Journal of Network and Computer Applications 34 (2011) 1707–1716

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Application of the Java Message Service in mobile monitoring environments

Martin Kuehnhausen *, Victor S. Frost

Information and Telecommunication Technology Center, The University of Kansas, Lawrence, KS 66045, USA

ARTICLE INFO

Article history:
Received 23 December 2010
Received in revised form
6 May 2011
Accepted 12 June 2011
Available online 24 June 2011

Keywords:
Telemetry
Transport protocols
Intermittently connected wireless networks
Communication system software
Data communication
Software engineering

ABSTRACT

Distributed systems and sensor networks in particular are in need of efficient asynchronous communication, message security and integrity, and scalability. These points are especially important in mobile environments where mobile remote sensors are connected to a control center only via intermittent communication. We present a general approach for dealing with the issues that arise in such scenarios. This approach is applied to provide flexible and efficient cargo monitoring on trains.

The Java Message Service (JMS) presents a flexible transport layer for asynchronous communication that enables transparent *store-and-forward* queuing for entities that need to be connected to each other. Previously JMS was primarily used in always-connected high-bandwidth enterprise communication systems. We present the advantages of using JMS in a mobile, bandwidth-limited, and intermittently connected monitoring environment and provide a working implementation called the Transportation Security SensorNet (TSSN) that makes use of an implementation of JMS called ActiveMQ. This solution is employed here to enable monitoring of cargo in motion along trusted corridors.

Results obtained from experiments and a field trial show that using JMS provides not just a practical alternative to often custom binary communication layers, but a better and more flexible approach, by providing transparency. Applications on both communication ends only need to implement JMS connectors while the remaining functionality is provided by the JMS implementation. Another benefit arises from the exchangeability of JMS implementations. In utilizing JMS we demonstrate a new, flexible and scalable approach to cope with challenges inherent in intermittent and low-bandwidth communication in mobile monitoring environments.

1. Introduction

There exist a plethora of problems that need to be addressed whenever disparate systems are deployed in the field that need to communicate with each other and a control center. Additional challenges arise given that these systems are often heterogeneous where different system elements are not always compatible to each other. Here we used the following scenario as a motivating example to point out the nature of the problems.

Sensors are connected to cargo containers which they monitor. A train is then used to transport these containers. The sensors, in our case electronic seals, have limited capabilities and are managed locally by a more powerful system element which we call the sensor node that has extended functionality including a communication link back to a control center. Hence, one sensor node controls more than one electronic seal. From an architecture perspective a sensor node can control many different sensors. Whenever a seal detects an event it notifies the sensor node immediately. The sensor node then performs an evaluation of the event and decides whether or not to send it to the control center. In this paper, we focus on sending messages to and receiving control messages from the control center.

The link between the sensor node and control center may provide only intermittent communication. The sensor node must deal with establishing the connection as well as transmitting messages. Especially the latter can cause problems; in a synchronous communication model the sensor node would only be able to send one message at a time and block while waiting for its acknowledgement. This is not feasible in this case because of the intermittent connection, low bandwidth and high latency of the communication link. An asynchronous communication model overcomes this blocking problem. Furthermore, since messages cannot be sent out immediately due to the intermittent connectivity they need to be stored. This can be achieved by implementing a queuing mechanism inside the sensor node.

It is also possible to send control messages such as location or receive status inquiries from the control center to the sensor node. Again, since there does not necessarily exist an active connection to the sensor node messages need to be queued. Hence, the applications in the control center are responsible for implementing proper queuing and retry mechanisms.

Security and message integrity are critical aspects of the overall monitoring system. If people, who want to steal cargo from the

* Corresponding author.
E-mail address: mkuehnha@ittc.ku.edu (M. Kuehnhausen).

1084-8045/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2011.06.003

monitored containers, were able to tamper with the message contents then they could spoof the system. Security is essential and needs to be implemented in each application that sends or receives messages as part of such a monitoring system. This brings up another issue, scalability. Implementing asynchronous communication and security components for each application in a small system may work for experiments but is not feasible for large production environments. Custom implementations for different components or every type of sensor are labor intensive, while a generic standards-based approach scales and eases deployment. In terms of the cargo monitoring scenario for trains there could be many control centers, thousands of sensor nodes and even more sensors on containers. The above is a very common scenario for sensor network deployments even though the particular details of the deployments may be different.

The primary use of Java Message Service (JMS) is in always-connected high-bandwidth enterprise communication systems but its concepts and techniques are useful in other scenarios as well. In this paper, we demonstrate that by using JMS in these mobile monitoring environments it is possible to overcome the problems discussed above. In particular, we focus on the problems of asynchronous communication, message security and integrity, and scalability.

In addition, one of the main advantages of using JMS is the fact that applications do not need to be modified to implement their own *store-and-forward* or *resend* mechanisms. How this is achieved is explained in detail later. Furthermore, an example of an implementation of a mobile monitoring system is given that was field tested in stationary as well as intermittently connected mobile scenarios.

Because JMS is primarily used in always-connected enterprise networks that are not bandwidth limited, JMS has not previously been considered an option for environments composed of mobile elements limited via intermittent communications channels. The contribution of this work is to demonstrate that JMS provides a feasible solution, eliminating custom code and/or applications for security and *store-and-forward* or *resend* message transmissions. This work also shows that JMS offers additional benefits, e.g., integration with web services.

1.1. Asynchronous communication

Reliable communication between control centers and the sensor networks cannot always be ensured. Additionally communication is based on the form of underlying connectivity that is provided. The connectivity may vary. The fact that the system could use a 3G system when available and resort to the satellite communication only when needed exposes several issues.

First, message sizes should be small in order to accommodate for the slow data rates. Possible optimizations are discussed in Section 2.4.4 but compression or conversion into binary formats are suitable options here.

Second, in order to address reliable transmission of messages either a *store-and-forward* or a *resend* mechanism needs to be implemented on both communication ends. The *store-and-forward* technique in this context would mean that the sensor network needs to hold on to the data captured until connectivity is established. By contrast, in the *resend* scenario they would attempt to transmit the data continuously or with a backoff timer. In this paper, we show that JMS is able to handle asynchronous communication scenarios well.

1.2. Message security and integrity

The data produced by sensor networks will likely be sensitive and needs to be kept private. This is especially true for systems whose main purpose is to provide monitoring of cargo. Cargo information as well as status updates and events should only be visible to authorized entities. Furthermore, it is critical that messages being transmitted, for example, control messages that allow the opening of cargo containers, cannot be tampered with.

