Tracking and Tracing Containers through Distributed Sensor Middleware

Klaas Thoelen, Sam Michiels, Wouter Joosen
IBBT-DistriNet - Department of Computer Science - K.U. Leuven
Celestijnenlaan 200A
3000 Leuven, Belgium
{klaas.thoelen,sam.michiels}@cs.kuleuven.be

ABSTRACT

In a container transport system, wireless sensor networks (WSNs) can be used for monitoring products while they are being transported. To be commercially interesting, these WSNs must be integrated with enterprise systems of various actors in the supply chain. The need for interoperability between networks and partners, the heterogeneity of WSN technologies being used, and the mobility of sensor nodes make this integration far from trivial. This paper presents lessons learned from a research project in collaboration with industry in which we developed a prototype middleware solution for container transport. The prototype triggered valuable feedback which is highly relevant to consider when designing middleware for realistic sensor applications.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures*

General Terms

Design, Management

Keywords

Wireless sensor networks, middleware, integration, enterprise infrastructure, supply chain management

1. INTRODUCTION

To increase the quality of control during transport, cargo containers can be equipped with wireless sensor nodes that are capable of (i) collecting environmental data like temperature, humidity and light intensity, (ii) controlling actuators such as the air conditioning system of a temperature-controlled container, and (iii) running services like for instance localization, data aggregation, or distance measuring.

AUTONOMICS 2008, September 23-25, Turin, Italy

DOI 10.4108/ICST.AUTONOMICS2008.4666

Since containers are constantly being transported among storage sites, this causes the formation of ad-hoc wireless sensor networks (WSNs) at each site, monitoring the current local collection of containers.

In order to become useful to logistic companies, WSNs must be integrated with their existing enterprise servers, databases, and gateways [1, 3]. Enterprise Resource Planning Systems and Transport Management systems can benefit considerably from the data supplied by the WSNs. Customs declaration of content and initial security checks, for instance, can be handled automatically by inspecting a container's monitored data. This would substantially reduce the traversal time of containers at each storage site and ultimately lead to faster delivery to the end-user.

In addition to this end-to-end integration, the challenge is to integrate all partners in a supply chain [12]. Every partner is only a link of the total chain, just as the data collected by the sensor nodes at its premises is just a piece of the total monitoring trace of the container. Enterprise systems of various partners need to be coupled to leverage on the end-to-end integration and (i) enable the creation of full monitoring traces of containers along the supply chain and (ii) allow customers to easily check quality conditions at all times during transport.

The WSN landscape, however, is still far from standardized and various WSN types might be used with different data and messaging formats. In a worst case scenario, every storage site in the supply chain deploys a different WSN type, making the initial goal of monitoring a container during transport extremely difficult.

In addition, integrating WSNs and existing enterprise systems creates large-scale and highly heterogeneous network infrastructures and implies complex end-to-end software deployment. Deploying software at various nodes in the end-to-end network infrastructure is not a trivial task to accomplish; middleware support is needed to enhance application development and network administration in such environments.

In conclusion, it can be stated that the development of a fully integrated platform for container transport in which (i) various WSN types are used, (ii) sensor nodes need to be able to migrate between WSNs while guaranteeing continuous monitoring traces, and (iii) supply chain partners need to be able to exchange the monitored data, is not a trivial task.

The contribution of this paper is the presentation of the experiences we gained from developing an end-to-end middleware for transport of cargo containers. Furthermore, we report on a number of key issues that have been identified

in the feedback from involved industrial partners.

The paper is structured as follows. In section 2 we describe the key challenges of developing an end-to-end middleware for container transport and present an architecture which handles these challenges. In section 3 we discuss our prototype implementation of this architecture. After a discussion of industry feedback (section 4) and related work (section 5), we end the paper with concluding remarks in section 6.

2. TOWARDS A DISTRIBUTED MIDDLEWARE FOR LOGISTICS

Before going into the details of the middleware architecture (section 2.2), we first describe the key challenges in the development of a distributed middleware for container transport.

