Towards an Extensible Agent-based Middleware for Sensor Networks and RFID Systems

Dirk Bade
Distributed Systems and Information Systems
Computer Science Department
University of Hamburg, Germany
Vogt-Koelln-Strasse 30, 22527 Hamburg
bade@informatik.uni-hamburg.de

ABSTRACT

Sensor networks as well as RFID systems are among the hyped technologies nowadays. A lot of research efforts have been spent to develop standards, middlewares and applications. The industry already made large investments to foster the adoption of these technologies, consequently pushing the development, and already deployed the resulting technologies in different domains. However, the addressed technologies are still very young, best practices as well as standards are expected to frequently change, as new demands arise when using the technologies in our everyday life. Because of this, middleware systems are expected to undergo frequent redesigns as well, requiring well suited design paradigms to avoid a software engineering nightmare. We therefore propose an agent-based middleware for sensor networks and RFID systems. This middleware will meet the challenges for having a robust, adaptable and flexible middleware, which is moreover easily extensible to cope with expected re-engineerings and changes while maintaining a clear and elaborate design.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design; D.2.11 [**Software Engineering**]: Architectures; H.4.0 [**Information Systems Applications**]: General

General Terms

Sensor Networks, RFID, Middleware

Keywords

Sensor Networks, RFID, Software Agents, Middleware

1. INTRODUCTION

One of the most important milestones towards reaching Mark Weiser's vision of Ubiquitous Computing is the ability for computing systems to be aware of their environment. For this purpose, computing systems are being augmented with lots of different kinds of sensors to monitor certain states of affairs in their environment. *Wireless Sensor Networks* (WSN) and *Radio Frequency Identification* (RFID) are among the most promising research areas as WSN allow monitoring the physical environment and RFID technology enables the tracking of physical objects therein.

Although a lot of research efforts have been made to promote these technologies, the industry as well as the consumer sector are still in an early stage of adoption. High investments, few standards, and missing killer applications are some of the reasons for dilatory deployment. But the hardware evolves, costs of sensors, tags, readers, etc. rapidly decrease and several alliances and organizations are continuously publishing new standards so that strategic investments as proposed by Gartner [15] may finally become profitable.

Regardless the initial difficulties, several pioneering projects in the area of WSN and RFID systems have already been realized. But until now, most current projects are developed for a specific purpose. They do not interoperate and are neither generically designed to fit other purposes nor do they adhere to existing standards. But once WSN and RFID become widespread (and possibly converge in the future [25]) more experiences with the technologies are gained and the need for well designed infrastructure components will become evident as standards are expected to be frequently revisioned and new standards will arise.

These circumstances are very challenging from a software engineering point of view and we address this issue by proposing a unified middleware infrastructure for WSN and RFID based on software agents. Software agents are autonomous entities often employed for the development of complex and distributed systems [4]. They are capable of sensing their environment, may reactively or proactively act therein and thus adapt to changes and they are able to communicate and cooperate. The paradigm of agent-oriented software engineering therefore allows to build interoperable and reusable software components enabling a robust, flexible and extensible infrastructure [4, 16]. Regarding the unpredictable evolution of WSN and RFID systems, this paradigm is thus ideally suited to cope with the aforementioned challenges.

In the next section we will briefly introduce the basic technologies and highlight their progress in standardization. Afterwards, Section 3 discusses the challenges for engineering future sensor network middlewares in the scope of expected changes and identifies some non-functional design goals to meet these challenges. In Section 4 our proposal for an agent-based middleware obeying these design goals is presented and subsequently discussed in Section 5. Finally, we present some related work in Section 6 and conclude with our prospects of future work in Section 7.

2. FUNDAMENTALS

In this section WSN, RFID, agent and middleware basics are introduced to gain a common understanding of the challenges and concepts described in further sections.

2.1 Wireless Sensor Networks

Wireless Sensor Networks (WSN) are a means for monitoring certain attributes of the physical world (used e.g. in environmental, health and home applications) [2, 24]. Such networks consist of a multitude of autonomous nodes, each equipped with sensors, a processing unit and communication capabilities. Once the nodes are deployed in a certain region they start to sense their environment and build up a kind of ad-hoc network with their neighboring nodes. In most WSN one or more base stations can be found, to which the percepts of each node are transmitted using multi-hop routing. For this purpose the nodes cooperate with each other by forwarding percepts of other nodes. Due to the limited resources of the nodes by means of energy as well as processing and communication capabilities, research in the area of WSN mostly concentrates on how to efficiently manage and distribute the information [25].

