Towards a Software Architecture for Sensor Middleware

Sam Michiels
Bart Elen
Wouter Joosen
Pierre Verbaeten

Report CW 466, November 2006

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Towards a Software Architecture for Sensor Middleware

Sam Michiels
Bart Elen
Wouter Joosen
Pierre Verbaeten

Report CW 466, November 2006

Department of Computer Science, K.U.Leuven

Abstract

Developing and deploying end-to-end applications for sensor networks in a realistic (business) context remains highly complex. We identify two main reasons for this complexity: the need for interoperability between applications and the underlying system software, and the need to integrate functionality that runs on different types of hardware platforms. This position paper argues for an integrated, generic software architecture for sensor applications. This architecture is the blueprint of a middleware platform that can be assembled with components and customised for three types of sensor platforms in such a way that it contains minimal but sufficient functionality to meet the application requirements. In this way, software developers can be relieved from the lowest level details, which optimises man power efficiency while still allowing to exploit a system's resource capabilities in the most optimal way.

CR Subject Classification : C.2.4, D.2.11

Towards a Software Architecture for Sensor Middleware

Sam Michiels, Bart Elen, Wouter Joosen and Pierre Verbaeten
DistriNet Research Group, Department of Computer Science,
K.U.Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium
sam.michiels@cs.kuleuven.be

Abstract

Developing and deploying end-to-end applications for sensor networks in a realistic (business) context remains highly complex. We identify two main reasons for this complexity: the need for interoperability between applications and the underlying system software, and the need to integrate functionality that runs on different types of hardware platforms. This position paper argues for an integrated, generic software architecture for sensor applications. This architecture is the blueprint of a middleware platform that can be assembled with components and customised for three types of sensor platforms in such a way that it contains minimal but sufficient functionality to meet the application requirements. In this way, software developers can be relieved from the lowest level details, which optimises man power efficiency while still allowing to exploit a system's resource capabilities in the most optimal way.

1 Introduction

Deploying end-to-end sensor applications remains highly complex given the very dynamic and heterogeneous environments in which they must operate. The dynamism and heterogeneity of the environment require special attention for interoperability between (1) system software and the applications that make use of it, and (2) functional components that reside at highly heterogeneous hardware platforms. With end-to-end sensor applications we mean applications that are distributed over one or more wireless sensor networks (WSNs) and an enterprise back-end, and are driven by application-specific policies.

To illustrate the need for interoperability, we first sketch a realistic end-to-end business example in the context of cargo logistics. Subsequently we zoom out and highlight the generic characteristics that are apply to many similar business scenarios.

Imagine a container harbor in which various applications monitor cargo (e.g. containers) in arriving ships, equipped with a WSN to collect various data about the load, such as location, temperature, or oxygen level.

When a ship arrives at the harbor, its load will be checked by various stake holders such as state authorities, the harbor exploitation company, security organisations, or the owner of specific containers in the ship. Each of these stake holders may want to execute highly different applications on the ships's WSN, for instance, calculating import duties, checking for stowaways, detecting and registering chemical substances, or checking the status and quality of the load.

Stake holders may place extra sensor nodes in the harbor infrastructure (e.g. on docks and bridges, or in the PDAs of harbor employees), which must cooperate with and integrate in with specific WSNs of passing ships. For example, a security organisation that searches for dangerous chemical products might provide powerful detection sensors, or the harbor exploitation firm may have placed GPS enabled anchor nodes that help in localising specific items.

The sketched applications execute parts of their functionality on highly heterogeneous hardware platforms. The business related part (e.g. logistics planning, data logging, tax calculation) will obviously execute on heavy weight enterprise servers, while data retrieval components run on lightweight sensor nodes. We identify three types of sensors that can play a relevant role in complex WSNs:

leaf nodes, coordinator nodes, and travelling nodes. Leaf nodes have installed a predefined set of services which can be managed by external parties. They typically concentrate on data retrieval. Coordinator nodes have enough resources available to offer extra services to other nodes and may act as gateway to external environments. Given their coordinating role, they can take into account business policies that describe how a WSN should behave. Travelling nodes are typically attached to mobile carriers and can dynamically join and leave a WSN. These nodes may extend the capabilities of a WSN, for instance, to detect specific chemical products in the load of a ship.