We consider one company of a supply chain, which is divided over multiple sites, like for instance a warehouse, a harbor dock and an airport terminal. Each individual site of the company uses a certain WSN type (e.g. Berkeley motes [13], Sun SPOTs [10] or Sentilla nodes [9]) to monitor the local containers, but different sites might use different WSN types. All monitoring data is stored in a central database situated at the company's headquarters and is used in the company's backend infrastructure to aid in transport and resource planning.

2.1 Key challenges

In order to support the integration of WSNs with the company's backend infrastructure the distributed middleware needs to handle the following challenges:

1. **End-to-end interoperability:** successful end-to-end integration requires coupling sensor nodes, gateways, backend servers and databases to enable fluent bi-directional interactions.

2. **External data exchange:** an interface must be provided through which monitored data can be exchanged in a common data format with interested parties like supply chain partners, customs, or a national food agency.

3. **Heterogeneity of data and messaging formats:** each WSN type uses a custom format to represent monitoring data and a custom messaging service to communicate with their gateway. For instance, different sensor nodes might measure temperature in Celsius or Fahrenheit and use different message structures to forward the collected data. These custom formats need to be abstracted and converted into standardized formats so that data exchange between gateways and backend infrastructure is carried out in a common data format through a standardized interface.

4. **Intermittent network connection of sensor nodes:** when network connectivity is lost, sensor nodes must be able to temporarily store readings locally in order to guarantee a continuous trace of monitored data. This is particularly useful when a container is transported between sites by a carrier without network connectivity. As soon as connectivity is detected again, the stored readings need to be forwarded towards the backend for processing and persisting.

5. **Inter-network mobility of sensor nodes:** when containers are transported among sites which have a similar WSN technology deployed, the sensor nodes must be able to migrate between the WSNs without losing data or requiring manual intervention.

2.2 Distributed middleware architecture

Based upon the challenges discussed in the previous section and the nature of the problem context, we propose a 4-tiered architecture as illustrated in Figure 1.

The nodes to which middleware is deployed, are very dissimilar and range from resource-constrained sensor nodes to backend servers. Consequentially, the sets of software components that compose the middleware will differ among the various tiers.

Three service layers. The middleware at each tier of the deployment consists of 3 service layers. On top of the middleware, an application layer is situated which uses the services provided by the middleware.

First of all, the Distribution Services layer takes care of end-to-end communication; it provides reliable messaging in the WSN to avoid data loss and a message queue to buffer bursts of data being forwarded towards the backend.

Secondly, the Common Services layer provides general services such as logging, encryption and authentication and primarily takes care of the respective non-functional requirements posed by a container monitoring application.

Thirdly, the Application-Specific Services layer consists of services like for instance sensing and caching in the WSN tier, data conversion in the gateway tier and aggregation in the backend tier. The caching service ensures that sensed temperature data are cached temporarily in case they cannot immediately be forwarded towards the backend. The conversion service at the gateway converts data messages from the WSN into standard messages; the aggregation service, for example, triggers an alarm when the measured temperature exceeds a given threshold.

Representative use-case scenarios. We illustrate the workings of our architecture by means of the sequence diagram in Figure 2. We first discuss the initialization of the sensor node after which we focus on a single sensor reading being performed at a sensor node in range of a gateway.

When a container arrives at the company's dock, it is equipped with a sensor node. Besides physically attaching the node to the container, this involves administrative coupling of the node's unique identifier to the container representation in the backend server application. As soon as the node's monitoring application is activated the connectivity service starts receiving beacons from the local gateway and the node will integrate itself into its WSN. At the same time, the monitoring application initiates the sensing service to start the collection of temperature readings. After each reading, the reliable messaging service is instructed to forward the data towards the gateway.

The reliable messaging service at the gateway will in turn acknowledge each received data message and forward it to the conversion service. The collected data is then transformed into a standardized data format and meta-data such as time of arrival is added. The standardized data packet is then added to a message queue where it will be picked up by the aggregation service of the backend application. In the

