The University of Kansas

INFORMATION
& TELECOMMUNICATION
TECHNOLOGY CENTER
The University of Kansas

Technical Report

Transportation Security SensorNet: A Service Oriented Architecture for Cargo Monitoring

Martin Kuehnhausen and Victor S. Frost

ITTC-FY2010-TR-41420-22

April 2010

Project Sponsor:
Oak Ridge National Laboratory
Award Number 4000043403

Table of Contents

List of Figures

Transportation Security SensorNet: A Service Oriented Architecture for Cargo Monitoring

Martin Kuehnhausen, *Graduate Student Member, IEEE* and Victor S. Frost, *Fellow, IEEE*

***Abstract*—This paper describes a system architecture for a *Transportation Security SensorNet* (TSSN) that can be used to perform extensive cargo monitoring. It is built as a *Service Oriented Architecture* (SOA) using open *web service* specifications and *Open Geospatial Consortium* (OGC) standards. This allows for compatibility, interoperability and integration with other *web services* and *Geographical Information Systems* (GIS).**

The two main capabilities that the TSSN provides are remote sensor management and alarm notification. The architecture and the design of its components are described throughout this paper. Furthermore, the specifications used and the fundamental ideas behind a SOA are explained in detail.

The system was evaluated in real world scenarios during field trials and performed as specified. The alarm notification performance throughout the system, from the initial detection at the *Sensor Node* service to the *Alarm Reporting* service, is on average 2.1 seconds. Location inquiries took 4.4 seconds on average. Note that the majority of the time, around 85% for most of the messages sent, is spent on the transmission of the message while the rest is used on processing inside the *web services*.

Finally the lessons learned are discussed as well as directions for future enhancements to the TSSN, in particular to security, complex management and asynchronous communication.

***Index Terms*—Telemetry, Transport protocols, Intermittently connected wireless networks, Communication system software, Data communication, Software engineering**

I. Introduction

THE theft and tampering of cargo are common problems in the transportation industry. According to Wolfe [1] the “FBI estimates cargo theft in the U.S. to be $18 billion” and the Department of Transportation “estimated that the annual cargo loss in the U.S. might be $20 billion to $60 billion”. Wolfe [1] also gives good reason to believe that the actual number may be even higher than $100 billion because of two reasons. First it is assumed that about 60 percent of all thefts go unreported and second the indirect costs associated with a loss are said to be three to five times the direct costs.

With the advances in technology, this problem has evolved into a cat-and-mouse game where thieves constantly try to outsmart the newest cutting edge security systems.

In terms of securing cargo, there are usually two aspects: first ensuring the physical safety of the cargo and second monitoring and tracking it. The latter especially has become of more interest as of late because many shipments cross national borders and cargo may be handled by a multitude of carriers. All of this leads to a huge demand for tracking and monitoring systems by the cargo owners, carriers, insurance companies, customs and many others.

This paper is part of a series that describe the design, various components and conducted experiments of the TSSN. In particular we focus on the software architecture here and refer to papers that deal with the other parts of the TSSN in the following. [2] gives an overview of the hardware utilized and describes in detail truck trials and a short haul train trial. [3] presents a new and flexible approach to deal with challenges such as intermittent and low-bandwidth communication in mobile monitoring environments and a long haul train trial in Mexico. Furthermore [4] discusses a framework for analyzing and visualizing SOAP messages to overcome the challenges of complexity and disparity that web service monitoring and management approaches face. Security associated with the TSSN and specifically issues that arose when integrating elements from the Web Services Architecture (WSA) led by the World Wide Web Consortium (W3C), specifically publish/subscribe communication and service security are described in [5].

Here, a framework is introduced which builds on open standards and software components to allow “monitoring cargo in motion along trusted corridors”. The focus lies on the use of a *Service Oriented Architecture* and *Geographical Information System* specifications in order to allow an industry wide adoption of this open framework.

In the following we discuss the problems of proprietary systems, the advantages of open standards and the approach of using a *Service Oriented Architecture* in the transportation industry. We introduce the design and architecture of the framework and explain the individual components as well as the software parts and specifications that are used in the implementation.

The discussion of proprietary systems in contrast to open standards in the following section provides an overview of the challenges that trade and shipping partners face. It explains why it is important to design an open system that is based on standards. Some of the main advantages are a decrease in cost, more efficient shipment management, and enhanced visibility and tracking capabilities. This paper presents the architecture of the TSSN that was implemented to show that such an open system can be built and deployed successfully.

M. Kuehnhausen and V. S. Frost are with the Information and Telecommunication Technology Center, The University of Kansas, Lawrence, KS, 66045, USA; Corresponding author: mkuehnha@ittc.ku.edu

This work was supported in part by Oak Ridge National Laboratory (ORNL)—Award Number 4000043403. This material is also partially based upon work supported while V. S. Frost was serving at the National Science Foundation.

II. Problem area

In order to address the problem of cargo security, the *Transportation Security SensorNet* project has been created. Its goal is to promote the use of open standards and specifications in combination with web services to provide cargo monitoring capabilities. The main question is the following:

> "How can a *Service Oriented Architecture*, open standards and specifications be used to overcome the problems of proprietary systems that are currently in place and provide a reusable framework that can be implemented across the entire transportation industry?"

The three main aspects of this question are discussed next.

A. *Proprietary Solutions*

Current commercial systems in the transportation industry are often proprietary. This is because a lot of effort is spent on research and development in order to create *intellectual property*. The assumption is then that as long as the competitors do not have access to the system and its protocols that *intellectual property* is safe and provides a competitive advantage. Another common "benefit" of keeping the systems closed is the perceived additional security since in order to successfully attack the system its implementation and protocols have to be *reverse engineered*.

The problem with this is that these advantages are often one-sided and lead to stove pipe systems provided by a single vendor. Once a proprietary system has been implemented it has to be maintained. What happens if a customer that uses the system invested a lot of money into a its infrastructure and the training of its employees and the company that provides the system releases a new version of it which of course costs money again. The customer has several choices:

1) Upgrade: Throughout the literature this is often considered the most expensive option because of the cost for the upgrade to the new version and the additional training to the employees that has to be provided. The benefits of upgrading are the use of new technology, potential gains in efficiency through new features and the latest bug fixes.

2) Do Not Upgrade: By many regarded as the most cost efficient solution, choosing not to upgrade compromises new features and updates for the ability to save costs. An approach that is taken by some companies is the *skip a version* technique. This allows companies to plan better as internal processes and systems often have to interoperate and need to remain compatible to each other.

3) Change Vendor: In this situation, the new version of the system that is provided by company A does not provide the necessary features or is simply too expensive. Furthermore, a different company B offers a similar product with more features or for less money. The move to the new system is now dependent on the following things: How big are the estimated savings and what are the direct and indirect costs of the transition? It often happens that after careful consideration the costs outweigh the estimated gains and the customer goes back to considering whether or not to simply upgrade. If a transition is made, the process could be time consuming and turn out to be more complicated than expected.

Picture this extreme case as well. What happens if the vendor goes out of business? All of the sudden, the short-term goal is to maintain support for the system and to keep it running while in the long-term to look for a suitable replacement and be forced to transition. Even if this case does not happen the dependency on the vendor can be crucial. If the system has errors or a particular enhancement is desperately needed, the vendor decides what to do about it. For big companies that are major customers this may not be such a big problem because they often get preferential treatment. But for small and medium businesses the wait might be too long and lose them customers and revenue.

The main point here is that many customers are locked into proprietary solutions that are incompatible with similar solutions offered by competitors. In a 2003 survey by the Delphi Group [6] it was found that 52% of developers and 42% of consumers see standards enabling the "approval of projects otherwise threatened by concerns over proprietary system lock-in". Furthermore, an overwhelming 71% of developers and 65% of consumers feel that the use of open standards "increases the value of existing and future investments in information systems".

The problem of non-interoperability with regard to geospatial processing is the topic of a paper by Reichard [7]. Because *Geographical Information Systems* are often immensely complex, companies that invest heavily into this area often only support their product. As described in the sample scenario, this leads to a lack of coordination among entities such as the *Federal Emergency Management Agency* (FEMA), the *National Transportation Safety Board* (NTSB) and the *Environmental Protection Agency* (EPA) because of the inability to share vital information which is the key to fast decision making and data analysis

B. *Variety of Open Standards*

1) Standards principles: The idea of open standards and specifications is to define *interfaces* and *protocols* that can be used as references for the implementation of a system. There are many standards committees and industry groups that aim to define them, most often focused on a particular area. Some of the most well-known ones include the *World Wide Web consortium* (W3C), the *Organization for the Advancement of Structured Information Standards* (OASIS), the *International Telecommunication Union* (ITU) and the *International Organization for Standardization* (ISO).

The main principles that govern the development of standards are usually the same across all organizations. The following is an overview according to ISO:

a) Consensus: All parties that are affected by the proposed standard get the chance to voice their opinions. This includes initial ideas and continues with feedback and comments during the standardization process.

b) Industrywide: The idea is to develop global standards that can be used worldwide by entire industries.

c) Voluntary: The standardization process is driven by the people that are interested in it and that see its future benefits across a particular industry. It is often based on *best practices* that are already commonly in use.