Two generic characteristics can be derived from the example above. First of all, the example presents a realistic business scenarios where a single WSN must cooperate with multiple applications, and integrate sensor nodes which may have installed different (implementations of) services. By consequence, application functionality and WSN functionality must be interoperable, which means that a vertical integration of both software levels is needed.

Secondly, the example clearly shows the heterogeneous landscape of hardware platforms that host various parts of a sensor application, i.e. on the one hand enterprise servers and, on the other hand various types of sensor nodes. This implies the need for horizontal integration of different (sensor) platforms, following a holistic approach that couples WSNs and traditional networks [3].

Given the context briefly sketched here, we argue that sensor application developers must be offered a generic middleware layer that can be customized for particular sensor types and various applications. Deploying such a middleware in a specific context involves integrating the resource capabilities of the sensor platform and the requirements of the applications running on it. We consider two types of integration: vertical integration, i.e. interoperability between multiple software layers within a single sensor platform by adding business policies or stripping unneeded functionality, and (2) horizontal integration, i.e. assembling for each specific sensor type a customized middleware layer that contains the minimal but sufficient set of components.

The paper is structured as follows. Section 2 discusses the main challenges that must be addressed to offer development support for end-to-end sensor applications: vertical integration of, for instance, business policies, and horizontal integration of functionality. Section 3 zooms in on the challenges related to vertical integration in the context of high-end sensor nodes: Section 3.1 describes the requirements to be supported by the sensor node middleware, while Section 3.2 illustrates first steps towards such a middleware in the context of WSNs. Section 4 summarises the ideas presented in this paper.

2 Research challenges

In addressing the formulated position statement, we focus on two key research challenges: (1) defining a generic software architecture that acts as a blueprint to guide developers in horizontally integrating the functionality of the three sensor types, and (2) including this software architecture as a middleware layer which realizes the vertical integration of application software and the underlying sensor operating system.

2.1 Horizontal integration

The need for an integrated middleware architecture has already been stressed by [4]: "we need to integrate WSNs with other IT systems", the state-of-the-art sensor network solutions "lack an end-to-end, integrated middleware for managing distributed data retrieval". Middleware that is assembled for each of the three types of hardware platforms must allow to compose software that contains minimal but sufficient functionality to meet the application requirements. As such, the middleware must enable software developers to compose the required services (for instance for routing, localisation, or classification) in a resource-effective and consistent way.

From a functional perspective, the middleware must create a balance between state-of-the-art middleware (e.g. CORBA) and the current practice of creating limited, low-level, and handcrafted functionality, which makes complete application development hard. Mapping application requirements to a tailored software composition that supports these requirements is not trivial and it seems to be hard, if not impossible for an application programmer to do this by hand. First of all, in order to fully exploit

the available resources, the selection of functional components must be well balanced, which complicates the use of 'general purpose' components. Secondly, the large scale at which sensor networks are deployed and their highly dynamic character make it very difficult for a programmer to select the most appropriate composition of middleware services [3].

Consequently, when we approach middleware from a software engineering perspective, the middleware must offer first of all a comprehensive set of APIs to enable man-power-efficient application development. However, equally important to make end-to-end application development feasible, is to provide support for assembling software components in such a way that the available resources are optimally exploited. This will require a software architecture which provides a blueprint that can be instantiated as preferred.

2.2 Vertical integration

Given the heterogeneity of the three identified sensor types, vertical integration means hiding system details as much as possible, while still being open for fine-tuning or customising the underlying system in a controlled way. To this end, the middleware should offer support (1) to influence the system's behavior by injecting application specific preferences and stripping unneeded functionality, and (2) to exploit system capabilities in the most effective and efficient way by intelligently matching application requirements and the system's resource capabilities [2].