In this sense it is also important to distinguish between *point-to-point* and *end-to-end* security. Using transit networks or message relay mechanisms is not possible when messages are secured in a point-to-point manner because security may be compromised at each individual connection point. However, in *end-to-end* system security it is possible for messages to pass through individual connection points. Within the cargo transport scenario multiple parties may have access to intermediate parts of the system. In particular, some services are being used by multiple shippers and different transportation authorities. This makes *end-to-end* system security a must. Another issue to consider is that while control centers often have adequate storage and computing power, individual sensors or sensor networks may not. Dealing with limited computing power and bandwidth can be a challenge when implementing security for the targeted scenarios.

JMS integrates with web service architectures their specifications. It can be used as a tunnel for SOAP messages where web services specifications are then used to ensure message security and integrity. This is explained in more detail later. The success of JMS in the environment considered here is demonstrated for the first time in this work.

1.3. Scalability

Sensor networks in general can be set up in two basic ways. First, after an initial configuration they repeatedly report their sensor data to a control center. Second, a control center sends out messages to the sensors or sensor networks in order to control their reporting or inquire for specific sensor data. Thus efficient management and scalability can become an issue.

Even though the most common scenario is running a single setup with one central control center or base station and multiple sensors or sensor networks connecting to it, the integration of multiple systems can be problematic. For example, the cargo transport scenario allows management of different sensor networks and coordination between transportation systems. Furthermore, there are issues in dealing with multiple control centers and multiple sensor networks that need to be explored. This is especially important when it comes to managing policies and subscriptions properly. We show that JMS provides mechanisms to allow for scalability as well as policy and subscription management for these challenging environments.

In the following, we first introduce the system we developed called the Transportation Security SensorNet (TSSN) which is a specific case of a mobile monitoring environment scenario we described earlier. Next, we compare different messaging approaches to JMS and describe how our application of JMS differs from others. This paper also shows how JMS has the advantage of using web service specifications for security and message integrity.

We then present the use of JMS in mobile monitoring environments by demonstrating how the TSSN utilizes it for communicating efficiently between its control center and sensor networks. Finally, results from field trials are evaluated and it is shown that JMS presents a viable solution for efficient messaging in mobile monitoring environments.

2. Related work

2.1. Transportation Security SensorNet

The Transportation Security SensorNet by Kuehnhausen (2009) serves as a specific scenario and platform for demonstrating the need

of JMS for communicating efficiently between disparate mobile entities. As shown in Fig. 1 the TSSN uses a service-oriented architecture (SOA) approach for monitoring cargo in motion along trusted corridors. The complete system provides a web services-based sensor management and event notification infrastructure that is built using open standards and specifications. This web services-based implementation allows for platform and programming language independence and offers compatibility and interoperability with other systems.

The TSSN represents the integration of SOA, Open Geospatial Consortium (OGC) specifications and sensor networks. Previous systems and research focused either on the combination of SOA and OGC specifications or on OGC standards and sensor networks. However, the TSSN shows that all three can be combined and that this combination provides capabilities to the transportation and other industries that have not existed before. In particular, the preeminent lack of the performance in mobile sensor network environments has previously limited the application of web services because they have been perceived as too slow and having excessive overhead. The TSSN, as shown by the results in Kuehnhausen (2009), demonstrates that with proper architecture and design the performance requirements of the targeted scenario, a mobile monitoring environment, can be satisfied.

Furthermore, unlike existing proprietary implementations the TSSN allows 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.

An overview of the messages within the TSSN is shown in Fig. 2. The MRN represents a train-mounted sensor network (sensors and sensor node) that monitors seals on cargo containers. The MRN is able to receive control messages such as when to start and stop monitoring. When sensor event is detected it is

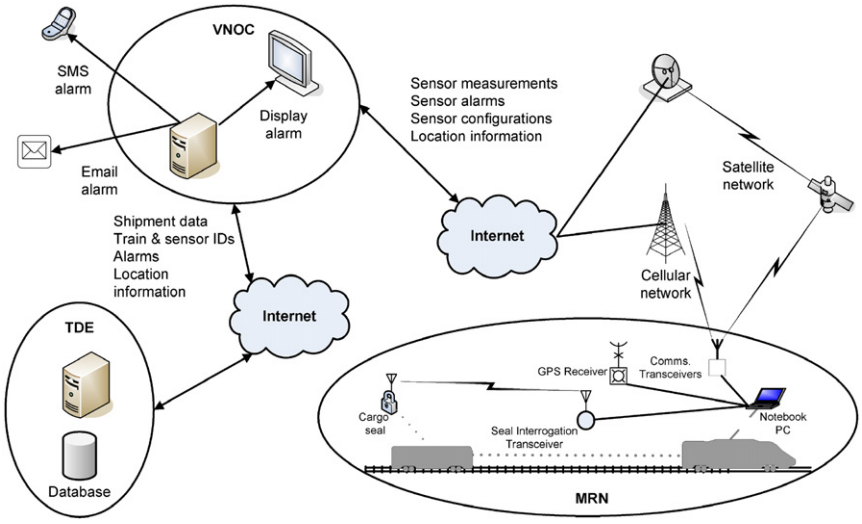

Fig. 1. TSSN physical architecture adapted from Fokum et al. (2010).

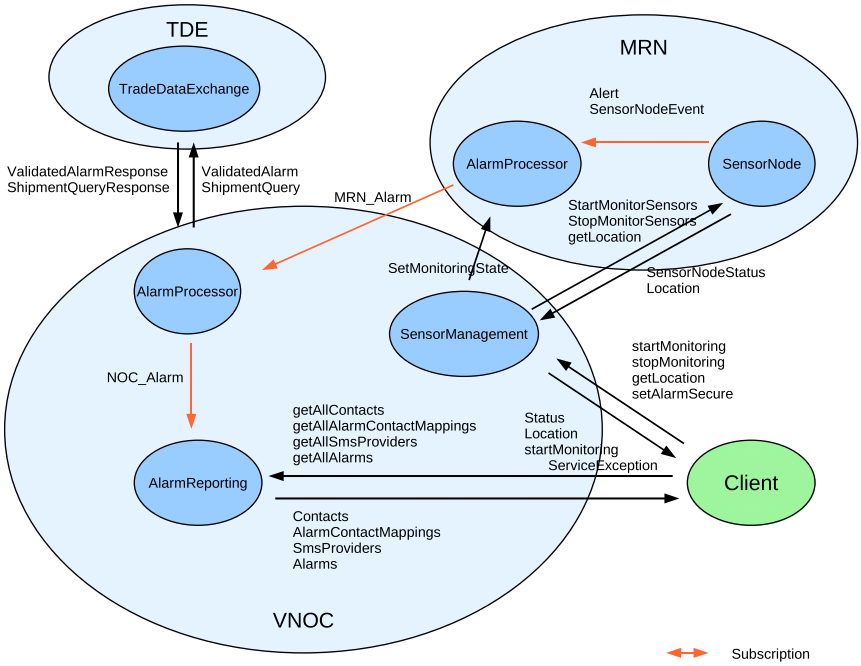

Fig. 2. TSSN message overview from Kuehnhausen (2009).