2) Aspects of Open Standards: The importance of open standards is emphasized in a paper by McKee [8]. It provides the evolution and success of the Internet as the "perfect example" for the use of open standards. In particular it explains that since the Internet is based upon communication and communication means "transmitting or exchanging through a *common system* of symbols, signs or behavior", the process of standardization can basically be seen as "agreeing on a *common system*". The other parts of the paper are focused on how *openness* can help *Geographical Information Systems* (GIS) but many of the points mentioned apply to open standards in general.

In particular the following aspects are associated with open standards:

a) Compatibility: This includes the ability to share data across vendors and systems in a uniform and non-proprietary form. It allows processes to use essentially the same data in order to perform their specific task without the need of costly conversions or interpretation errors. Most *common* formats are also backward compatible which means that no particular version of the system is needed to interpret the data. Only a certain subset of functionality might be provided when using in older versions though. Another advantage of open formats is the fact that even if a particular version of a format is completely outdated and only used in legacy systems, its specification is still accessible to everyone. Hence systems can still be designed to use the format.

b) Freedom of Choice: A major problem of proprietary solutions that was described earlier was the *vendor lock*. Once a customer implements a proprietary system and builds its infrastructure around it, choices in the future are limited. Open standards by definition are vendor independent. Furthermore many of them support a broad variety of implementation scenarios. These implementations often are not even limited to a particular platform, operation system or programming language. This is especially true for most of the web standards.

c) Interoperability: Through the use of clearly defined interfaces, standards dramatically enhance interoperability. The standards that define interface specifications do not provide a specific implementation but provide references to *best practices* and *implementation patterns* instead. Companies choose what kind of system implementation they prefer. This allows them to make use of existing infrastructure and capabilities that might otherwise have to be changed when using a proprietary system. The uniform access to functionality and data enables companies to connect a multitude of systems and make more use of them. Also, in case one part of the system has to be replaced, another one that simply provides the same interface can take its place. This allows great flexibility in terms of the overall system design.

d) Leverage: For companies the standardization of concepts, frameworks and common approaches provides a number of benefits. Since research and development can be extremely cost intensive, companies want to make sure there is a guaranteed *return on investment* for them. Open standards do not necessarily lead to increased revenue but they do provide insurance to the companies that they are on the "right" track and what they implement is actually used industrywide. This is very important because customers are aware that when they purchase a system from company A that uses a proprietary or non-standard implementation they might become a victim of *vendor lock*. Acquiring a system that is build on open standards allows them to choose the best and most cost effective solution from a variety of independent implementations. Another advantage is that once different implementations by the main vendors have been established, there is room for custom solutions by smaller vendors, often in the form of extensions or plugins.

e) Open Source: The biggest benefit of using open standards is that fact it leads to innovation. This is because everybody can contribute, suggest enhancements, outline *best practices* and address mistakes. In terms of software this approach is often referred to as *open source*.

However, there are several problems that can be associated with non-proprietary systems. Implementations are based upon the interpretation of the standards which may differ significantly. Furthermore, some implementations only support a subset of the original specification, are slower than the reference implementation or use incompatible sub systems.

C. Service Oriented Architecture

The concept of information processing and sharing across various applications using *web services* is the main focus of this paper. The basic idea is to define components of a system as *services* and users as *clients* that can retrieve data from them. Note that interaction between *services* is done using *embedded clients*. The *services* take care of things such as information processing, data analysis and storage. With all business logic embedded into *services* and interaction between them clearly defined using open standards an infrastructure is built that is called the *Service Oriented Architecture* (SOA).

The Internet allows the following two things that are relevant to information processing: a common means of communication and the ability for efficient information sharing. There exist many standards on how to transmit, receive, encode and decode data. SOA builds on top of them to provide new specifications that enable the design, implementation and use of *web services*. Through these *web services* companies, government agencies and others have the ability to share and process information in a uniform manner which cuts costs, time and resources and improves efficiency.

Now why is SOA such an "enabler"? What is possible now that was not possible before? According to Irmen [9] *automation* and *efficient communication* with partners are the two most important things in *supply chain management* which represents the core of the transportation industry. Let us take a look at how the *Service Oriented Architecture* addresses both of them in regard to the individual topics outlined in [9].

1) Automation: A vital part in transportation is the *screening* process. Companies that transport goods must ensure safety and therefore check all parties involved in the trade.

An important aspect of this is the use of a *denied trade list* which lists items and companies that are not allowed to import or export into specific countries. With the reduction in manual labor and transition to a *web services* based system that automatically performs these checks, efficiency could be greatly increased.

A closely related topic is *accountability*. Who is responsible if something goes wrong during the trade process? Since goods are often handled by many different parties, it must be possible to monitor the location of cargo and handovers tightly. This is especially important in cases of tampering or even theft of the cargo.

Furthermore, agencies and customs more and more require *electronic trade information* instead of paper documents in order to track trade. Because of different formats and legacy applications that are often unable to provide this information in its entirety, additional resources have to be allocated in order to remain compliant with current practices. *Web services* and open standards can overcome this problem with uniform interfaces and common data formats.

Having the ability to *monitor* the location not just for perishable goods but also for high value goods is of great importance in the transport chain. Current processes should be able to automatically route cargo based on its needs and cost effectiveness.

Irmen [9] also points out that “the lack of integration between products causes users to deal with multiple systems having disparate data and non-uniform input and output” and calls for the use of a single platform. Using the *Service Oriented Architecture* this “call” becomes less necessary because it is platform independent and at the same time able to provide integration of multiple systems and standardized data formats.

2) Efficient Communication: Building a virtual network among the parties involved in the trade process establishes efficient means of communication. It allows the *coordination* between otherwise disparate entities that is essential to provide cost effective and reliable shipping of cargo. The Internet provides the communication layer but it is the standards of *web services* that enable the integration of different systems.

Irmen [9] mentions the *Software-as-a-Service* (SaaS) approach which allows software to run on a per-use basis without the costs of complex hardware infrastructure. This works very well with SOA as the interfaces defined by those services are often *web services* interfaces that are essentially part of SOA.

Security within the transportation industry plays a big role because trade data is to be kept confidential and only distributed on a need-to-known basis. This puts an additional burden on the parties that are involved, as the parties must exchange data confidentially at each point of interaction. If open standards are used for this, *security* is implemented based on interfaces and policies that are easy to manage.

In order to manage the transportation chain in its entirety, a *global view* is often needed. This is problematic since individual parties often only deal with their respective neighbors. Using open standards and the *Service Oriented Architecture* approach each party could provide an uniform information interface that is accessible to other parties in the chain. This allows consistent reporting, monitoring and analysis at each step during the shipping process.

The reporting part especially has gained more attention over the past years as the focus has shifted towards more ethical and socially responsible business practices. *Accountability* coincides with this *social visibility* and therefore improvements in monitoring cargo not only lead to increased revenue on the business side but better public relations as well.

Overall the paper by Irmen [9] gives excellent reasons for open systems in terms of accountability, coordination, scalability and cost, these important aspects that need to be taken into consideration when designing an architecture such as the *Transportation Security SensorNet*.

III. Related Work

In the following sections related work that is relevant to various aspects of the *Transportation Security SensorNet* such as *Service Oriented Architecture*, web services, communication models, the *Open Geospatial Consortium* specifications and sensor networks is analyzed.

A. Microsoft - An Introduction to Web Service Architecture

Cabrera et al. [10] outline concepts that led to the implementation of *Service Oriented Architectures* and development of the *web services* specifications that surround them and are used by the TSSN. A lot of the main approaches have been standardized in various committees and organizations by now but were only in the early stages when Cabrera et al. first discussed them.

B. Adobe - Service Oriented Architecture

An Adobe technical paper by Nickul et al. [11] outlines general architecture approaches that can be taken when transitioning business processes to the *Service Oriented Architecture*. It mentions a widely used technology called the *Enterprise Service Bus* (ESB) that provides a standardized means of communication for all services that connect to it. For the TSSN this is of importance when it comes to asynchronous communication as the *Java Message Service* (JMS) uses queues that are on the ESB for message exchanges (see [3] reference, IV-A6).

In addition to the basic *Request-Response*, several other *message exchange patterns* that go beyond the standardized ones are described. A *registry* keeps track of service metadata. The *service provider* is responsible for updating it whenever a change occurs and the *service consumer* subscribes to the *registry* for any of these changes. The metadata that is provided is then used to configure a *service client*. Hence, the client can issue *requests* and receive *responses*.

The TSSN essentially uses a very similar approach with the UDDI. Web services automatically register with the UDDI when they are started and clients are able to use specific services by looking them up in the UDDI.

C. Open Sensor Web Architecture

An approach to implement the proposed standards of the *Sensor Web Enablement* (SWE) that are described in [12] is outlined by Chu et al. [13]. A more detailed definition of the system and its core services is provided in the thesis by Chu [14]. The system is called *NICTA Open Sensor Web Architecture* (NOSA) and is focusing on the combination of sensor networks and distributed computing technologies.

The TSSN uses a similar approach but has some significant differences. The goal of both implementations is to integrate a sensor network into a web services architecture using open standards. NOSA uses a sensor application that is tightly integrated into the *Sensor Operating System* and then provides sensor data and control to web services in a non-standard format. TSSN on the other hand implements sensor management and monitoring functionality inside a single service, the *Sensor Node* (see IV-C1) and allows different sensors to be "plugged in". This allows other services to use standard web service interfaces and SOAP messages in order to access sensors.

