = 16Kbit.

Outcome

In summary, we were able to investigate

- Service discovery, which yielded a simpler setup and bootstrap of the software, and led to less need for manual configuration.
- The added value of SOA: We were able to wrap non-Web Service enabled CESMO software in a Web Service and connect it to our software. Additionally, we were able to offer several new services based on information from the CESMO system, namely a NFFI service and a CSV service. These new services allowed previously separate systems to receive information from the CESMO network.
- Using the SOA software with real data under real workloads was a benefit; previously we had verified its functionality in our lab environment. During the experiment we saw that the software could handle the data and usage patterns of an operational system.
- The benefits of publish/subscribe, in that one easily could control the flow of information between the different parts of the system. Subscriptions allowed information consumers to subscribe to only those messages which were of interest to that particular system. Published messages went out only to subscribers, and were sent asynchronously with no need to poll for new data. This reduced network load since only relevant and needed information was sent over the network at all times.
- Using compression on the XML messages meant that the data exchange of the SOA system was comparable to that of the standard CESMO messages. In other words, we got a lot of added value without introducing any significant overhead.

However, before we were able to get the software up and running properly we had to tackle some hardware configuration issues. Basically, the configuration of the network at the SIA turned out to be a problem, since it overloaded the UIDM modem. A detailed description of this problem and how we solved it is presented below. Also, we had to replace the service discovery solution in order to get our software ready on time for the experiment. Originally we started out with WS-Discovery in our initial lab tests, but we found that it did not work particularly well together with the UIDM modems.

UIDM issues

The UIDM modems had an issue with internal buffer space. If too much information was sent to the UIDM modem, *even if the packets were not addressed to it*, then the modem would choke because of too much information and hang. After that it would have to be reset in order to function again. In our initial setup at the SIA, we had all the machines constituting the SIA connected to one hub, which in turn was connected to another hub where the SOA part of the SIA and the simulated frigates were attached. The problem with using a hub is that all packets it receives on one interface is sent out again on all the other interfaces. This meant that every single IP packet that was transmitted from any of the machines at the SIA would eventually make its way to the UIDM, thus filling up and overflowing its internal buffer

After having reset the UIDM modem several times and analyzed the traffic on the network, we concluded that we would have to reconfigure the network at the SIA if we were to have a successful experiment. We replaced the hub connected to the UIDM modem with a switch, and did some minor network reconfiguration. A switch, unlike a hub, will over time learn which interface is connected to which port, and selectively transmit IP packets only on the port that has the corresponding interface connected to it. This meant that using this setup, the

UIDM could see only the packets that were intended for it (i.e. the CESMO packets to and from the DA-20s). With this setup the UIDM performed flawlessly, indicating that one should be very careful with how one goes about to configure a network.

Service discovery issues

Our initial service discovery module in the SOA software was WS-Discovery. WS-Discovery is a draft specification of a service discovery system for Web Services in local area networks [8]. The current WS-Discovery draft is tightly coupled with the SOAP over UDP specification [9]. We used a Java implementation of the mandatory parts of the WS-Discovery draft [10].

In our lab setup we encountered two issues with WS-Discovery, or rather the communication part of it; SOAP over UDP:

1. Excessive transmission redundancy jammed the network.
2. Probe matches were too large to be sent over the UIDM modem.

The first issue arose because SOAP over UDP dictates that every message should be sent four times to ensure its delivery across the network. While this may well work as intended in an office Ethernet or civil WLAN, it is not a very good idea in a tactical network with low bandwidth. All in all, WS-Discovery used too many resources in the UIDM modem, exhausting buffer space when we allowed this default behavior, and no useful information could get through. We addressed this issue by reducing the number of times to transmit each message from four to one.
The second issue was that service discovery lookups sometimes could not be sent to the requesting node. The way WS-Discovery works is that someone wanting to discover a service sends a *probe* message, and then the node(s) which provide(s) the services will send a *match* back. This match message contains all services on the node(s), resulting in a large message if the node has a couple of services. Basically, the probe matches were too big to be sent over the UIDM (due to low MTU), causing the IP packets to be fragmented in a way that could not be understood by the receiver. This meant that in order to get WS-Discovery to work over the UIDM network we would either have to implement application level fragmenting of messages or violate the WS-Discovery draft specification by allowing a *probe* to be answered by several *match* messages, thus putting one service description in each reply and in this way circumventing the UIDM MTU problem. Obviously, we would not be able to get WS-Discovery as it was specified in the draft to work. We were also pressed for time, and thus chose to use a simpler, well-tried service discovery mechanism for use during the experiments.

Security experiment execution

The security experiments performed illustrated how civilian standards for SOA security can be used in a military setting. Some issues require further discussion however.