In other words, vertical integration of applications, middleware, and system software, means that these three levels of abstraction are open for external triggers that optimise the global behavior of the application. For example, in the container transport case, an immigration control application may like to increase the precision of sensor localisation when it receives indications of the presence of stowaways. This could imply that temporarily a more precise, but less efficient localisation service must be used.

The diverse resource constraints imply that the level of middleware abstraction is different for each of the three perspectives. The middleware layer in a sensor node will be much closer to and more tightly coupled with the operating system than an enterprise middleware layer. By consequence, vertical integration should be approached differently in each hardware platform type.

Finding a balance between hiding system details and being open for customisation leads to a broader concern of merging the middleware and the underlying operating systems and network infrastructures. Blair, et al. [1] argue that there is great potential in developing systems that are vertically integrated and can be seamlessly inspected and adapted as a unified pool of component-based functionality.

3 Towards a middleware architecture

By way of validating our ideas, this section describes first research steps in the direction of a middleware architecture that supports both horizontal and vertical integration of functional components. We first present the key characteristics such an architecture must support. After that, we describe the main components of the architecture and show how they support the identified requirements.

3.1 Key characteristics

The three main characteristics of the intended architecture are adaptability, reconfigurability, and uniformity. We explain these characteristics in the context of the three types of sensor nodes identified in the business example, i.e. leaf nodes, coordinator nodes, and travelling nodes.

3.1.1 Adaptability

In order to support applications in the most effective way, leaf sensor nodes must be adaptable by external parties when the execution context changes. This can be the case when the number and type of applications that use a WSN changes during the WSN's lifetime, when the requirements of a single application change dynamically, or when the sensor node itself changes (e.g. low battery).

Thus, in order to optimally exploit the available resources in a leaf sensor node, it must allow to activate and deactivate the services it has installed. For example, when a leaf node on a cargo ship is only used for forwarding data but actually offers services for time synchronisation and localisation

as well, the latter services can be de-activated as long as no application uses them.

3.1.2 Reconfigurability

Coordinator nodes must be reconfigurable in order to take into account various business contexts in which they operate. Depending on the circumstances, data may have to be retrieved as fast as possible, as accurately as possible, as efficiently as possible, etc. By consequence, coordinator nodes must be open for injection of high level descriptions of business policies, and they must offer the mechanisms to interpret and apply them.

For example, owners of specific cargo may ask at regular times where its load is approximately resided in order to keep their clients informed. A reasonable policy may be to answer with the latest position that was cached on a coordinator node, or to use a very efficient but less accurate localisation service. However, when a security organisation in a harbor has indications that the load of an arriving ship contains highly toxic products, it may request very accurate location information, regardless of the efficiency of the implementation. In this case, the coordinator node should forward the request to the other nodes in the WSN.

3.1.3 Uniformity

Travelling sensor nodes must operate in a specific WSN context, which does not necessarily use the same services that are locally installed. By consequence, its services must be interoperable with a wide variety of services on the WSN, which means that independently developed services must be usable in a uniform way.

In addition, upon joining a WSN, travelling sensors must be able to detect which services are currently being used in the WSN, to load extra services if necessary, and to remove services from the node when they are no longer needed or counter productive in combination with those in the WSN.

3.2 Helicopter view

This section first presents the key components in the middleware architecture. After that, it discusses how they support the preferred characteristics presented in Section 3.1, and to what extent the architecture represents a first step towards a complete architecture for sensor applications.

Figure 1 shows three instantiations of the blueprint architecturef for (a) a leaf sensor node, (b) a travelling node, and (c) a coordinator node.

3.2.1 Key components

Service Manager. The architecture offers applications a generic functional API which hides the heterogeneity of service implementations, and a control API to activate and deactivate services. The Service Manager translates a generic request to a particular service component type. If multiple services can handle the request, the Service Selector is called.

Service Selector. The Service Selector selects the most appropriate service implementation to handle an application request. The Service Manager provides the required service type. In case of a travelling sensor, the Service Discovery component may be called to identify and load available service implementations that are used in the local sensor network. By combining information of these two components, the Service Selector is able to decide which service implementation is most appropriate to handle the request.