Furthermore, the web services used by NOSA are implemented manually according to the OGC specifications which causes them to be limited as not everything that is specified is also implemented. In contrast, the TSSN uses automatic code generation (see IV-A1d) that enables it to use all OGC specifications. Since their elements and interfaces are generated the only thing that has to be implemented is functionality. This approach significantly reduces development efforts.

D. Electronic Freight Management

The Electronic Freight Management (EFM) initiative [15] is a project that focuses on the improvement of communication between supply chain partners using web technologies. One of the main goals is to provide a common and open technology platform for sharing cargo information among smaller and medium size trade partners. The idea is that the information is only entered once and then shared among members of the supply chain.

EFM because of its SOA approach provides a common electronic communication platform that maintains cargo related information on a web service basis. This information is then shared with authorized users while digital certificates and web service security ensure data integrity and confidentiality. The key benefit here is the improved visibility of shipment information which enables all supply chain members to perform their processes more efficiently and plan ahead better [16]. The data exchange is standardized and based on the Universal Business Language (UBL). Furthermore each individual transaction is uniquely identifiable by a Unique Consignment Reference (UCR).

The TSSN approach is similar but deals in particular with cargo monitoring in mobile environments. The Trade Data Exchange (TDE) as described later is responsible for managing and sharing shipment information.

E. Globus - Open Grid Services Architecture

Globus is an architecture that is based on grid computing. It focuses on providing capabilities as services in a grid environment using standard interfaces and protocols. An initial paper by Foster et al. [17] gives an overview of the architecture and design decisions. In particular, Globus supports "local and remote transparency with respect to service location and invocation" and "protocol negotiation for network flows across organizational boundaries". Its service approach is similar to the *Service Oriented Architecture* that is used by the *Transportation Security SensorNet*. Additionally, security concepts that work inside a grid are applicable to SOA and vice versa.

The current architecture of Globus is still based on the same principles that were initially described by Foster et al. [17]. The combination of custom components and web services components provides an architecture for security, data management, execution management, information services and a common runtime in a grid environment.

In contrast to the TSSN, Globus makes use of web service specifications in some of its components but also provides custom implementations and interfaces as for service discovery and notifications. The TSSN uses web services specifications and OGC standards almost exclusively which ensures standards compliance and compatibility. For service discovery the UDDI [18], [19] is used and for notifications *WS-Eventing* [20].

F. Service Architectures for Distributed Geoprocessing

A research article by Friis-Christensen et al. [21] outlines the implementation of an application that analyzes the impact of forest fires using web services. The main focus is the transition from a client application to a flexible web services architecture using *Open Geospatial Consortium* specifications. The components include multiple data sources that are made available through data access services like the *Web Map Service* and the *Web Feature Service*. A geoprocessing service performs the analysis of the data and provides it to a client. Furthermore a discovery service serves as the registry for all services and their metadata.

The prototype implemented uses synchronous communication in between services. The problem in this case is that the actual processing can take quite a long time. In the future the authors want to transition to an asynchronous communication model that is similar to the OGC *Web Notification Service*.

In addition, it is pointed out that even though standardized interfaces allow for a combination of services which provides flexibility, the transport of high volumes of data is often not feasible in geoprocessing scenarios which can lead to highly specialized but not very reusable services.

The implementation is interesting in the sense that it exclusively uses OGC specifications which makes it compatible to other *Geographical Information Systems*. The TSSN aims to be OGC compliant as well but includes specifications that deal with sensor networks such as the *Sensor Observation Service* and the *Sensor Alert Service*, something that Friis-Christensen et al. [21] does not address.

G. Web Services Orchestration

The problem of reusability of services and "next generation challenges" was addressed by Kiehle et al. [22]. The idea here is to increase transparency and reusability by splitting processes into smaller more reusable processes and utilizing a work flow management system called *Web Services Orchestration*. This is especially important for the integration of the *Transportation Security SensorNet* into systems used in the transportation industry. Its modular design and architecture allow single components to be reused and and information flows to be created.

The *Web Processing Service* specification describes how services can be arranged and combined into *service chains* that form a process. Two alternatives are commonly used in order to achieve this. A *Web Processing Service* can be set up to combine and "encapsulate" other individual web services and therefore provide the desired abstraction. However, the best way to define work flows is using the *Business Process Execution Language* (BPEL). BPEL enables complex *service chains* to be defined without the need for custom and potentially not reusable *Web Processing Services* that just "encapsulate" services.

H. Summary

The related work addresses the following key technologies that play an important part in the *Transportation Security SensorNet*:

1) Service Oriented Architecture: The development of SOA and its web services specifications has come a long way but is still far from over. Even though specifications exist, organizations and businesses often implement components that are similar to the specification but not compliant. As discussed before, this is the case for service discovery and notifications in Globus. Two common reasons behind this are the following. First, the specification may be available but there are hardly any reference implementations that can be used. Second, extensions to the specification that are necessary for a particular implementation or in a specific environment such as the grid are not covered by the standard.

2) Open Geospatial Consortium: The OGC specifications are often complex and there is significant development effort necessary to implement the elements, interfaces and functionality they define. Automatic code generation as described IV-A1d and used by the TSSN can facilitate their implementations but is not used very often.

3) Sensor Networks: The implications on communication models that sensor networks have, in particular asynchronous message exchanges, are often ignored in web service architectures. As seen in NOSA, the focus is on the implementation of a subset of OGC standards for a particular sensor network, but the link to an overall SOA seems to be missing. It is evident that current systems seem to lack the combination of SOA, OGC specifications and sensor networks. The TSSN combines all these technologies and bridges the gap between implementations that just deal with SOA and OGC specifications and systems that use OGC standards in sensor networks.

IV. Proposed Solution

A. Overview

This section describes the architecture of the *Transportation Security SensorNet* (TSSN). It provides an in-depth discussion of design aspects and the implementation.

1) Service Oriented Architecture:

> "Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains." [23]

Building a "Service Oriented Architecture for Monitoring Cargo in Motion Along Trusted Corridors" makes sense. According to a study by the Delphi Group [6], companies that collaborate usually request compliance for the following standards: XML 74%, J2EE (Java) 44% and SOAP 35%. The architecture used for the implementation of the TSSN utilizes all three technologies by separating functionality into *web services*. This allows for high flexibility and is cost effective.

Haas et al. [24] early on proposed various *models* for web service architectures. The *Message Oriented Model* focuses on message relations and how they are processed. An approach that centers around resources and ownership is the *Resource Oriented Model*. The *Policy Oriented Model* defines constraints and focuses on security and quality of service. Ideas from all these models have been combined with the *Service Oriented Model* into what has become SOA. Of the proposed models it has been the most widely implemented.

A book that provides an excellent overview of Java and *web services* is written by Kalin [25]. Note that SOA by definition is programming language and platform independent. It is built on the basis of requests and responses and the independence of *web services*. The choice to use Java for the implementation was made because the TSSN is built on top of previous research on the *Ambient Computing Environment for SOA* by Searl [26] which is written in Java.

The main components of the TSSN are sensor management and alarm notifications. An overview of the services and relevant message exchanges is shown in Figure 1.

The *Trade Data Exchange* (TDE) (see IV-E) provides shipment, route, logistics and relevant cargo information. It is managed externally and used by the system only through its specified interface. The *Virtual Network Operation Center* (VNOC) (see IV-D) is responsible for the processing of sensor data and alarms. One of the major capabilities that it provides is alarm notification. The *Mobile Rail Network* (MRN) (see IV-C) deals with the actual management of sensors on a mobile platform, e.g. a train. *Web services* at the *Mobile Rail Network* capture sensor data from the sensors and "preprocess" that data. A detailed description of each individual service is provided later in this section.

The architecture consists of web services that are separated into *service clouds*. These *service clouds* represent the different geographically distributed locations (e.g. Overland Park, KS for the TDE; Lawrence, KS for the VNOC and on a moving train for the MRN) where services are deployed and are shown in Figure 2.