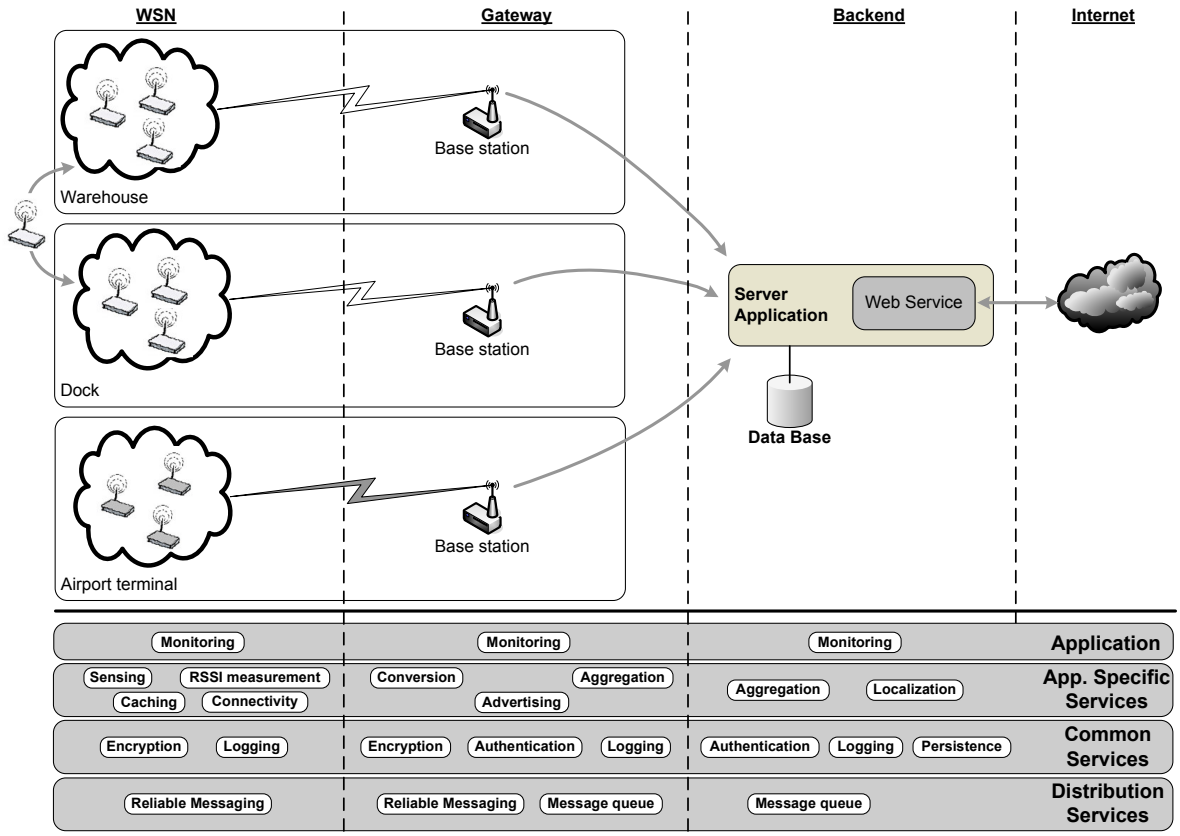

Figure 1: Deployment view of the end-to-end integration at the diverse sites of the company together with an architectural view of the services offered by the middleware.

backend tier, the aggregation service will process the sensor reading which possibly triggers an alarm, and call the persistence service to store it in a database. All this monitoring data, which originated at the sensor node, is also coupled to the container allowing us to request a monitoring trace based upon the container's identifier instead of the sensor node's identifier.

In another scenario, a truck without connectivity provisions might transport the container towards the warehouse. In this case, the sensor node's connectivity service would no longer receive beacons from a gateway and instruct the sensing service to use the caching service and locally store the temperature readings in flash memory.

When the truck arrives at the warehouse, the sensor node starts receiving beacons again and registers itself in the new WSN. The detection of a new gateway also triggers the caching service to start transmitting its stored data and freeing used memory for future caching. This cached monitoring data is again forwarded towards the backend where it is persisted no differently then the locally collected monitoring data.

The data, stored at the central database, can then be used to let interested parties, like supply chain partners, customs agencies or the national food agency, request a full monitoring trace of the container. This trace can be requested through the web service we provided; it contains all monitoring data of the container, no matter at which site of the company it was collected.

Handling key challenges. Our architecture handles the challenges described in section 2.1 as follows.