transmitted from a sensor (seal) to the sensor node where it is analyzed to determine whether or not to send out a notification to the VNOC. Sensor management and correlation of events with shipment and route information is then performed in the VNOC. According to specified mappings, people and organizations that subscribed to these notifications then receive e-mails and/or SMS messages containing detailed information about the time, location, cargo, logistics information, and nature of the event.

In terms of the communication, the critical link is between the Mobile Rail Network and the Virtual Network Operation Center because it cannot be guaranteed that there always exists a link and hence an asynchronous communication model had to be implemented. An approach that is able to deal with message queuing on both ends of the communication is the integration of JMS as the transport. The TSSN implementation fully supports asynchronous communication using queues in order to send and receive messages.

2.2. Message-oriented middleware

Message-oriented middleware is a technology that is used to decouple the exchange of messages between a sender and receiver by using intermediaries. This allows to provide flexibility and scalability. There are various approaches to implement message-oriented middleware, the most well-known are the Advanced Message Queuing Protocol (AMQP) by the AMQP Working Group (2009), JMS by Hapner et al. (2002) and the Data Distribution Service (DDS) by the Object Management Group (2007).

AMQP provides two attributes, a protocol model and a detailed protocol description, that can be used to develop messaging service implementations. Vinoski (2006) argues that this approach leads to more interoperability between vendors because the semantics of handling messages and what is actually transmitted are clearly defined.

JMS, on the other hand, defines an Application Programming Interface (API) and leaves the inner workings and the underlying transport protocol up to the implementation. The advantage here is that this protocol can easily be switched without having to modify the applications using JMS.

DDS takes a similar approach to JMS. However, DDS is data-centric while JMS is message-centric. Pardo-Castellote (2003) describes this in more detail but the basic idea is to uniquely identify data objects instances and provide mechanisms to modify their properties using keys.

2.3. Experiences with the Java Message Service

Musolesi et al. (2004) present their experiences in implementing a system called EMMA (Epidemic Messaging Middleware for Ad hoc networks) based on JMS. In particular they identified the need to adapt JMS in order to be applicable for mobile ad hoc networks. Their approach consists in synchronization of queues using a middleware layer that also manages reachability of individual nodes. For message delivery in partially connected networks they make use of an approach called epidemic routing which is described by Vahdat and Becker (2000), which works by propagating messages to neighbors, their neighbors and so on. In contrast the solution developed here and implemented in the TSSN is standards-based using the original JMS specification and, therefore, more compatible with other systems.

Vollset et al. (2003) present a middleware platform built for mobile ad hoc networks. Their solution is “serverless” in the sense that after an initial setup all the participating entities have a local copy of the JMS configuration. Furthermore, they implement a new multicast protocol for delivering messages on JMS *topics* to their *subscribers*. Their platform is an adaptation of the original JMS standard for ad hoc networks. In contrast, the solution proposed here deals with point-to-point connectivity. Here we show that JMS can be used unaltered.

In general JMS is primarily used in always-connected systems such as the one described by Allard (2007). The Mission Data Processing and Control Subsystem (MPCS) by Allard (2007) utilize JMS for different levels of event notifications. However, the communication link with the flight systems is custom. Although this represents an extreme case, it seems that using JMS for establishing mobile connectivity is undervalued. Another, more realistic example is the remote real-time oil well monitoring system by Hongsheng et al. (2005), where clients receive event notifications via JMS but the data that is collected by the remote terminal units is sent to the data processing station using a custom process.

2.4. Web services

SOA present a flexible solution to some of the problems mentioned earlier such as message security and scalability. The idea is to implement specific functionality in web services that communicate with each other using standardized interfaces. Message exchanges in general use the flexible SOAP message format by Lafon and Mitra (2007). This has a number of advantages, routing and security mechanisms are available. JMS is able to transmit SOAP messages. Hence, applying JMS enables the use of web service specifications in mobile monitoring environments.

2.4.1. WS-Addressing

The *WS-Addressing* core specification by Gudgin et al. (2006b) and its SOAP binding by Gudgin et al. (2006a) define how message propagation can be achieved using the SOAP message format. Usually the transport of messages is handled by the underlying transport protocol but there are several advantages of storing this transport information as part of the header in the actual SOAP message. For example, it allows the routing of messages across different protocols and management of individual flows and processes within web services.

JMS uses a similar concept for its addressing but its properties are adapted to the management of messages in queues. However, since SOAP messages can be transported using JMS, which is the case here, flexible routing of messages using addressing models is possible.

2.4.2. WS-Security

The *WS-Security* specification as described by Lawrence et al. (2006) deals with the many features needed to achieve *end-to-end* message security. This provides security throughout message routing and overcomes the limitations of *point-to-point transport layer security* such as HTTPS. Furthermore, the specification provides support for various security tokens, trust domains, signature formats and encryption technologies. Whenever SOAP messages are transported using the JMS, *WS-Security* can be applied. In this scenario JMS simply acts as a tunnel.

2.4.3. WS-ReliableMessaging

Without additional specifications like *WS-ReliableMessaging* by Fremantle et al. (2007), the delivery of SOAP messages is based purely on best effort and cannot necessarily be guaranteed. JMS provides several mechanisms for dealing with message reliability issues. Within *transactions*, messages are *acknowledged* and if necessary *redelivered*. When a message carries the *persistent* attribute, JMS message *brokers* store the message in order to be able to recover it in case of a failure.

2.4.4. Efficient data transmission

The SOAP 1.2 Primer by Lafon and Mitra (2007) includes references to several enhancements of the original SOAP standard.