The *web services* are developed according to the *web service* specifications and the standards provided by the OGC. This means that they aim to be standards compliant. Since the OGC specifications are at times very complex, the *Geography Markup Language* (GML) for example defines over 1000 elements, the basis for the framework was implemented using

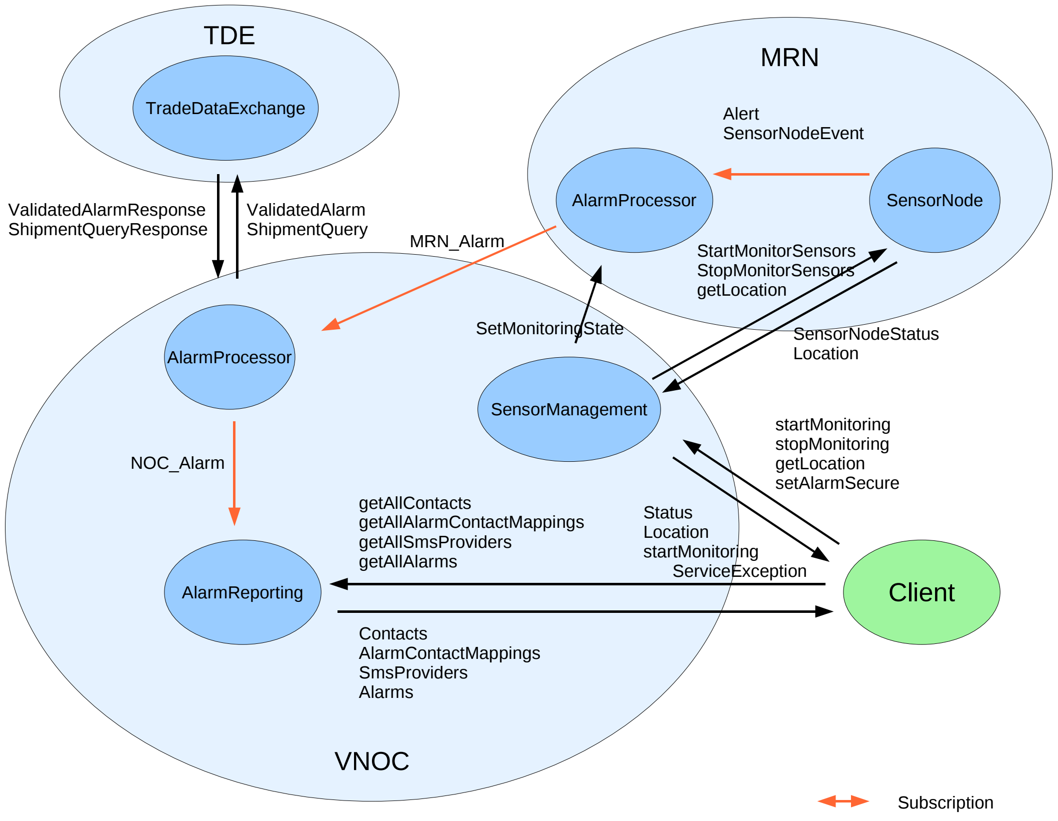

Fig. 1. Service message overview

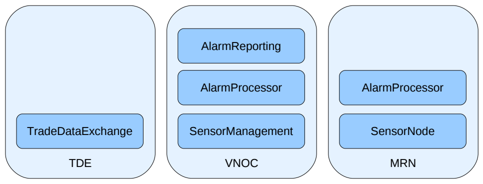

Fig. 2. Service cloud

custom interface definitions first and adding the OGC ones later. This enabled fast prototyping and testing of the system.

The following sections explain in-depth the approaches and technologies used in the architectural prototype and implementation of the TSSN.

a) Ambient Computing Environment for SOA: The infrastructure described by Searl [26] called *Ambient Computing Environment for SOA* (ACE_SOA) forms the basis of the implementation of the TSSN. It provides a complete SOAP stack using *Apache Axis2* and a variety of other useful programs that assist in the development of a SOA. ACE_SOA deals with multiple ownerships and federations that provide *web services*. In particular it covers the following aspects:

- *Service Discovery* across different federations
- *Authentication* of clients and services
- *Authorization* of clients and services
- *Subscriptions*

The implementation of the capabilities provided is based on *Apache Axis2* and *web service* specifications. It is explained in detail in the following sections.

b) Apache Axis2: Apache Axis2 is a software stack that allows the development and running of *web services* and clients. Its architecture as described by Chinthaka [27] consists of the following main components:

AXIs Object Model (AXIOM): AXIOM is an XML object model that aims for high performance while requiring low amounts of memory. The idea behind it is the application of a *pull parser*. This allows objects to be built from XML only up to the information that is needed by the user while the rest of it is *deferred*. The advantage of this is that the memory that an object requires is significantly reduced. Furthermore, this approach also increases performance since the entire object model does not have to be constructed before information can be retrieved, which is the case in the Document Object Model (DOM) parser.

Extensible Messaging Engine: Axis2 provides a very modular architecture that allows for a variety of different implementations of *web services* as long as they adhere to certain specifications. A variety of transports such as HTTP, SMTP, JMS and TCP can be used for message exchanges. Inside the *engine* each message goes through *phases* that are part of the *piping model* which is used to implement *Message Exchange Patterns* (MEP). Inside these *phases* messages can be modified, filtered or processed. The advantage of doing this inside a *phase* is that it applies to all messages. This allows for service independent processing implementations. The *message receiver* will then be responsible for handing over the actual message to the service implementation accordingly. They also take care of *synchronous* and *asynchronous* message communication.

Context Model: Axis2 provides a hierarchical context model that distinguishes between the following levels:

- *Configuration* of Axis2
- *Service Group* which is a collection of *services*
- *Service* which contains several *operations*
- *Operation* that consists of *messages*
- *Message* that is sent or received

These contexts are important in the implementation of *web service* specifications such as *WS-Security* and *WS-Policy*. It means that these specifications can be applied on a level basis which provides great flexibility.

Pluggable Modules: In order to provide even more flexibility and to make the implementation of *web service* specifications easier to use, Axis2 provides *modules*. These allow an implementation of message processing that is common and useful for many *web services* to be shared. *Modules* can also be *engaged* or *disengaged* on the following levels:

- *System* which means that every service makes use of the module such as *WS-Addressing*
- *Service* which useful for *WS-Eventing*
- *Operation* that for example allows fine grained security using *WS-Security*

More information about the modules that are used in the TSSN see IV-A4.

Data Binding: Since a majority of data processing, element definitions and interface specifications are in XML, Axis2 provides a variety of *data binding frameworks* such as XMLBeans [28], Java Architecture for XML Binding (JAXB) [29] and JiBX [30]. In addition, the *Axis2 Data Binding* (ADB) can be used, which due to its tight integration with Axis2 is highly performant. For instance, every object contains a *factory* that is able to transform XML into the specific object and vice versa.

Further development was done by the author on this *data binding* to support a full range of OGC specifications such as the *Sensor Observation Service*, *Sensor Alert Service* and most notably the *Geography Markup Language*.

As part of this work several changes to the initial version of Axis2 were made in order to either fix bugs or support more functionality. In particular the build structure was adapted to work better with the TSSN development. It makes extensive use of *Apache Ant* for the automatic generation of elements from their respective XML schema definitions, the compilation of Java classes and the deployment of *web services* and clients.

c) SOAP: *Service Oriented Architectures* make use of SOAP [31] as a flexible message format. The TSSN does the same since *web service* specifications can easily be integrated and applied to SOAP messages.

d) WSDL: All services in the *Transportation Security SensorNet* are defined using the *Web Services Description Language* (WSDL) version 2.0. An in-depth introduction is provided in [32]. This section explains how the combination of WSDL files and XML schemas make up the foundation of a web service.

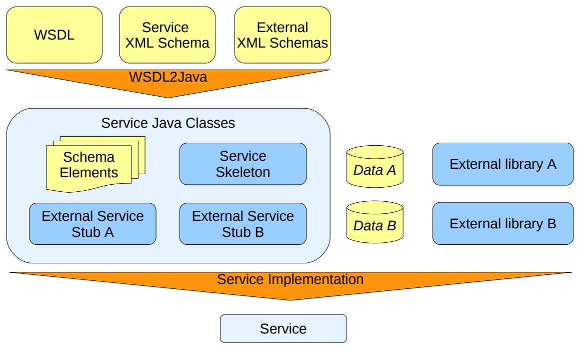

Fig. 3. Service composition

Utilizing the automatic code generator of Axis2 called *WSDL2Java*, all elements defined in the XML schemas are available as Java classes. Furthermore a *skeleton* is created that contains the operations of the web service as methods. Interaction with other services is achieved using their respective *stubs* which provide methods for each of its defined operations. They allow clients to perform requests directly using Java. This is because Axis2 provides the entire SOAP stack from the message format to the parsing into elements all the way up to the invocation of a method that represents a service operation. The composition of the generated parts, data and external libraries then forms the actual service implementation (see Figure 3).

2) Services: The services that are implemented in the TSSN make use of a variety of components. For long term information storage, a MySQL database is used. A *object-relational mapping* tool called *Hibernate* [33] enables objects to be stored and retrieved transparently without the need of complicated database interactions. *Esper* [34] provides complex event and alarm processing and is used at the VNOC. The *Alarm Processor* at the MRN currently uses a less complex approach. The *Sensor Node* is responsible for the actual communication with the sensors. It must use a device specific protocol [35] and a serial connection library for Java called *RXTX*.

Each component and its particular use is explained in the later sections when each individual service is described. At a high level, one of the main aspects when dealing with web services is the definition of whether they are *stateless* or *stateful*:

a) Stateless: By default web services are meant to be *stateless*. This is because most message exchanges are completely independent of each other. Web services usually offer calculations, information or capabilities that only require the service to perform a specific action and give a response. This is part of the *autonomy* approach of web services.

Even in the case where a web services provides data, the service is still considered *stateless* since the retrieval of the data at any given time is not dependent on the internal state of the service but only on the underlying data. If the data changes there is no state change in the web service and it still provides the same functionality.

b) Stateful: The need for *stateful* web services has been identified for the TSSN because there are certain limitations in just using *stateless* web services. Given a *online* data processor that analyzes sensor data; using a *stateless* web service, it is impossible to react to trends and complex events because the service is limited to single data objects that it receives.