2.2 RFID

Although RFID systems also aim at monitoring the physical world, their primary use is the identification and tracking of real-world objects [22]. For this purpose objects, e.g. assets, are required to have a unique digital identity. This is provided by tags, which are comparable to a barcode, but may be read by specialized readers without a line of sight and in bulk over a distance ranging from several centimeters to a hundred meter (depending on the tag). The identity stored on a tag is often referred to as an Electronic Product Code (EPC) and can be used to link the identity with further information about the object stored somewhere in a network. Several application domains already make use of RFID technology, e.g. manufacturing control, asset tracking, warehouse and fleet management [24, 21] and concepts for several other domains are already being developed.

RFID systems are similar to WSN in the sense that data is read by specialized sensors (i.e. RFID reader) and can also be written back to tags in some cases. Hence, we also have streams of raw data which need to be processed and transformed to higher level events. And indeed, it is expected that RFID and WSN technologies will further converge in the future [25].

2.3 Software Agents

There is no definition of software agents in literature that is generally agreed upon. A basic definition states that software agents are able to perceive their environment through sensors and act upon it through effectors. As this definition is applicable to a multitude of software components one would not necessarily call an agent, other definitions specify certain characteristics an agent must have. Regarding these, an agent must be autonomous meaning the ability to process a task with as few guidance by its principal as possible. Moreover, an agent should be able to react to changes in its environment, but also to proactively follow its design goals. Additionally, agents must have the capability to cooperate by means of exchanging messages and must be able to adapt their behavior according to changes in the environment [16].

These definitions and characteristics lead to a very abstract view of what an agent actually is. From a software engineering perspective, agents are similar to objects, but a little bit more abstract. They can be seen as software components, developed to exhibit the above mentioned characteristics. And these agents are normally executed on a special middleware, called agent platform, which manages the lifecycle of agents and offers additional infrastructure services like a message transport system and a directory service. We will further enhance this brief introduction in the subsequent sections, once the context allows an explanation by example.

2.4 Middleware

Middleware in general shall shield the application layer from the details of lower layers in a way that applications can transparently use different shapes of services without the need to know any implementation details. Any changes in the lower layers hence do not affect the applications, but only the middleware. In most cases a middleware also provides a set of additional commonly used services for a specific purpose, so that applications do not need to implement the necessary functionality themselves.

Different middleware architectures have already been proposed for WSN as well as for RFID systems and even some for a combination of both technologies [5, 10, 14, 17, 22, 25, 26]. Regarding WSN, the term middleware often refers to a software layer residing between the application layer and the lower level hardware-oriented layers of sensors. It's main purpose is to support the development, maintenance, deployment and execution of sensing-based applications [24, 21], particularly focusing on power and topology management, data aggregation, transmission protocols, etc. inside a sensor network [1]. But a holistic view on WSN and traditional networks is often not provided, i.e. the connection to infrastructure networks is hardly considered by the middleware [21, 25, 29]. It is important to point out that our proposal coexists with current WSN middlewares as we are focusing on the post-processing of sensor data beyond the sensor network border.

In contrast to WSN, RFID middleware concentrates on efficiently processing and constructing meaningful events. In this case, the term 'meaningful' plays an important role, as the focus of an RFID middleware is not only on distributing data to a specific base station (as in WSN), but on processing and enriching data with contextual information on its way up the processing hierarchy. RFID tags themselves are mostly not capable of processing the data stored on the tag. Instead they are given just enough resources to communicate with a reader [22]. As a consequence, readers are the bottom-most layer of common RFID middleware and simply push data the RFID stack upwards, where the data is filtered, aggregated, translated, enriched, etc. before meaningful *Application Level Events* (ALE) can finally be sent to applications for further processing [22].

2.5 Standards

Standardization issues play a major role in the adoption of technologies. Software and hardware developers want to be sure that their work gains acceptance by customers and will not be outdated within the near future. Also the customers need to feel confident that their investments in a technology are future-proof and are globally used in order to facilitate the cross-enterprise exchange of information. And finally, standards are the basis for competitive marketplaces where different system components may be traded, and consequently interoperability needs to be assured [8].

WSN is not widely deployed yet and one does not know whether the reasons are a lack of interest by the industry or a lack of standards. There are few standardization efforts for WSN and these are mostly concentrating on processing and communication mechanisms inside the network [1, 25, 29] (e.g. IEEE 1451, ZigBee). Existing standards often rely on standards borrowed from other areas and just add minor changes to adopt them to the special WSN characteristics. But to the best of our knowledge there are no standards for the interface between the data acquisition network (the sensor network itself) and the data distribution network (the backend responsible for post-processing the data), which is in our context the most interesting part. As a result, merging the data of different sensor networks in higher hierarchy levels of processing often relies on proprietary solutions [25].