In particular they deal with potential performance problems and the need for binary data transport in SOAP. The *XML-binary Optimized Packaging* (XOP) specification by Mendelsohn et al. (2005) defines the use of *MIME Multipart/Related* messages provided by Levinson (1998) to avoid encoding overhead that occurs when binary data is used directly within the SOAP message. XOP extracts the binary content and uses URIs to reference it in the *extended part* of the message. An abstract specification that uses this idea is the *Message Transmission Optimization Mechanism* (MTOM) by Nottingham et al. (2005).

Another extension of the SOAP standard is the *Resource Representation SOAP Header Block* (RRSHB) by Gudgin et al. (2005) that allows for caching of data elements using *Representation header blocks*. They contain resources that are referenced in the SOAP *Body* which might be hard to retrieve or simply referenced multiple times. Instead of having to reacquire these resources over and over again, a service may choose to use the cached objects which speeds up the overall processing time.

These extensions and the web service specifications described above show that SOA is feasible for the scenarios considered. In the following sections we discuss this unique combination of JMS, web services and mobile monitoring environments.

3. Proposed solution

To provide a flexible, scalable and suitable solution for a mobile monitoring environment, a transparent *store-and-forward* approach is used here. The reason is that a *resend* mechanism is often implemented directly in the application which is not flexible. The *store-and-forward* approach, however, allows for an efficient and scalable centralized storage pool that is automatically forwarding the messages.

How JMS can be applied effectively is developed here using the Transportation Security SensorNet (TSSN) by Kuehnhausen (2009) as a motivating example (Fig. 1). The TSSN provides monitoring capabilities in mobile environments and makes use of JMS. The TSSN uses an SOA approach for monitoring cargo in motion along trusted corridors. The system is built using web service specifications and utilizes a Java Message Service implementation for connectivity between its Virtual Network Operation Center (VNOC), the control center in this case, and the Mobile Rail Networks (MRN) it monitors, which contain the sensor nodes and sensors. A sensor node controls multiple sensors.

The TSSN uses the JMS through one of its open-source implementations called ActiveMQ which is described in detail by Snyder et al. (2009). Each application in the TSSN is a web service. These web services can be utilized through their JMS addresses. ActiveMQ establishes a *queue* for each web service and uses these *queues* to *store-and-forward* messages to them.

This *queue* approach has the advantage that applications do not need to be modified and implement their own *store-and-forward* or *resend* mechanisms. It is also transparent to clients of the web services since apart from using another address, the JMS address, interfacing with the web services stays the same. The TSSN data traffic between the VNOC and the MRN consists of SOAP messages that are enveloped in JMS messages.

In this paper, we explain the basic concepts of JMS and how they are matched to mobile monitoring environments and one implementation, the TSSN. Furthermore the details for the JMS implementation within the TSSN are discussed.

3.1. Java Message Service

JMS by Hapner et al. (2002) provides a standardized specification for synchronously and asynchronously transporting messages using queues. Its implementation is vendor specific but the interfaces are clearly defined in the specification so that this is an open system where changing vendors is possible. The following sections match JMS concepts to aspects of mobile monitoring environments.

3.1.1. Components

In the JMS context, clients are called *producers* when they create and send messages. The receiving end is called a *consumer*. Note that a client can be both, a producer and a consumer, at the same time. Clients connect to JMS *providers* which are entities that have the specified interfaces to send and receive messages.

Most of the connections are point-to-point and a *queue* is the commonly used *destination* of a message. The queue contains messages from *producers* to a single *consumer* that have not been received. Within the TSSN unique *queues* are used to represent the individual web services. Messages are usually delivered in order *First In, First Out* (FIFO) following the basic principle of a queue, but this is dependent on the underlying implementation of JMS.

Topics have multiple *consumers* and can have one or many *producers* publishing messages. They are used in publish–subscribe models and contain messages that have not yet been published.

A *message* can be any object or data that needs to be transported using JMS. JMS describes messages as entities that consist of a *header*, which contains identification and routing information and a body carrying the data. Additional properties such as application, provider or standards specific properties can be attached to messages. This is effectively used in providing functionality like security or reliable messaging.

Note that since JMS does not define a message format per se, implementation may vary significantly. For service-oriented architectures the agreed standard message format is SOAP. Easton et al. (2008) describe in detail how SOAP can be used within JMS. Because the TSSN is based on SOA and uses SOAP messages it is able use web service specifications as part of JMS and therefore provide features such as *WS-Addressing* and *WS-Security* as described in Section 2.4.

In order to identify objects within the JMS implementation in a standardized way the specification makes use of the Java Naming and Directory Interface (JNDI) Application Programming Interface by Sun Microsystems (1999). JNDI provides a directory service for objects. JMS clients look up objects and use them in connections as shown in Fig. 3.

3.1.2. Messaging models

JMS supports the two common messaging models; point-to-point and publish–subscribe. These are also called message *domains*. Both of them allow for true asynchronous communication in which the message *consumer* does not need to be connected to the *producer* at the time when the message is sent and vice versa.

Point-to-point. This messaging model makes use of queues and is shown in Fig. 4. Its main application is a request–response type of message exchange. Messages in this model are truly unique in the sense that once the *consumer* receives and acknowledges the message it is removed from the queue. While there can be only a single *consumer*, messages can be put on the queue by multiple *producers*. This is the model used here because message propagation throughout the mobile monitoring system is done from one web service to another.

Publish–subscribe. Whenever there is the need for multiple *consumers* to receive messages, a subscription model is useful. The *consumers* subscribe to a specific *topic* and receive messages as soon as they are published by one or multiple *producers* as shown

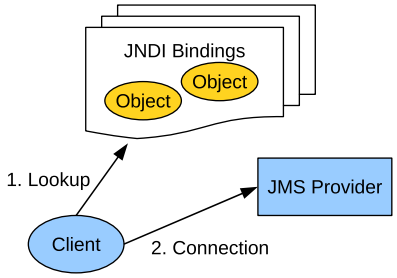

Fig. 3. JMS administration according to Hapner et al. (2002).

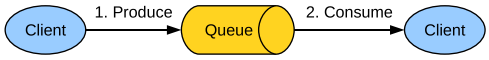

Fig. 4. Point-to-Point messaging.