Policies. The decision process that influences the activities of the Service Manager and the Service Selector has been separated in two policy components. The Service Manager Policy describes how to map an abstract service request to a concrete service type. The Service Selector Policy describes how to select an appropriate service implementation.

3.2.2 Discussion

The architecture described supports the characteristics presented in the previous section as follows: the control API of the Service Selector can be called to adapt the functionality of a node, the middleware hides the heterogeneity of service implementations by offering a uniform functional API, and the behavior of coordination sensors can be customised by injecting policies that map a service request onto the most appropriate component.

In addition, the middleware fits in the approach to address the key challenges presented in Section 2. On the one hand, the architecture enables hor-

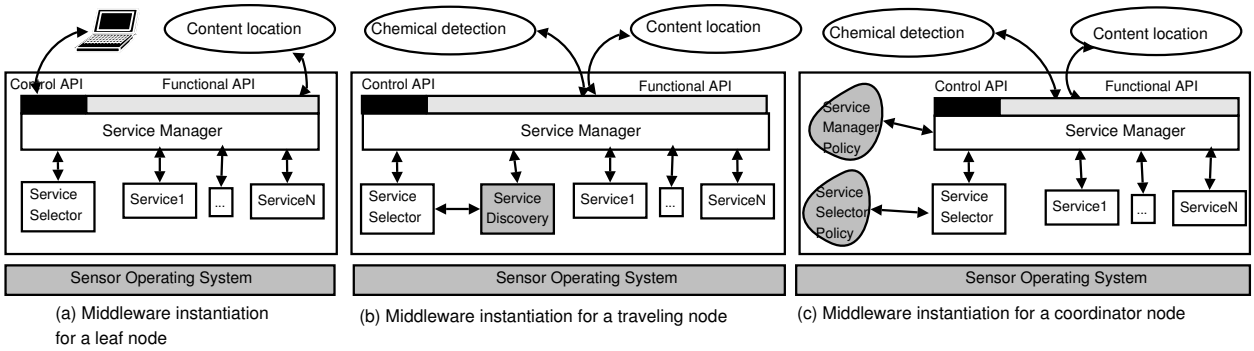

Figure 1: Three instances of the blueprint middleware architecture. Figure (a) shows the architecture for a leaf node that has N services installed (e.g. for routing, time synchronisation, or localisation). Figure (b) shows how the architecture is customised on travelling sensor nodes, by a service discovery component that identifies the services running on the WSN it joins. Figure (c) shows the architecture for a coordinator node. In this case, the system's behavior can be customised by specific policies.

izontal integration support by providing a generic structure that guides optimal component assembly according to the type of sensor on which it will execute. On the other hand, it enables vertical integration by offering policies and a generic API through which system resources can be exploited in the most effective way.

Obviously, the presented architecture shows a high level abstraction of our middleware layer. The next step will be to investigate management and coordination aspects (how to consistently deploy new policies or services in a WSN? how to coordinate simultaneous adaptations in multiple sensor nodes? how to combine business policies of multiple applications that run simultaneously on a WSN?), security (how to authorise different applications that execute on a single WSN?), and safety (how to ensure consistency when dynamically changing a sensor node?).

4 Summary

This paper sketches a research direction that aims for supporting the development of end-to-end applications for sensor networks. We have sketched an application context to illustrate the requirements and to clarify the two key challenges that need to be addressed: horizontal integration of functionality in three different types of sensors, and vertical integration of domain specific policies into a single sensor platform. We have presented a helicopter view on our middleware architecture that application developers can customize.

References

[1] G. S. Blair, G. Coulson, and P. Grace. Research directions in reflective middleware: the lancaster experience. In *Proceedings of the 3rd Workshop on Reflective and Adaptive Middleware (RM2004)*, Toronto, Ontario, Canada, Oct. 2004.

[2] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, Jan/Feb 2004.

[3] K. Roemer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *ACM Mobile Computing and Communication Review*, 6(4):59–61, Oct 2002.

[4] H. Tirri. Challenges of Large-Scale High Power Wireless Sensor Networks, 2005. Keynote talk, The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II), http://www.cs.helsinki.fi/u/tirri/.