The **end-to-end interoperability** is fulfilled primarily by the Distribution Services layer which enables neighboring tiers to communicate with each other via common distribution services.

The provided web service enables interested parties to retrieve the data collected by the WSNs, thus taking care of the **data exchange** requirement.

The **heterogeneity** challenge is handled by the gateway's conversion service by abstracting the different data formats used in the various WSNs. This allows monitoring data to be forwarded to the server application in a standard data format and through a standard interface, regardless of the WSN type used.

The **intermittent network connection** of the sensor nodes is in a large part handled by the caching and connectivity services at the application specific middleware layer of the WSN-tier. A failing connection is detected by the connectivity service which triggers the use of the caching service. Like this, a continuous monitoring trace of the container can still be guaranteed, as long as the sensor node is frequently in range of a gateway to empty its cache memory.

Mobility of sensor nodes is handled by a combination of services at different tiers. As long as a node's connec-

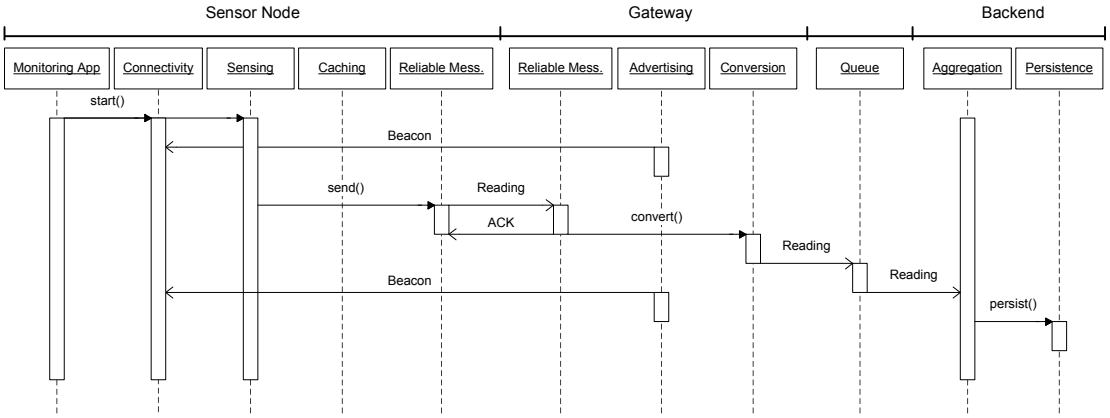

Figure 2: Sequence diagram of a single sensor reading of a sensor node in range of a gateway.

tivity service receives beacons from the advertising service of a gateway, it considers itself part of the gateway's WSN. Only when a node no longer receives beacons from its former gateway, but does receive beacons from another, will it migrate to this new WSN and register itself there. In this way, containers can be moved among sites, using similar WSN technology, without manual intervention to guarantee further continuous monitoring.

3. PROTOTYPE: THE MULTITR@NS PLATFORM

Based upon the challenges and the architecture we described in the previous sections, we implemented a prototype (Figure 3) as part of the MultiTr@ns project [6]. The two lower layers of our middleware were composed by reusing state-of-the-art software components provided by TinyOS and the Java Enterprise Edition (Java EE) platform; the application-specific services and the monitoring applications were implemented by us.

We developed two TinyOS monitoring applications which are installed on a set of Crossbow's micaz sensor nodes; one for nodes collecting temperature measurements at containers and another for base stations of the WSNs. Both applications make use of a set of middleware services deployed locally on the respective tier.

Each sensor node is programmed with a system-wide unique node address, and one gateway is deployed for each site of the company that is considered. We implemented a connectivity service which receives beacon messages containing a network identifier. This identifier is included as meta-data in all of the sensor node's communication with the gateway. As long as beacons are received, the monitoring data is transmitted reliably towards the base station by the Packet Link Layer of TinyOS. When the base station is out of range, the beacon timer will fire which triggers the application to delegate data to the caching service until new beacons are received.

The gateway tier consists of a base station node and a Java EE Application Client. The latter handles the conversion between data formats and forwards data to the server application. While we had to implement the application specific conversion service, we could reuse the Java Message Service (JMS) included in Java EE to handle the communication towards the server. Upon initialization, our advertising service, situated at the base station, receives a network identifier from the application client and starts broadcasting this by means of the beacon messages.