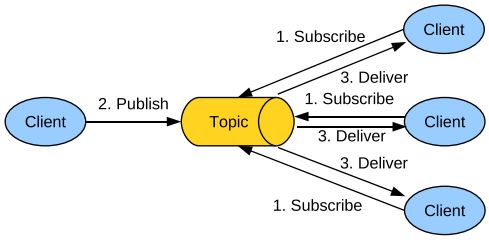

Fig. 5. Publish/Subscribe messaging.

in Fig. 5. There exists no direct connection between *publishers* and *subscribers*.

Here the JMS *publish–subscribe* messaging model is currently not used since publications are handled using the web service standard *WS-Eventing* for SOAP messages. However, JMS *publish–subscribe* presents a flexible approach to scalability that may be used in future versions of the system instead of the current *WS-Eventing* approach. Since switching between messaging models within this system is only based on configuration parameters and not on a different implementation it is possible to use JMS *publish–subscribe* effectively in mobile monitoring environments as well.

3.2. Web services integration

The implementation of the TSSN uses the Apache Axis2 web services software stack. 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 scenarios of interest here the client can experience timeouts; thus *synchronous* communication is not feasible.

A better option is to make the communication between services *asynchronous*. This resolves timeout issues and deals with connections that are only temporary. The TSSN was adjusted at the client, transport and service level in order to support *asynchronous* communication. In order to correlate request and response messages Axis2 makes use of the *WS-Addressing* specification, in particular the *ReplyTo* and *RelatesTo* fields.

There exist various forms of transport protocols that are suitable for *asynchronous* communication. Axis2 by default supports HTTP, SMTP, and JMS as *asynchronous* transports but other transports can easily be defined and plugged in.

A *transport receiver* for JMS was added to this system's web services. This represents the receiving end of the communication and allows web services and clients to consume JMS messages by creating a JMS address for them while a *transport sender* allows JMS messages to be produced.

Axis2 by default sets up a *queue* for each of the services and uses the service name as the *queue* name. Since a service is not necessarily unique, this name can be changed in the service configuration. For the MRN this naming consists of the node id which is used to represent a sensor network and the name of the service. For the VNOC the name is made up of the host on which the service is run and its service name. This makes it possible to easily identify queues and avoid misconfiguration of the JMS implementation while also offering a naming scheme that is scalable.

3.3. JMS implementation

Apache ActiveMQ is used here for JMS messaging. A detailed introduction is given by Snyder et al. (2009). ActiveMQ is mostly used in enterprise systems where high-bandwidth connectivity is a given and high throughput is important. Note that the version used here had to be modified by the authors because ActiveMQ could not work correctly without an existing and permanent Internet connection. However, being able to function without constant connectivity is essential in mobile monitoring environments. Hence, ActiveMQ was modified so that it was able to start up without an existing Internet connection. Furthermore, it was modified so that the status of the network interface would be checked and JMS queue connectivity established or reestablished automatically. The original version of ActiveMQ would not attempt to reestablish connectivity by itself.

The following explains the important components of ActiveMQ that are used here. A *broker* is responsible for managing queues and topics. It receives message from *producers* which connect to it and delivers them to the corresponding *consumers*. *Brokers* allow *producers* and *consumers* to use various protocols to connect to it. In ActiveMQ these connectivity entities are defined as *transport connectors*.

Multiple *brokers* can form a *network of brokers* using *network connectors*. This allows the use of *distributed queues* and is the configuration used to connect the VNOC and MRN. In order to be flexible the configuration of a *network bridge* is initiated by the MRN. Establishing a *duplex connection* then enables messages to be forwarded in both directions. The advantage for the scenarios of interest here is that the VNOC does not need to be reconfigured every time a new MRN is set up.

ActiveMQ allows several different protocols (e.g., AMQP by AMQP Working Group, 2009, OpenWire by Apache Software Foundation, 2009, REST by Fielding, 2000, Stomp by Strachan, 2005, XMPP by Saint-Andre, 2004), to be used for message transmission. The system here envelops SOAP messages in JMS messages. Then, ActiveMQ makes use of the OpenWire protocol by Apache Software Foundation (2009), which is an optimized binary compressed format tailored to efficient management of JMS *queues* and *topics* as well as network connectivity, to transport them. This is another advantage of using ActiveMQ in scenarios of interest here since it makes sure that communication between brokers is bandwidth efficient.

3.4. Distributed queues

Connections from the VNOC to the MRN and vice versa are *point-to-point* which corresponds to *queues* in JMS. *Queues* can be distributed across several *brokers*. Whenever the *brokers* are connected to each other they exchange information about which *broker* has the *consumer* and the other *brokers* forward their queue messages to that *broker*. In the following paragraphs we explain how the two common types of message exchanges work using distributed queues and how they are used here.

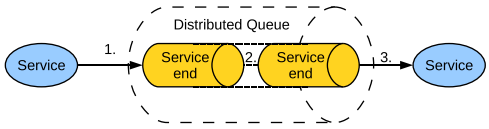

Fig. 6. One-way JMS message transmission.

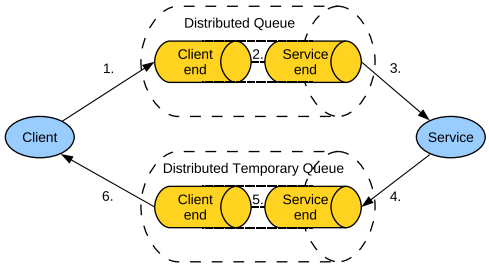

Fig. 7. Two-way JMS message transmission.

Notification messages require only one-way communication as shown in Fig. 6. Within the system a web service acts as a *producer* and puts the notification onto the queue which corresponds to the web service it wants to notify. This is done by using the specified *transport connector* to connect to the local *broker* and deliver the message to it. The *broker* then puts the message on the *queue end* that it manages. Whenever the *broker* can contact the *queue end* with the consumer it forwards the message. The receiving web service uses a listener to detect when its *queue* at the broker contains new messages. It then uses its local *transport connector* to consume the notification.

Control messages that are sent by the VNOC to the MRN are good examples of two-way communications. As shown in Fig. 7 in a request–response scenario the client creates a *temporary queue* at its local broker that only itself knows about. This is where the response message will be put. The request then follows the usual path from the local *transport connector* to the local *broker*, from the local *broker* to the *broker* with the specified *consumer* and then using the remote *transport connector* to the according web service. The JMS message that is transmitted contains a *ReplyTo* field with the *temporary queue* that is used for the response. The response is then sent back using the web service's local *transport connector*, local *broker*, remote *broker* until it is consumed from the *temporary queue* by the original client.