Let us say that a web service is monitoring whether *seals* that lock cargo containers are broken and is supposed send out warning messages whenever they are. The service has limited capacity in terms of storing historic data but should still be able to intelligently determine if a sensor reading that shows that a seal is broken is just a misreading or a real threat. This is only possible if the service keeps track of previous states. In contrast, a *stateless* service would only be able to react to the current reading and is forced to make decisions based on this single piece of data. Another example is the *Alarm Processor* service (see IV-C2) at the MRN that is used in the TSSN implementation. It classifies sensor data from containers either as *information* or *security* depending on whether one is currently allowed to open the container or not.

3) Clients: Clients are able to make use of the operations provided by the *web services*. They usually utilize the same modules as the service. This means that in theory all *web services* could have clients. Since a lot of the services in the TSSN interact independently from users, the number of clients that are available to users is actually smaller.

One of the aspects of clients in the TSSN is the management of the sensors. The *Sensor Management* service (see IV-D1) provides this among other things like retrieving the location of a particular *Sensor Node*. Another aspect is the management of alarm notifications. For this purpose the *Alarm Reporting* service (see Figure 10) defines various management operations for clients.

To facilitate the use of those clients, a *Command Center Graphical User Interface* was implemented that works just like a desktop application. This is in addition to the command line interface that every client provides using the *Apache Commons Command Line Interface* (CLI) library.

4) Modules: Axis2 provides the possibility to "plug in" *modules* that add functionality or change the way a service behaves. This allows a specific capability to be shared among different services without having to implement it in each of them. In general, the web service specifications that are used in Axis2 are implemented as modules. For more information see IV-A1b.

a) Ping: In order to check the status of a particular service Axis2 provides a module that adds an operation called *pingService* to a service. This can be used to check the status of either a specific operation or all operations that the service defines. The client part that actually uses this operation was not part of Axis2 and had to be implemented by the author.

b) Logging: Especially for debugging purposes and performance evaluations, it is of great benefit to be able to see the raw SOAP messages that are sent and received. A *logging* module was implemented to provide this functionality. In particular the following information is captured of each SOAP message:

- *Time* when the message was sent or received
- *Service* which is used
- *Operation* that is being executed
- *Direction* of the message, which can be either incoming or outgoing. Note that there are special directions that deal with incoming and outgoing faults.
- *From* address of the message
- *Reply to* address that may differ from the *From* address
- *To* address of the message
- *Schema element* that is being "transported" as part of the operation containing the request parameters or the response elements
- *Size* of the message in bytes
- *Message* which represents the entire SOAP message in a readable form

In terms of analyzing the *Transportation Security SensorNet* and its performance the *logging* module was engaged in all services. Quantitive results obtained using the logging capability can be found in [2], [3], [4]; a tool to visualize and animate the timing of the messages is described in [4].

c) Addressing: An implementation of the *WS-Addressing* specification as described in [36], [37] comes as part of the *addressing* module in the Axis2 core. It fully supports all components of the standard and its *ReplyTo* and *RelatesTo* fields are used among other things to allow for *asynchronous* communication (see IV-A6) in the TSSN.

d) Savan: The *Savan* module enables web services and clients in Axis2 to make use of various forms of subscription mechanisms as defined by the *WS-Eventing* specification [20].

e) Rampart: In order to provide security according to the *WS-Security* specification [38] for the TSSN the *Rampart* module was developed by Axis2. It makes extensive use of the *WS-SecurityPolicy* standard described by Lawrence et al. [39].

5) Subscriptions: Subscriptions are a fundamental part of the overall architecture of the TSSN. They are used by the *Alarm Processor* at the VNOC as well as in the MRN. These web services, that act as information publishers, utilize the *Savan* module to provide the operations defined in *WS-Eventing*.

6) Synchronous and asynchronous communication: By default Axis2 uses request-response in a *synchronous* manner. This means that the client has to wait and is therefore *blocking* until it receives the response from the service. In certain scenarios, for instance when the service needs a large amount of processing time, the client can experience timeouts. Furthermore, in the TSSN where the MRN is only intermittently connected to the VNOC, *synchronous* communication shows its limitations. A better option is to make the communication between services *asynchronous*. This resolves timeout issues and deals with connections that are only temporary. The following aspects need to be taken into consideration when using *asynchronous* communication:

a) Client: The client needs to make changes in regard to the how the request is sent out. Axis2 provides a low-level *non-blocking client API* and additional methods in the service stubs that allow callbacks to be registered. These *AxisCallbacks* need to implement two methods, one that is being invoked whenever the response arrives and the other to define what happens in case of an error.

b) Transport Level: Depending on the transport protocol that is being used, Axis2 supports the following approaches.

- *One-way* uses one channel for the request and another one for the response such as the *Simple Mail Transfer Protocol* (SMTP)
- *Two-way* allows the same channel to be used for the request and the response, for example HTTP

For asynchronous communication to work the two-way approach was modified through the Axis2 *client API* which provides the option of using a *separate listener*. This tells the service that it is supposed to use a new channel for the response. In order to correlate request and response messages Axis2 makes use of the *WS-Addressing* specification, in particular the *RelatesTo* field.

c) Service: The final piece of asynchronous communication is to make the service processing asynchronous as well. This is done by specifying *asynchronous message receivers* in the services configuration in addition to the *synchronous* ones. Axis2 then uses the *ReplyTo* field of the *WS-Addressing* header in the client as a sign to send an immediate *acknowledge* of the request back to it. Furthermore it processes the request in a new thread and sends the response out when it is done, allowing the communication to be performed in asynchronous manner completely.

There exist various forms of transport protocols that are suitable for *asynchronous* communication. Axis2 by default supports HTTP, SMTP, JMS and TCP as transports but other transports can easily be defined and plugged in. The *Java Message Service* (JMS), for instance, makes use of *queues* which allow clients and services to store on them and retrieve messages in a flexible manner. This is essential for satellite communication which and discussed in detail in [3].

B. TSSN Common Namespace

Elements are often shared among a variety of services. Since defining the same element over and over again is neither a scalable nor maintainable approach, it makes sense to specify a common namespace for them and let the web services that want to use them, include them. In the TSSN these shared elements are part of the *TSSN Common* namespace.

C. Mobile Rail Network

The MRN is a collection of services that is located on a train or in a rail yard. Its services provide the abilities to manage sensors, monitor them and propagate sensor alerts to the VNOC. This section describes them in detail.

1) Sensor Node: The *Sensor Node* contains the actual sensor monitoring and management application and its components are shown in Figure 5. It provides several abstraction layers that allow various forms of sensors to be used. The current implementation makes use of cargo cable seals from Hi-G-Tek (HGT) [35]; these are considered one type of sensor

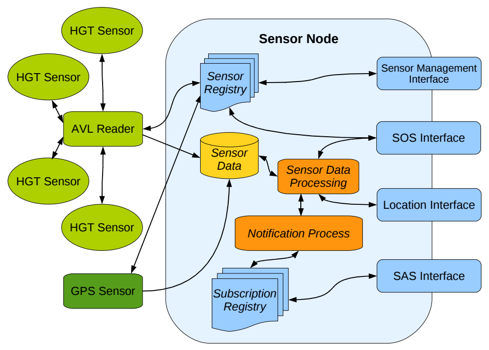

Fig. 5. Mobile Rail Network Sensor Node

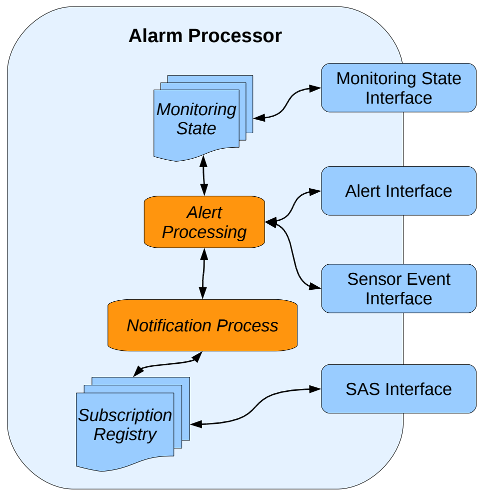

Fig. 6. Mobile Rail Network Alarm Processor

in the TSSN. Interaction with these sensors is performed using a (HGT) Automatic Vehicle Location (AVL) reader. The *Sensor Node* implements the functionality that allows higher level management of the sensors, e.g., here a collection of intelligent cargo seals connected to a series of containers, and the data that they provide through the use of a *sensor registry*, the *sensor data* storage and *sensor data processing*. Attaching a GPS sensor to the *Sensor Node* allows sensor events to be tagged with the specific location that they appeared at. The core functionality of the *Sensor Observation Service* that allows the service to offer its capabilities and observations is implemented. Furthermore, a *subscription registry* is available for alert notifications.

2) Alarm Processor: The *Alarm Processor* on the MRN performs an initial filtering of sensor events generated by the *Sensor Node*. It subscribes to of all events of the *Sensor Node*, providing interfaces for generic sensor events as well as sensor alerts. Alerts reported to the *Alarm Processor* include potential alarms that the *Sensor Node* reports, GPS acquisitions and losses, and status messages of the monitoring application such as when it is started and stopped. In case the data is not as complex as an alert, the *event* element provides a simple structure with a timestamp and a data field.