The backend tier consists of a GlassFish server which runs a Java EE application and stores monitoring data in a Derby database by using the Java Persistence API (JPA). Our custom monitoring application keeps a software representation of the company up to date. This representation associates deployed sensor nodes with containers and all monitored data. Because our prototype includes one application server and database for the whole company, data can easily be passed among the various sites and a complete monitoring trace spanning the container's presence in the company is easily generated.

The monitoring trace, together with the current state of each deployed WSN and some administrative tasks, are made accessible through a web service that was implemented using the Java API for XML Web Services (JAX-WS).

4. DISCUSSION

The experience of developing the MultiTr@ns prototype and various discussions with the industrial partners involved in the MultiTr@ns project, triggered valuable feedback. We identified the following main issues:

- **Sensor node and data discovery.** While our current prototype illustrates a one-server setup per company, it is more realistic that every site will deploy its own server and WSN to establish full control of the infrastructure and to secure private company data. On an even larger scale, multiple companies might form a fully integrated supply chain in which a single sensor node monitors a container throughout the total chain. This implies that while containers are being transported, data is stored in the current handler's database ultimately resulting in federated data management along the supply chain. It might also not always be clear at which company the container is currently located and thus which network to address for

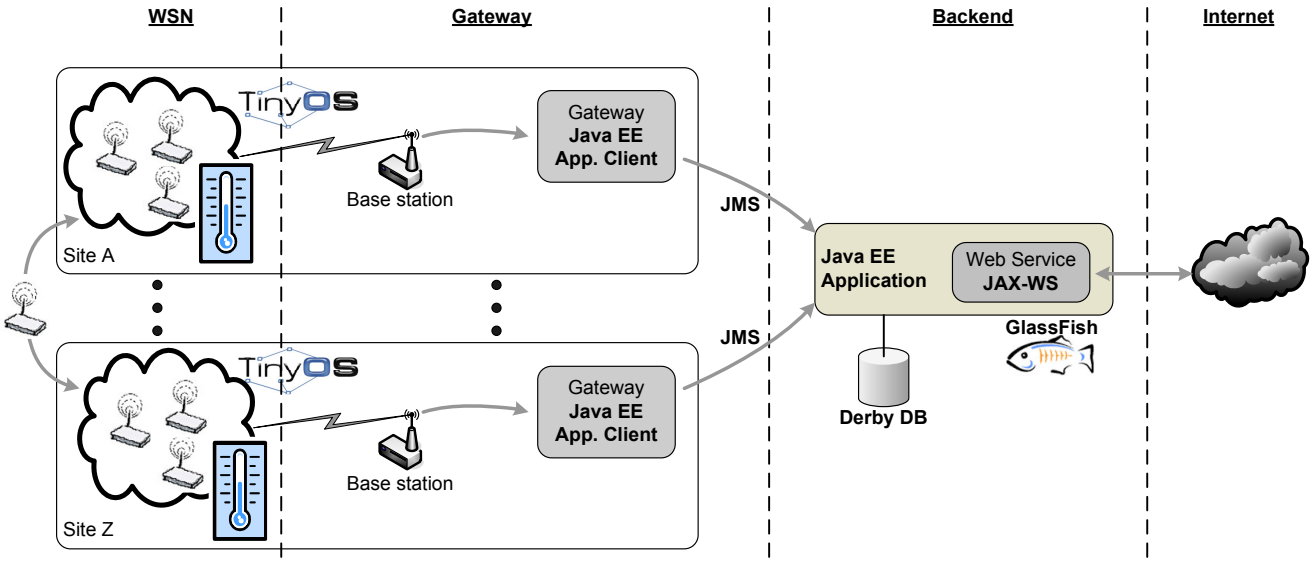

Figure 3: Deployment view of the implemented MultiTr@ns platform.

current data. To handle these situations, a resolve service is needed which matches a container's unique identifier to the addresses where current and/or historical data can be retrieved. The Object Name System and Discovery Services of the GS1 EPCglobal Architectural Framework [2] provide this kind of functionality in RFID systems but need further definition to be applicable in a WSN context.