In this system all of the queue creation, message queuing and brokering are transparent to the web services. Whenever asynchronous communication using JMS is required the clients and web services simply use JMS addresses instead of the default HTTP ones; these can be set in their configuration files. This makes the solution scalable and flexible since a *store-and-forward* mechanism does not need to be implemented in each web service but is provided by an ActiveMQ JMS message broker. This holds true not only for SOAs that make use of web services but is also applicable to other systems.

4. Results

The system described above has been successfully tested in field trials. Results are presented from two experimental conditions: a stationary scenario and a mobile scenario. The mobile scenario was carried out by mounting the equipment onto a train. Throughout the experiments communication between the VNOC and the MRN in the TSSN is using a satellite link, in this case Iridium at 2.4 kb/s. Both scenarios were performed as part of a longhaul trial in Mexico (see Fig. 8).

The MRN sensor node was set up on a Panasonic ToughBook placed inside a locomotive that used a transceiver to communicate with Hi-G-Tek electronic seals (see Hi-G-Tek, 2009) and an Iridium satellite modem to transmit information to the VNOC and the TDE which were located in Overland Park, KS. The hardware setup is discussed in detail in Fokum et al. (2010).

The focus of this paper is on showing that JMS can be used in mobile monitoring environments. Monitoring the ActiveMQ message queue (see Figs. 9–11) shows when messages were put on the queue and taken off. These results also show when we had an ActiveMQ connection and JMS messages could be transported through the distributed queue. Table 1 shows performance data for the presented cargo monitoring scenario.

4.1. Stationary

Figures 9 and 10 show tests results acquired from when the MRN was set up in a rail yard but not mounted on a train and not moving. The time each message spent on the distributed queues (shown as "On Queue in sec") as well as when the MRN and the VNOC brokers had a JMS network bridge established and were, therefore, fully connected (shown as "Bridge established") is displayed. It can be seen that while all messages were successfully transmitted the time a message was on a queue is dependent on the quality of the satellite connection.

4.2. Mobile

The more interesting scenario is to use TSSN in a mobile monitoring environment. For this purpose the MRN was deployed on a train and sensors attached to cargo containers. Figure 11 shows results of roughly the first hour (8:30–9:30am, July 30, 2009) of the longhaul trial along the path shown in Fig. 8. Due to a hardware problem after about 9:30am the system clock synchronization is significantly off and the remaining data that was logged for off-line performance analysis, especially time measurements, cannot be used. However, it is important to emphasize that the experiment was successful because the TSSN kept operating correctly, i.e., messages were transmitted and received using ActiveMQ for more than 32 h.

Figure 11 shows in detail when messages were put on a queue, when they were consumed and at which time the JMS network bridge (actual connectivity) was established. A comparison of times message required to be transmitted from the MRN to the VNOC is shown in Table 1. Whenever connectivity, in ActiveMQ called a bridge, is established the actual message transmission takes about 11.6 s on average. This is in stark contrast to when the satellite link is down and needs to be established before sending out messages. In that case it took 616.2 s on average with the slowest message being received 1273.1 s or more than 21 min after it was sent.

The key characteristic here is the availability of the satellite link. The trial was performed in a mountainous environment where the satellite view was partially obstructed and hence the times measured may not be the same in a different geographic region. Looking at the average case of about 7 min per message transmission though the system is found to be in range of mobile monitoring environments.

A comparison of these results to a previous shorthaul trial which is described in detail in Fokum et al. (2010) and Kuehnhausen (2009) is shown in Table 2. During the shorthaul the MRN was continuously connected to the VNOC using a GSM modem with a peak throughput of about 700 kb/s. Looking at the minimum times and assuming this as the best case scenario the satellite configuration is slower by a factor of about 13.