Although we were able to utilize all the intended XML and Web Services security standards successfully during the experiment, we experienced problems related to XML Signature. More specifically, we were only able to verify the signatures when UDP was used as transport protocol by WS-Messenger. When HTTP was used as transport protocol, signature verification consistently failed and had to be disabled. The reason for this was apparently that WS-Messenger performed XML deserialization and serialization of the messages when HTTP was used, which caused changes to the layout of the XML that could not undone by XML canonicalization. In particular, XML canonicalization retains all whitespace that is between a start/end tag pair. Therefore, any attempts by an intermediary to introduce blanks or newlines (e.g. in order to improve readability) will change the checksum of the document and break the signature. Consequently, an XML signature is broken by layout changes such as changes in indentation.

The availability of the security services is also a critical point that needs to be taken into consideration. In our case, the security services (i.e., the SAML identity provider and the XACML PDP) were replicated at each location. An advantage of such local replication is that the services remain available even in the case that the communication between locations fails. Considering that the unavailability of a security service may prevent access to other services, this is an important consideration. On the other hand, if the information relied on by the security services is of a dynamic nature, security will likely be affected unless consistency can be ensured. In addition, there are also bandwidth considerations that may play a role between a more centralized or decentralized/replicated architecture, depending on the usage patterns and the frequency of updates.

As mentioned previously we also relied on predistributed keys for the experiment. In a scenario with few participants and limited bandwidth this may provide a good solution. However, such a solution clearly does not scale to a high number of participants, in which case a more sophisticated key management scheme would be required.

Conclusion

In this paper we have presented the SOA CESMO experiments performed at a national exercise in late 2008. The goal of our participation was, by using CESMO as a case, to show that introducing SOA into an existing operational system can contribute to automate and speed up existing processes, as well as add value through new and extended functionality.

In summary, we were able to investigate

- Service discovery, which yielded a simpler setup and bootstrap of the software, and led to less need for manual configuration.
- The added value of SOA: We were able to wrap non-Web Service enabled CESMO software in a Web Service and connect it to our software. Additionally, we were able to offer several new services based on information from the CESMO system, namely a NFFI service and a CSV service. These new services allowed previously separate systems to receive information from the CESMO network.
- Using the SOA software with real data under real workloads was a benefit; previously we had verified its functionality in our lab environment. During the experiment we saw that the software could handle the data and usage patterns of an operational system.
- The benefits of publish/subscribe, in that one easily could control the flow of information between the different parts of the system. Subscriptions allowed information consumers to subscribe to only those messages which were of interest to that particular system. Published messages went out only to subscribers, and were sent asynchronously with no need to poll for new data. This reduced network load since only relevant and needed information was sent over the network at all times.

- Using compression on the XML messages meant that the data exchange of the SOA system was comparable to that of the standard CESMO messages. In other words, we got a lot of added value without introducing any significant overhead.
- We were also able to demonstrate aspects of XML Security, which can give end-to-end security for Web services.

The experiment provided us with an opportunity to show the flexibility of SOA. The ad hoc reconfiguration of the network that we had to do (as discussed in the sections on *"Planned Network Setup"* and *"Final Network Setup"*) did not prevent our SOA software from functioning. The dissemination of information between nodes could be controlled at runtime by establishing and terminating subscriptions, allowing us to adapt to the changing conditions.

References

[1] NORCCIS-II
http://www.c2is.net/nii/index.html

[2] NorTac-C2IS
http://www.kongsberg.com/eng/**kda**/products/Armyc2is/**NORTaC**-C2IS/

[3] D. Eastlake, J. Reagle, and D. Solo "XML-Signature Syntax and Processing," W3C Recommendation, 2002.

[4] A. Nadalin et al., "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," OASIS Standard, 2006.

[5] S. Cantor et al., "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0," OASIS Standard, 2005

[6] Tim Moses, "eXtensible Access Control Markup Language (XACML) version 2.0," OASIS Standard, 2005

[7] WSO2 Web Services Application Server - open source, enterprise-ready, Web services engine based on Apache Axis2
http://wso2.com/products/create/wso2-web-services-application-server-wsas/

[8] Web Services Dynamic Discovery (WS-Discovery) draft specification, 2005
http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf

[9] SOAP-over-UDP, 2004
http://specs.xmlsoap.org/ws/2004/09/soap-over-udp/soap-over-udp.pdf

[10] A WS-Discovery implementation written in Java
http://code.google.com/p/java-ws-discovery/

[11] WS-Messenger (WSMG)
http://www.extreme.indiana.edu/xgws/messenger/

[12] R. Haakseth et al., "Experiment report: "SOA Cross Domain and Disadvantaged Grids" – CWID 2007", ISBN 978-82-464-1272-6, FFI-Report 2007/02301
http://rapporter.ffi.no/rapporter/2007/02301.pdf