- **Support for heterogeneity in WSNs.** Forwarding agencies will more then likely not all favor the same sensor node type to monitor their containers. At harbor docks for instance, this will cause a variety of sensor nodes which need to communicate with the local gateway. This means future gateways must be able to convert a range of messaging and data formats into standard ones. Such a heterogeneous situation would benefit considerably from further WSN standardization and more specific, the use of a common networking stack. This requirement is partly handled by the IETF 6LoWPAN standard [4] which, while still in development, integrates WSNs into the Internet by extending IPv6 support towards the sensor nodes.

- **Dynamic reconfiguration.** During container transport, sensor nodes migrate between different WSNs. A simple and straightforward requirement like encrypted transmission of the monitoring data is however not a trivial task to accomplish. In different WSNs, different encryption keys might be used and thus need to be exchanged securely between nodes and gateways. To make things even worse, the encryption algorithms used by a sensor node and its current gateway might not be the same, requiring a new encryption service to be deployed on either the sensor node or the gateway at runtime. This requires middleware support for dynamic reconfiguration. Technologies like over-the-air software distribution [14], OSGi's orchestration [7] and the current generation of sensor nodes, like Sun SPOTs [10] and Sentilla motes [9], which run a Java VM out-of-the-box, make this a more feasible challenge.

- **Cross-cutting requirements.** Non-functional services, like for instance a logging service, are crosscutting the functional services like temperature sensing. They are not always required and/or might need frequent customizations to various application requirements. Imagine a logging service in the WSN tier: if logging is not required, it is better totally shut down to save on energy and memory usage; if critical data is to be monitored, data must be logged in a secure, encrypted manner. Such services are mostly common to all tiers and need to be implemented and executed in a non-disruptive manner to not interfere with the functional services. Interception of the message flow at crucial places or introduction of Aspect-Oriented Software Development (AOSD) seem promising solutions but need more in depth research as towards feasibility, especially in the resource constrained WSN tier.

- **Support for security.** Although we clearly acknowledge the need for security, the current architecture and prototype pay little attention to it. Various situations can be identified that need security to be dealt with [5] like for example (1) malicious node registration which might corrupt the monitoring data or flood the WSN preventing real data throughput, (2) user specific access levels of historical and real-time data, or (3) user specific rights for service deployment. Handling these situations requires the deployment of additional security components in both the middleware and the underlying networking stack.

We are currently preparing a second generation architecture in which we try to integrate state-of-the-art approaches to handle a number of issues discussed.

5. RELATED WORK

Previous efforts have been made to integrate WSNs with enterprise infrastructure in a container monitoring context.

SAP introduces the Enterprise Integration Component (EIC) [1] which provides similar functionality as our middleware, like end-to-end integration, abstraction of different WSN types and persistence. The EIC implements two interfaces; one towards the WSNs and one towards backend applications, and can be compared to a combined gateway- and backend-tier of our architecture. Although they state sensor node mobility as a requirement, this is not further elaborated upon.

IBM's Secure Trade Lane [8] aims at making container shipments more predictable and secure. The monitoring of containers is done by so-called TREC devices which contain a long range radio like GSM/GPRS and satellite. This long range radio causes node migration among WSNs to be a non-issue since connection with the backend is always available. The Secure Trade Lane does not support WSN heterogeneity and uses only TREC devices, which requires a strong cooperation among supply chain partners. They state that a central database is unacceptable in an environment where so many parties are involved due to privacy issues. A service oriented architecture is proposed in which the owner of the data can define whom to share data with and under what conditions.

The MASC (Monitoring and Security of Containers) system [5] introduces a container monitoring system that uses recognized security organizations as trusted third parties (TTPs) that control access to data collected by sensor nodes monitoring containers. Although data is stored in a central database, controlled by a TTP, and node migration is not elaborated upon, they introduce a tree structure which represents the supply chain. At the top of the tree, a forwarding agency, holding the overall responsibility of the container, is delegating parts of the containers transport to several lower-level logistic service providers. This means that the federated data, located at the logistics service providers, might be aggregated by the root forwarding agency, thus scaling down the challenge of sensor data discovery.