The *Alarm Processor* handles alerts and events that it

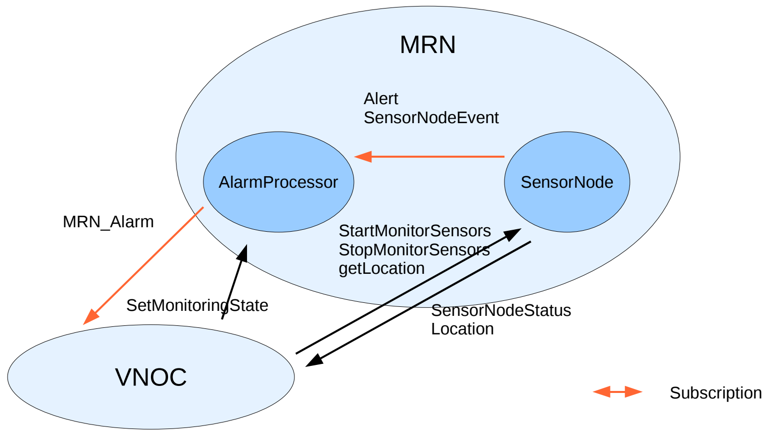

Fig. 4. Mobile Rail Network message overview

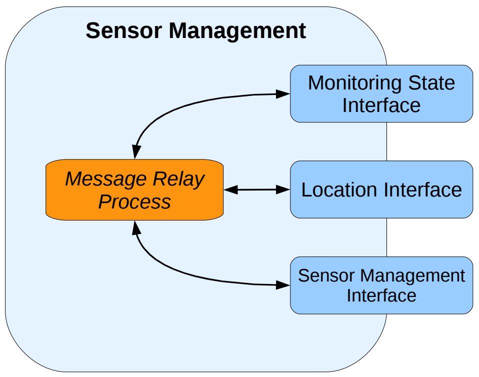

Fig. 8. Virtual Network Operation Center Sensor Management

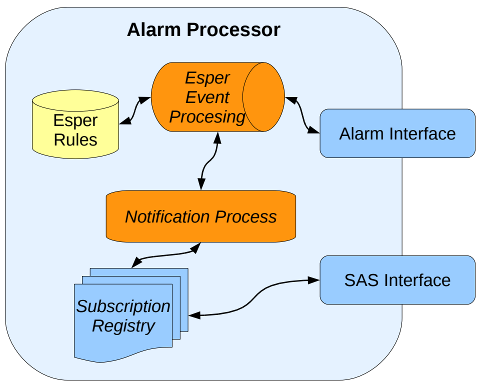

Fig. 9. Virtual Network Operation Center Alarm Processor

receives from the *Sensor Node* and classifies them into either *information* or *security* alarms depending on its current monitoring state. It is also responsible for deciding whether or not to forward the alarm to the VNOC for further processing and possible transmission to the decision maker.

D. Virtual Network Operation Center

The VNOC as shown in Figure 7 represents the management facility of the TSSN and consists of services that receive and process alerts received from MRN. It works with the TDE to associate shipment and trade information with a particular alert. Furthermore, the *Alarm Reporting* service provides clients with the ability to be notified upon specific events. The processes that are involved in performing these tasks are the topic of this section.

1) Sensor Management: The *Sensor Management* service (Figure 8) is responsible for controlling sensors and alarm reporting. It provides methods for starting and stopping sensor monitoring. Additionally the monitoring state which defines how alerts are interpreted and processed can be specified. The *Sensor Management* service essentially relays these "control" messages to the according MRN. Another functionality that is provided is the ability to query for a specific MRN's location. The implementation details of the interfaces that it provides to clients are described in the following.

The *Sensor Management* service allows the control of *Sensor Nodes* and their monitoring state. Additionally, it is able to retrieve the location of *Sensor Nodes*.

2) Alarm Processor: In contrast to the "basic" processing that is performed by the *Alarm Processor* at the MRN, the *Alarm Processor* as shown in Figure 9 at the VNOC has more resources such as the associated shipment and trade information available which is provided by the TDE and can therefore process alarms in a more complex way. This advanced filtering and processing is done using a complex event processing system called *Esper* developed by Bernhardt et al. [40].

Esper works on the basis of *sliding windows* in which events that are close together on the time axis are analyzed and correlated. It also supports using historical data from a variety of sources. An efficient query and filtering language called *Event Processing Language* allows for the most complex scenarios to be implemented. In the TSSN it is used for instance to filter out alarms for which shipment information could not be retrieved from the TDE and mark them as *security* notifications.

The *MRN_Alarm* operation is used as a notification interface

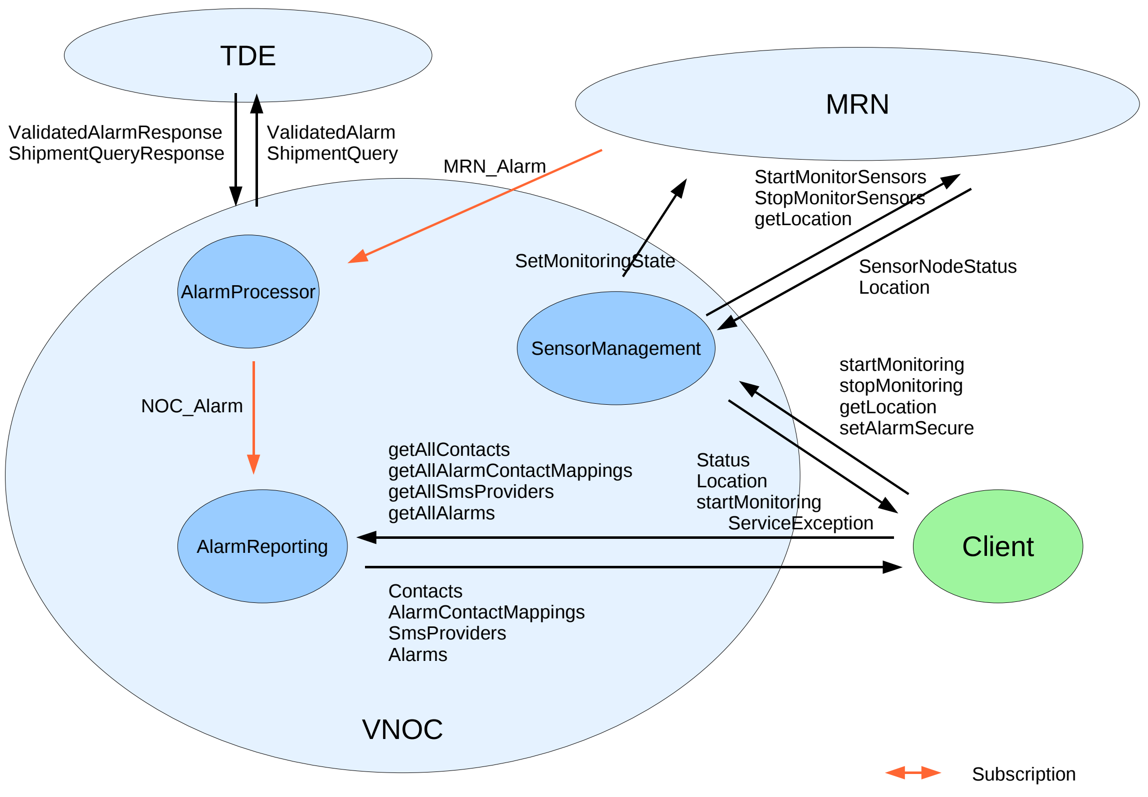

Fig. 7. Virtual Network Operation Center message overview

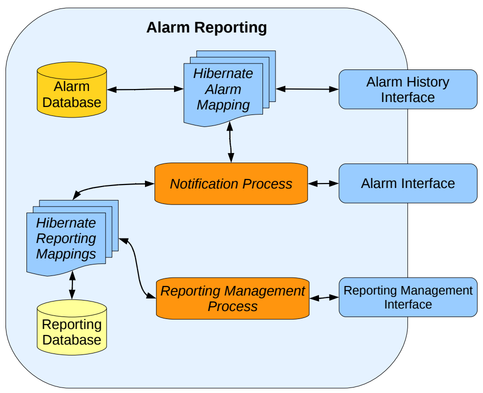

Fig. 10. Virtual Network Operation Center Alarm Reporting

for alarms from the *Alarm Processor* on the MRN. The *Alarm Processor* service subscribes to alarms from its counterpart on the MRN. Upon receiving an alarm, shipment data is retrieved from the TDE and attached to the original alarm. *Esper* then processes the alarm and passes it on to the *Alarm Reporting* service.

The *Alarm Processor* at the VNOC primarily provides functionality for the MRN to deliver alert notifications. It uses *Esper* to perform complex event processing, taking into consideration alert data and information from the TDE, and to forward alarms to the *Alarm Reporting* service.

3) Alarm Reporting: The *Alarm Reporting* service (Figure 10) deals with the following two aspects. First, it stores alarms long term to allow for in-depth reporting and analysis. Second, clients that want to be notified of particular alarms can register with the *Alarm Reporting* service. Whenever alarms occur notifications are sent out to the registered clients via email and/or SMS accordingly.

For long term data storage and to maintain a registry of the client notifications the *Alarm Reporting* service makes use of the *MySQL* database. In order to remain flexible and provide an abstraction layer to the core database functionality a tool called *Hibernate* [33] was utilized. An excellent introduction to the *object-relational* mapping is provided by Bauer et al. [41]. The main advantage is that objects referenced in code can easily be *persisted* into a relational database and vice versa. The only thing that needs to be defined is the *mapping*. Once that has been defined *Hibernate* takes care of the rest.