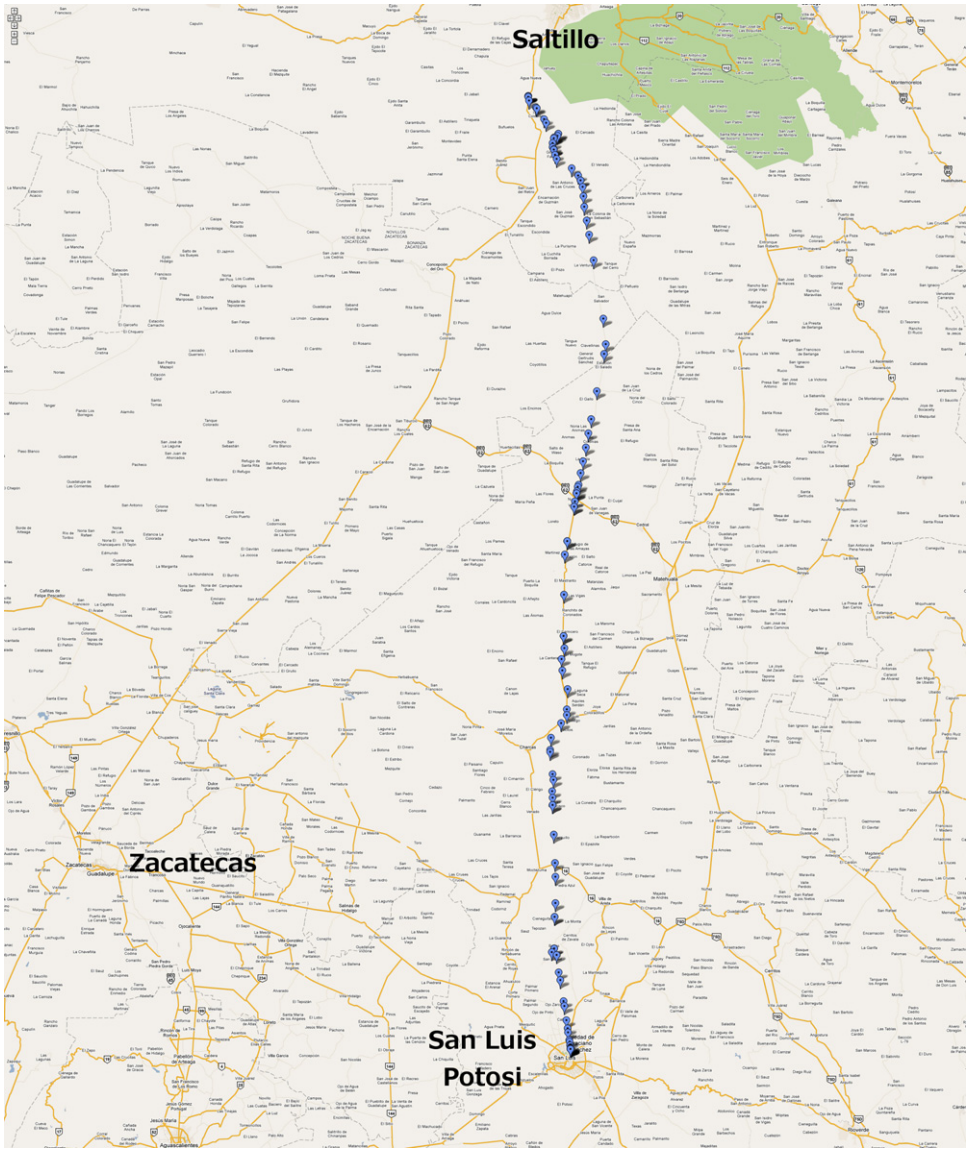

Fig. 8. Route for longhaul field trial, route starts at San Luis Potosi and ends approximately 210 mile down the track (from Google Maps).

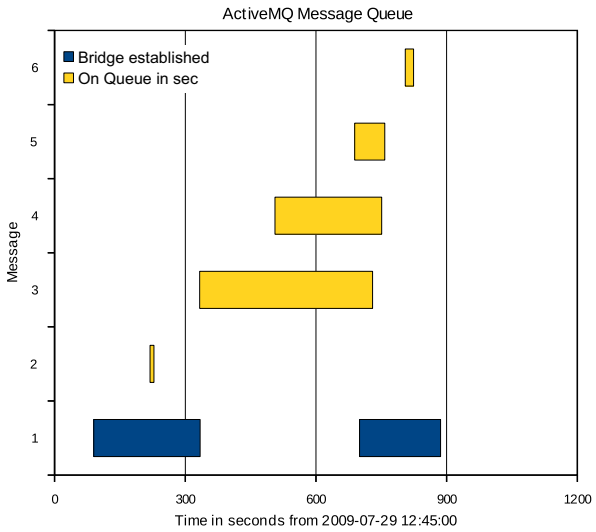

Fig. 9. Initial test.

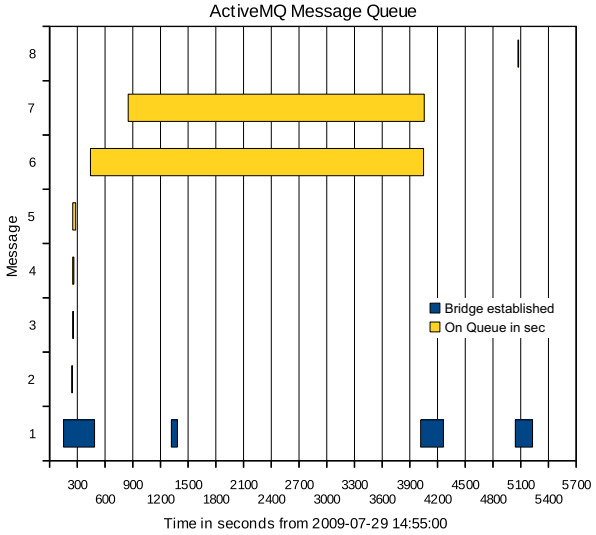

Fig. 10. Follow-up tests.

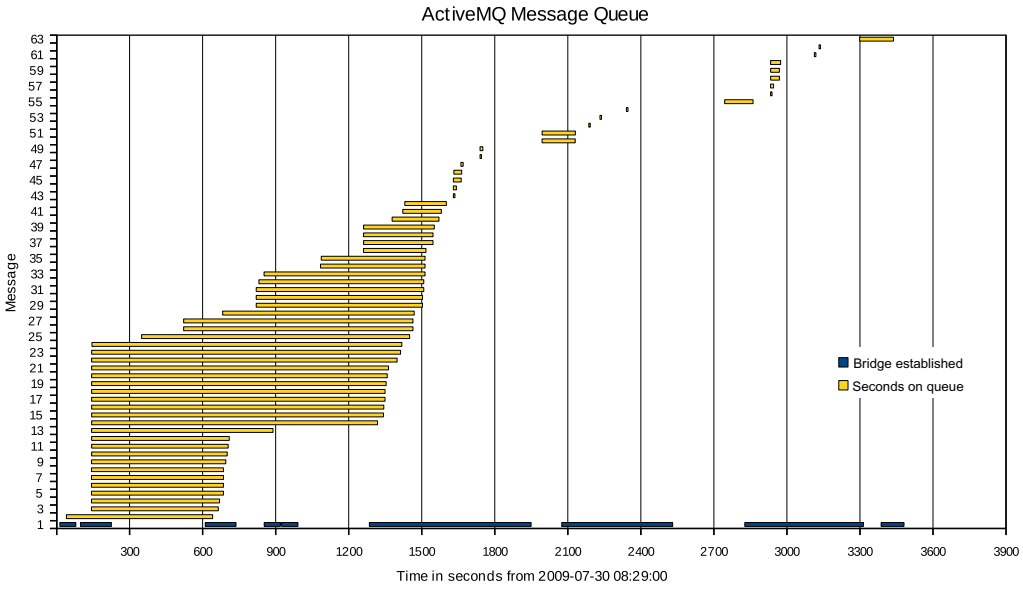

Fig. 11. ActiveMQ message queue for mobile scenario.

Table 1
Elapsed time from MRN to VNOC during trial in seconds.

Case	Min.	Max.	Mean	Median	Std. dev
Link down	31.29	1273.1	616.26	553.23	411.36
Link up	5.85	40.53	11.62	6.02	10.77
Average	5.85	1273.1	481.90	430.97	441.86

Table 2
Comparison of elapsed time from MRN to VNOC of longhaul to shorthaul trial in seconds.

Case	Min.	Max.	Mean	Median	Std. dev
Shorthaul	0.45	2.90	1.89	1.94	0.62
Longhaul	5.85	1273.1	481.90	430.97	441.86

The sensor node could communicate with all the sensors it monitored. It has to be noted that multihop communications were not supported by the sensors even though this feature is essential for a rail-based sensor network. Satellite communication was working whenever a clear view of the sky was available and the system reliably attempted to "call home" every 10 min based on a timer. The TSSN worked as designed. It was able to propagate detected events through the entire system to the end user and perform sensor management tasks reliably.