The EPC Sensor Network [11] introduces a combined global standard infrastructure for WSNs and RFID systems based on the EPCglobal Architectural Framework [2] to support data sharing between partners. The EPCglobal Architectural Framework is a collection of standards and services which enhance the supply chain through the use of Electronic Product Codes (EPC) and RFID systems. The EPC uniquely identifies a product through a hierarchical identifier comprised of a manufacturer id, a product code and a serial number. The EPC Sensor Network tries to add WSN support to the initial goals of the framework. Logically they also refer to the Object Name System and Discovery Services of the framework for managing the federated data, but as stated before, even for RFID systems, the standards do not fully specify how this should be solved.

6. CONCLUSION

In this paper we presented an end-to-end middleware that supports the development of applications that monitor the status of containers during their transport. We identified key challenges that an end-to-end middleware for container transport must handle and we presented the design of a multi tiered middleware architecture. Our prototype implementation leverages on state-of-the-art software components (TinyOS, Java EE).

In the context of the MultiTr@ns project, we evaluated our middleware solution and discussed it with various industrial partners; this feedback identified a number of key issues which are highly relevant when designing end-to-end middleware for realistic applications.

This paper shows that developing realistic sensor applications is still far from trivial; many challenges have to be handled, some of which were discussed in this paper. The prototype we have developed illustrates, however, that it is feasible to make substantial progress by leveraging on state-of-the-art middleware components, in combination with specific services being added.

We will continue this approach and aim for bridging the gap between industry requirements and middleware support to develop sensor applications in an efficient manner. The emergence of a new generation of wireless sensor nodes running more advanced (Java enabled) platforms (cf. SunSPOT [10] or Sentilla [9]) confirms the opportunities in the development of enhanced sensor middleware and illustrates increasing support for programming sensor nodes.

7. ACKNOWLEDGEMENTS

The authors are grateful to the members of DistriNet's WSN team for their valuable comments on the paper and for proof reading the text.

Research for this paper was sponsored by IBBT, the Interdisciplinary institute for BroadBand Technology, and conducted in the context of the MultiTr@ns project.

8. REFERENCES

[1] L. Gomez, A. Laube, and A. Sorniotti. Design guidelines for integration of wireless sensor networks with enterprise systems. In *MOBILWARE '08: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–7, 2007.

[2] GS1 EPCglobal. http://www.epcglobalinc.org/standards/ (visited June 2008).

[3] W. Horré, S. Michiels, N. Matthys, W. Joosen, and P. Verbaeten. On the integration of sensor networks and general purpose IT infrastructure. In *MidSens '07: Proceedings of the 2nd international workshop on Middleware for sensor networks*, pages 7–12, 2007.

[4] IPv6 over Low power WPAN (6LoWPAN). http://www.ietf.org/html.charters/6lowpan-charter.html (visited June 2008).

[5] J. O. Lauf and D. Gollmann. Monitoring and security of container transport. In *Proceedings of New Technologies, Mobility and Security*, 2007.

[6] MultiTr@ns: Multimodal transport - mobility and logistics. http://projects.ibbt.be/multitrans/ (visited June 2008).

[7] OSGi Alliance. About the OSGi Service Platform, technical whitepaper, revision 4.1, June 2007. www.osgi.org/wiki/uploads/Links/OSGiTechnicalWhitePaper.pdf.

[8] S. Schaefer. Secure trade lane: a sensor network solution for more predictable and more secure container shipments. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 839–845, 2006.

[9] Sentilla. http://www.sentilla.com/ (visited June 2008).

[10] Sun SPOT. http://www.sunspotworld.com/docs/ (visited June 2008).

[11] J. Sung, T. S. Lopez, and D. Kim. The EPC sensor network for RFID and WSN integration infrastructure. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 618–621, 2007.

[12] A. Talevski, E. Chang, and T. Dillon. Reconfigurable web service integration in the extended logistics enterprise. *Industrial Informatics, IEEE Transactions on*, 1(2):74–84, May 2005.

[13] TinyOS Community Forum. http://www.tinyos.net/ (visited June 2008).

[14] Q. Wang, Y. Zhu, and L. Cheng. Reprogramming wireless sensor networks: Challenges and approaches. *IEEE Network Magazine*, 20(3):48–55, 2006.