Since the objects that are being stored in the database are defined using XML schemas and then automatically compiled into Java objects during the build process, it makes sense to specify the mappings in XML as well. This is done in the TSSN. Another approach that is supported by *Hibernate* is using *annotations* within the Java objects themselves. This is not possible because of the aforementioned build process as the objects would have to be reannotated at every build.

The registry that is used for notifications contains *alarm contact mappings* that specify what kind of alarms a specific contact wants to be notified of. In case the contact wants to

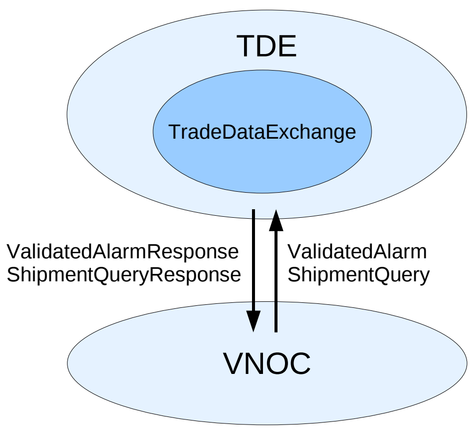

Fig. 11. Trade Data Exchange message overview

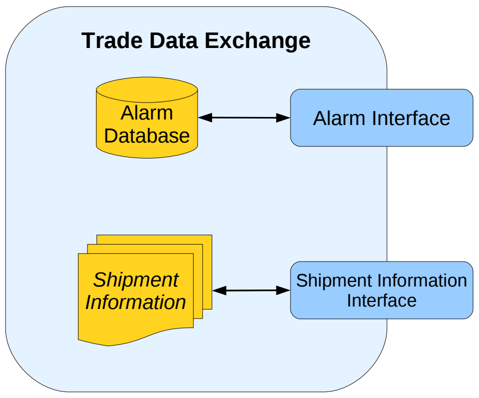

Fig. 12. Trade Data Exchange Service

receive SMS notifications, a SMS provider has to be specified as well.

The *Alarm Reporting* service receives alarm notifications from the *Alarm Processor* at the VNOC. It provides a notification interface primarily for the subscription of alarms from the *Alarm Processor*. The *Alarm Reporting* service subscribes to alarms and provides this operation for its notifications. An alarm here is a combination of the *tssn:MRN_AlarmBean* and shipment and trade information received from the TDE.

E. Trade Data Exchange

The *Trade Data Exchange* [42], as shown in Figure 11, in a sense represents a shipment and other trade data information provider. It aims to be a collection of heterogeneous systems that stores and manages the business aspects of a transport of goods. This is due to the fact that there is a variety of different systems implemented by the parties that participate in the transport chain (see II-A and II-C). Some provide route information while others manage contracts and shipment data. For the current implementation of the TSSN this "collection" of information and management services is combined into a single service, the TDE service.

The TDE service (Figure 12) interacts with the *Alarm Processor* at the VNOC. Upon request it provides shipment and trade information for a specified alarm. It also provides functionality that can be used for long term alarm storage, although in its current implementation fairly limited. Since the service was designed externally, the elements used are not compatible to the TSSN common elements or any of the other services.

F. Open Geospatial Consortium Specifications

As described before, the amount of work that is required to fully implement OGC specifications such as the *Sensor Observation Service* and the *Sensor Alert Service* is immense. The focus of the first stage of the implementation of the TSSN is on the sensor management and alarm notification capabilities. However, at the MRN the *Sensor Node* provides an implementation for the *Sensor Observation Service* as defined by the OGC. Furthermore, services in the TSSN that utilize subscriptions, in particular the *Alarm Processor*, are able to receive *subscribe* requests and publish *alerts* in a manner that is similar to the *Sensor Alert Service*. The difference to the proposed SAS specification is that the services that subscribe are already aware of the *capabilities*, *sensor* types and *alert* types. Therefore the operations that allow the retrieval of this information need to be implemented in order to be fully compliant.

V. Results

Several experiments were performed at various development stages of the TSSN. First, lab tests were conducted in order to ensure the functionality of the individual web services and their interactions.

Then, as described in [2], truck trials were completed to test basic interaction of the implemented web services and feasibility of hardware components and sensors in a mobile environment. The message exchanges between web services were correct and the system was able to recover from dropped communication links and lost GPS fixes. In addition, it was found that the read range of the sensors used is about 400 meters.

A short haul rail trial was conducted after the successful completion of truck tests. Results of the short haul rail trial are found in [2]. One of the goals was to determine the performance of the TSSN when detecting events on intermodal containers in a rail environment. Furthermore SMS message and email notification of events was investigated and data collected that could be used in the modeling of system trade-offs and communication models. The system performed well: the time it took from detecting an event to generating an alert was about 2 seconds and the average delivery time of the alert was about 12 seconds. This is well within the bounds of the requirements of the transportation industry for efficient tracking and monitoring of cargo. Note that for the short haul trial a GSM communication link was used that proved to be stable and reliable. This allowed the web services to interact synchronously with each other.

Enhancements made to the TSSN to work well in low bandwidth mobile bandwidth limited and intermittently connected monitoring environments are described in [3]. In particular, the communication link between the MRN and the VNOC was changed to a dial-up satellite connection and the web services were adapted to utilize a distributed queuing approach and hence communicate asynchronously. The viability of these adjustments was tested in a long haul rail trial in Mexico. Again the TSSN worked well and was able to transmit messages in about 12 seconds whenever connectivity was established. In case the satellite link was down and needed to be established it took about 10 minutes on average to deliver messages from the MRN to the VNOC. However the average case of about 7 minutes per message transmission through the system is found to be in range of mobile monitoring environments.

VI. Conclusion

The implementation of the *Transportation Security SensorNet* using a *Service Oriented Architecture* works. Testing has been completed in a lab environment as well as in the real world and TSSN was evaluated in [2], [3], [4]. The complete system provides a *web services* based sensor management and alarm notification infrastructure that is built using open standards and specifications. Particular functionality within the system has been implemented in *web services* that provide interfaces according to their respective *web service* specifications.

Using standards from the *Open Geospatial Consortium* allows the integration of the system into *Geographic Information Systems*. Although not all the interfaces are fully implemented as of summer 2009, the basic *Sensor Observation Service* and *Sensor Alert Service* are. Other OGC specifications can be integrated a lot easier now because enhancements to the Axis2 schema compiler have been made by the author (see IV-A1b).

WS-Eventing plays an important role in the *Transportation Security SensorNet* as it is essential for the alarm notification chain. The specification that is used by all the clients and services is *WS-Addressing*. Note that HTTP, which represents the underlying *transport layer* of most the *web services*, already provides an addressing scheme. This however, is not as useful as it seems because web services may change their *transport layer* and messages sometimes require complex routing. The reasoning behind this and other things have been explained in detail.

Overall the TSSN provides a *Service Oriented Architecture for Monitoring Cargo in Motion Along Trusted Corridors*. This *web services* based approach allows for platform and programming language independence and offers compatibility and interoperability. The integration of SOA, OGC specifications and sensor networks is complex and difficult. As described in III-H, most systems and research focuses either on the combination of SOA and OGC specifications or on OGC standards and sensor networks. However, the TSSN shows that all three areas can be combined and that this combination provides capabilities to the transportation and other industries that have not existed before. In particular, web services in a mobile sensor network environment have always been seen as slow and producing a lot of overhead. The TSSN, as shown by the results in [2], [3] demonstrates that with proper architecture and design the performance requirements of the targeted scenario can be satisfied.

Furthermore, the *Transportation Security SensorNet* and its *Service Oriented Architecture* allow sensor networks to be utilized in a standardized and open way through web services. Sensor networks and their particular communication models led to the implementation of asynchronous message transports in SOA and are supported by the TSSN.

VII. Future work

After evaluating the current implementation, several points of improvement were identified.

a) Security: The current system only provides entry points for the *WS-Security* in terms of the *Rampart* module. There are several issues in the current implementation of the module, especially with regard to attaching policies to *web services* and clients. This is discussed in [5]. Further development is underway to implement *WS-Security*. In between the *Virtual Network Operation Center* and the *Mobile Rail Network* communication is secured by establishing a *Virtual Private Network* (VPN). However, this is not practical using a satellite link because of performance reasons. Sensors management is done at the *Sensor Node* but as of now there is no support for the secure handover to other *Sensor Nodes*. The remote management systems need to be improved in this area.

b) Service Discovery: Due to several problems in the specific implementation of the UDDI that was used, for the trials most of the services were made aware of the other services through the means of configuration instead of service discovery. Since using a UDDI provides far better scalability, it is an essential piece of future versions of the TSSN.

c) Multiple service clouds: During the trials all services were unique which in an operational system this is not the case. There are issues that need to be explored in dealing with multiple versions not only of single *web services* but multiple VNOC's and MRN's. This is especially important when it comes to managing policies and subscriptions properly.

Acknowledgment

This work was supported in part by Oak Ridge National Laboratory (ORNL) Award Number 4000043403. This material is also partially based upon work supported while V. S. Frost was serving at the National Science Foundation.