5. Conclusion

Using JMS in mobile monitoring environments addresses asynchronous communication, message security and integrity, and scalability for the scenarios of interest and it has been shown here to work in harsh environments. We showed that JMS technology can be utilized to provide drop-in connectivity between distributed and delay tolerant systems. Previously JMS was primarily used in always-connected high-bandwidth scenarios, here we demonstrated that its concepts and techniques are useful in mobile monitoring environments as well.

JMS provides a transparent asynchronous communication model that is scalable and flexible since a *store-and-forward* mechanism does not need to be implemented in each component but is provided by a JMS message broker. This is done using distributed queues that are managed by network connectors as described above.

Since JMS allows the transport of all types of messages including SOAP messages, web service specifications were used here to provide features such as *end-to-end* message security and integrity. In terms of scalability JMS makes it possible to connect disparate systems with limited effort without having to implement *store-and-forward*, *resend* mechanisms and security again and again. Furthermore, JNDI and support for different messaging models enhance scalability for JMS-based systems.

In this paper, we presented a new and flexible approach to deal with challenges in mobile monitoring environments such as intermittent and low-bandwidth communication. This approach that utilizes the features of JMS providing asynchronous communication, message security and integrity, and scalability.

Acknowledgments

The authors would like to thank various reviewers and Daniel Fokum for their constructive and helpful comments.

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.

References

Allard D. Development of a ground data messaging infrastructure for the mars science laboratory and beyond. In: IEEE aerospace conference; 2007. p. 1–8.

AMQP Working Group. Advanced Message Queuing Protocol (AMQP) version 0–10 Specification. Specification; 2009, 〈http://www.amqp.org〉.

Apache Software Foundation. OpenWire Version 2 Specification. Specification, Apache ActiveMQ; 2009, 〈http://activemq.apache.org/openwire-version-2-specification.html〉.

Easton P, Mehta B, Merrick R. SOAP over java message service 1.0. W3C working draft, W3C, July 2008, 〈http://www.w3.org/TR/2008/WD-soapjms-20080723〉.

Fielding RT. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine; 2000.

Fokum D, Frost V, Kuehnhausen M, DePardo D, Oguna A, Searl L, et al. An open system transportation security sensor network: field trial experiences. IEEE Transactions on Vehicular Technology 2010;59(8):3942–55.

Fremantle P, Patil S, Davis D, Karmarkar A, Pilz G, Winkler S, et al. Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1. OASIS standard, OASIS, June 2007, ⟨http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.pdf⟩.

Gudgin M, Gudgin M, Hadley M, Rogers T, Rogers T, Hadley M. Web services addressing 1.0 – SOAP binding. W3C recommendation, W3C, May 2006a, ⟨http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509⟩.

Gudgin M, Hadley M, Rogers T. Web services addressing 1.0 – core. W3C recommendation, W3C, May 2006b, ⟨http://www.w3.org/TR/2006/REC-ws-addr-core-20060509⟩.

Gudgin M, Lafon Y, Karmarkar A. Resource representation SOAP header block. W3C recommendation, W3C, January 2005, ⟨http://www.w3.org/TR/2005/REC-soap12-rep-20050125/⟩.

Hapner M, Burridge R, Sharma R, Fialli J, Stout K. Java(TM) Message Service Specification. Specification, Sun Microsystems; 2002, ⟨http://java.sun.com/products/jms/⟩.

Hi-G-Tek; 2009, ⟨http://www.higtek.com/⟩.

Hongsheng L, Yu W, Yongzhong D, Zhongxiao P. Implementation of network-computing and nn based remote real-time oil well monitoring system. In: International conference on neural networks and brain. ICNN&B '05, vol. 3, October 2005. p. 1810–14.

Kuehnhausen M. Service-oriented architecture for monitoring cargo in motion along trusted corridors. Master's thesis, University of Kansas, July 2009.

Lafon Y, Mitra N. SOAP version 1.2 part 0: Primer (second edition). W3C recommendation, W3C, April 2007, ⟨http://www.w3.org/TR/2007/REC-soap12-part0-20070427/⟩.

Lawrence K, Kaler C, Nadalin A, Monzillo R, Hallam-Baker P. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). OASIS standard, OASIS, February 2006, ⟨http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf⟩.

Levinson E. The MIME Multipart/Related Content-type. RFC 2387 (Proposed Standard). August 1998, ⟨http://www.ietf.org/rfc/rfc2387.txt⟩.

Mendelsohn N, Ruellan H, Gudgin M, Nottingham M. XML-binary optimized packaging. W3C recommendation, W3C, January 2005, ⟨http://www.w3.org/TR/2005/REC-xop10-20050125/⟩.

Musolesi M, Mascolo C, Hailes S. Adapting asynchronous messaging middleware to ad hoc networking. In: MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing. New York, NY, USA: ACM; 2004. p. 121–6.

Nottingham M, Ruellan H, Mendelsohn N, Gudgin M. SOAP message transmission optimization mechanism. W3C recommendation, W3C, January 2005, ⟨http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/⟩.

Object Management Group. Data Distribution Service for Real-time Systems Version 1.2. Specification; 2007, ⟨http://www.omg.org/spec/DDS/1.2/⟩.

Pardo-Castellote G. 19-22 2003. Omg data-distribution service: architectural overview. In: Twenty-third international conference on distributed computing systems workshops, 2003. Proceedings; 2003. p. 200–6.

Saint-Andre P. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard). October 2004, ⟨http://www.ietf.org/rfc/rfc3920.txt⟩.

Snyder B, Bosanac D, Davies R. ActiveMQ in action. Manning Publications; 2009.

Strachan J. Stomp Protocol Specification, Version 1.0. Specification, FuseSource; 2005, ⟨http://stomp.codehaus.org/Protocol⟩.

Sun Microsystems, Java Naming and Directory Interface Application Programming Interface (JNDI API). Specification; 1999, ⟨http://java.sun.com/products/jndi/⟩.

Vahdat A, Becker D. Epidemic routing for partially-connected ad hoc networks. Technical Report, Duke University; 2000.

Vinoski S. Advanced message queuing protocol. IEEE Internet Computing 2006;10(6):87–9.

Vollset E, Ingham D, Ezhilchelvan P. JMS on mobile ad-hoc networks. In: Personal wireless communications. PWC, Springer-Verlag; 2003. p. 40–52.