The authors wish to acknowledge Kansas City Southern Railway for their participation in the rail trials. We would also like to acknowledge the support of EDS, an HP company, and Kansas City SmartPort, Inc. our partners on this project.

References

[1] M. Wolfe, "In this case, bad news is good news," *Journal of Commerce*, July 2004, www.ismasecurity.com/ewcommon/tools/download.aspx?docId=175.

[2] D. Fokum, V. Frost, D. DePardo, M. Kuehnhausen, A. Oguna, L. Searl, E. Komp, M. Zeets, D. Deavours, J. Evans, and G. Minden, "Experiences from a transportation security sensor network field trial," in *GLOBECOM Workshops, 2009 IEEE*, 30 2009-Dec. 4 2009, pp. 1–6.

[3] M. Kuehnhausen and V. S. Frost, "Application of the Java Message Service in Mobile Monitoring Environments," Information Telecommunication and Technology Center, University of Kansas, Lawrence, KS, Tech. Rep. ITTC-FY2010-TR-41420-18, February 2010.

[4] ——, “Framework for Analyzing SOAP Messages in Web Service Environments,” Information Telecommunication and Technology Center, University of Kansas, Lawrence, KS, Tech. Rep. ITTC-FY2010-TR-41420-20, March 2010.

[5] E. Komp, V. S. Frost, and M. Kuehnhausen, “Implementing Web Services: Conflicts Between Security Features and Publish/Subscribe Communication Protocols,” Information Telecommunication and Technology Center, University of Kansas, Lawrence, KS, Tech. Rep. ITTC-FY2010-TR-41420-19, February 2010.

[6] D. Group, “The value of standards,” Delphi Group, Ten Post Office Square, Boston, MA 02109, Survey, Jun. 2003, www.ec-gis.org/sdi//ws/costbenefit2006/reference/20030728-standards.pdf.

[7] M. Reichardt, “The Havoc of Non-Interoperability,” OGC, OGC White Paper, Dec. 2004, http://portal.opengeospatial.org/files/?artifact_id=5097.

[8] L. McKee, “The Importance of Going “Open”,” OGC, OGC White Paper, Jul. 2005, http://portal.opengeospatial.org/files/?artifact_id=6211.

[9] M. Irmen, “10 ways to reduce the cost and risk of global trade management,” *Journal of Commerce*, March 2009, http://www.joc.com/node/410216.

[10] L. F. Cabrera, C. Kurt, and D. Box, “An Introduction to the Web Services Architecture and Its Specifications,” Microsoft, Microsoft Technical Article, Oct. 2004, http://msdn.microsoft.com/en-us/library/ms996441.aspx.

[11] D. Nickul, L. Reitman, J. Ward, and J. Wilber, “Service Oriented Architecture (SOA) and Specialized Messaging Patterns,” Adobe, Adobe Article, Dec. 2007, www.adobe.com/enterprise/pdfs/Services_Oriented_Architecture_from_Adobe.pdf.

[12] M. Botts, G. Percivall, C. Reed, and J. Davidson, “OGC Sensor Web Enablement: Overview And High Level Architecture,” OGC, OGC White Paper, Dec. 2007, http://portal.opengeospatial.org/files/?artifact_id=25562.

[13] X. Chu, T. Kobialka, and R. Buyya, “Open sensor web architecture: Core services,” in *In Proceedings of the 4th International Conference on Intelligent Sensing and Information Processing*. Press, 2006, pp. 1–4244, http://www.gridbus.org/papers/ICISIP2006-SensorWeb.pdf.

[14] X. Chu, “Open sensor web architecture: Core services,” Master’s thesis, University of Melbourne, Australia, 2005, http://www.gridbus.org/reports/OSWA-core%20services.pdf.

[15] D. Fitzpatrick, D. Dreyfus, B. A. Hamilton, M. Onder, and J. Sedor, “The electronic freight management initiative,” U.S. Department of Transportation, Federal Highway Administration, Tech. Rep. FHWA-HOP-06-085, April 2006.

[16] K. Troup, , D. Newton, M. Wolfe, and R. Schaefer, “Columbus electronic freight management evaluation - achieving business benefits with efm technologies,” Science Applications International Corporation (SAIC), Tech. Rep., March 2009.

[17] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, “The physiology of the grid: An open grid services architecture for distributed systems integration,” in *Open Grid Service Infrastructure WG, Global Grid Forum*, Jun. 2002, http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[18] T. Bellwood, L. Clement, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen, “UDDI Version 3.0,” OASIS, OASIS Specification, Jul. 2002, http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[19] T. Bellwood, “Rocket ahead with UDDI V3,” IBM, IBM Article, Nov. 2002, http://www.ibm.com/developerworks/webservices/library/ws-uddiv3/.

[20] D. Box, L. F. Cabrera, C. Critchley, F. Curbera, D. Ferguson, S. Graham, D. Hull, G. Kakivaya, A. Lewis, B. Lovering, P. Niblett, D. Orchard, S. Samdarshi, J. Schlimmer, I. Sedukhin, J. Shewchuk, S. Weerawarana, and D. Wortendyke, “Web services eventing (ws-eventing),” W3C, W3C Member Submission, Mar. 2006, http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/.

[21] A. Friis-Christensen, N. Ostländer, M. Lutz, and L. Bernard, “Designing service architectures for distributed geoprocessing: Challenges and future directions.” *Transactions in GIS*, vol. 11, no. 6, pp. p799 – 818, 20071201. [Online]. Available: http://search.ebscohost.com.www2.lib.ku.edu:2048/login.aspx?direct=true&db=aph&AN=28048261&site=ehost-live

[22] C. Kiehle, K. Greve, and C. Heier, “Requirements for next generation spatial data infrastructures-standardized web based geoprocessing and web service orchestration.” *Transactions in GIS*, vol. 11, no. 6, pp. p819 – 834, 20071201. [Online]. Available: http://search.ebscohost.com.www2.lib.ku.edu:2048/login.aspx?direct=true&db=aph&AN=28048260&site=ehost-live

[23] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, “Reference Model for Service Oriented Architecture 1.0,” OASIS, OASIS Standard, Oct. 2006, http://docs.oasis-open.org/soa-rm/v1.0/.

[24] H. Haas, D. Booth, E. Newcomer, M. Champion, D. Orchard, C. Ferris, and F. McCabe, “Web services architecture,” W3C, W3C Note, Feb. 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[25] M. Kalin, *Java Web Services: Up and Running*. O’Reilly, February 2009.

[26] L. S. Searl, “Service Oriented Architecture for Sensor Networks Based on the Ambient Computing Environment,” ITTC, ITTC Technical Report, Feb. 2008, www.ittc.ku.edu/sensornet/trusted_cooridors/papers/41420-07.pdf.

[27] E. Chinthaka, “Web services and Axis2 architecture,” IBM, IBM Article, Nov. 2006, https://www.ibm.com/developerworks/webservices/library/ws-apacheaxis2/.

[28] A. S. Foundation, “XMLBeans,” Jul. 2008. [Online]. Available: http://xmlbeans.apache.org/

[29] J. Fialli and S. Vajjhala, “Java architecture for xml binding (jaxb) 2.0,” Java Specification Request (JSR) 222, October 2005.

[30] D. Sosnoski, “JiXB,” Mar. 2009. [Online]. Available: http://jibx.sourceforge.net/

[31] Y. Lafon and N. Mitra, “SOAP version 1.2 part 0: Primer (second edition),” W3C, W3C Recommendation, Apr. 2007, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[32] D. Booth and C. K. Liu, “Web services description language (WSDL) version 2.0 part 0: Primer,” W3C, W3C Recommendation, Jun. 2007, ”http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626.

[33] R. Hat, “Hibernate Reference Documentation 3.3.1,” Tech. Rep., Sep. 2008, http://www.hibernate.org/hib_docs/v3/reference/en-US/pdf/hibernate_reference.pdf.

[34] EsperTech, “Esper - Event Stream and Complex Event Processing for Java.” [Online]. Available: http://www.espertech.com/

[35] Hi-G-Tek. [Online]. Available: http://www.higtek.com/

[36] M. Gudgin, M. Hadley, and T. Rogers, “Web services addressing 1.0 - core,” W3C, W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509.

[37] M. Gudgin, M. Gudgin, M. Hadley, T. Rogers, T. Rogers, and M. Hadley, “Web services addressing 1.0 - SOAP binding,” W3C, W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509.

[38] K. Lawrence, C. Kaler, A. Nadalin, R. Monzillo, and P. Hallam-Baker, “Web Services Security: SOAP Message Security 1.1 (WS-Security 2004),” OASIS, OASIS Standard, Feb. 2006, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[39] K. Lawrence, C. Kaler, A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist, “WS-SecurityPolicy 1.2,” OASIS, OASIS Standard, Jul. 2007, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf.

[40] T. Bernhardt and A. Vasseur, “Event-driven application servers,” 2007. [Online]. Available: http://dist.codehaus.org/esper/JavaOne_TS-1911_May_11_2007.pdf

[41] C. Bauer and G. King, *Hibernate in Action*. Manning, 2005.

[42] K. SmartPort, “Trade Data Exchange - Nothing short of a logistics revolution,” *Journal of Commerce*, November 2008. [Online]. Available: http://www.joc-digital.com/joc/20081110/?pg=29