Regarding RFID technology standardization has made a good progress. Driven by large interest and large investments multiple standards arose during the last years. Besides the *International Organization for Standardization* (ISO) also *EPCglobal*, a consortium of several companies and universities, is engaged in the process. EPCglobal published several standards for data representation and interfaces, among which the *Architecture Framework* [8] is in our context the most important one as it specifies a set of interfaces and roles within an RFID middleware. The main goal of these tasks is to gather, filter, enrich and transform raw sensor data in a way that application level events (ALE) can be forwarded to interested parties. But as this abstract architecture framework does not specify a real system architecture, several concrete architectures taking the infrastructure roles into account have been proposed [5, 10, 22, 26].

Implementing a concrete architecture, for WSN as well as for RFID systems, can be quite challenging, because both technologies are quite young and in some aspects still in an early stage of development. And it gets even more challenging when considering a unified middleware for both WSN and RFID [25]. In the following section we will therefore outline some of the difficulties in realizing such a middleware and list some requirements for future engineering of middlewares for sensor networks in general.

3. ENGINEERING CHALLENGES

Regardless the unpredictable future of RFID and sensor networks, universities as well as several companies are developing infrastructures and applications taking the already existing standards of the according technologies into account and using proprietary solutions if necessary.

Although a lot of standards have already been published for such infrastructures, there are still problems adhering to them. Reasons for this are threefold: First, there exist different standards for different infrastructures. This seems conclusive, but sensor networks in general (including RFID systems) have a least common denominator (e.g. post-processing of percept data) [25], which is not accounted for by the standards. Second, standards for specific aspects of an infrastructure are missing due to the lack of appropriate use cases [8]. As most technologies in these areas are quite young and rarely used, there are few experiences and hence new standards are not proposed until the requirements are fixed. Third, WSN and RFID technologies may further converge in the future [25] and standards will have to be redeveloped or merged in order not to get lost in standardization.

For these reasons, we expect the standards to be subject of frequent changes within the next years. Hence, the development of an infrastructure adhering to the standards (and possibly being compatible to the 'old' ones) may become a software engineering nightmare, because the process of software development needs to be iterated over and over again as new demands and standards arise. These development cycles require a flexible and extensible software architecture, otherwise substantial redesigns will become inevitable. In the following we will thus discuss a set of non-functional design goals for future sensor network middlewares, which take the above mentioned challenges into account.

3.1 Design Goals

One has to distinguish between functional and non-functional design goals. While functional goals define 'what' a system shall do, non-functional goals specify 'how' a system is supposed to be (i.e. quality goals). In order to be prepared for future sensor network developments, we identified some non-functional design goals, specifying evolution qualities, that are of special importance for new generation system architectures (an overview of functional goals can be found e.g. in [1, 10, 22, 23, 26]). Most of these should be obvious and sensor middleware should naturally adhere to them, but in practice this is often not the case.

Robustness and Adaptivity A middleware for sensor networks will need to be robust not in the sense that only the data acquisition network but also the backend, the data distribution network, needs to be tolerant towards failures. This requirement is accompanied with the need to be able to adapt to changing conditions, especially if different participators account for specific services in a network. Therefore a loose coupling of components and the possibility to dynamically choose an appropriate service at runtime is necessary.

Flexibility Someday, handling sensor data will not only be a matter of companies with global-scale processing networks, but will also be managed by individuals within local networks. A middleware must thus be flexible in a way that it must be deployable in different scales, i.e. certain functions an individual does not necessarily needs may be omitted for the sake of simplicity and more sophisticated functions as required by enterprises must be easy to integrate. From a software engineering perspective, the functions must also be easily exchangeable as requirements and standards change.

Scalability More and more assets will be equipped with RFID tags, more sensor networks be deployed and eventually the data from all of these be joined in global-scale networks. Therefore, a middleware needs to be able to process single percepts as well as thousands or millions percepts.

Extensibility If sensor networks and RFID systems become widely deployed, new use cases accompanied by new requirements will arise. A middleware architecture must thus be extensible and the extensions should not necessarily require the applications built upon that middleware to change, but instead new applications should be able to directly use the extensions.

Having the above mentioned goals in mind, we propose an agent-based system architecture for WSN and RFID systems. In the following we will present the architecture and afterwards discuss our approach with respect to these goals.

4. AGENT-BASED MIDDLEWARE

To face the above mentioned challenges a flexible and adaptable architecture with loosely coupled components is required. Therefore, we propose a middleware infrastructure based on software agents for processing event streams originating from different kinds of sources (e.g. WSN, RFID reader or any other source). To ease understanding we will first present a motivating example and refer to that example in the subsequent sections.

4.1 Motivating Example

A trading company expects to receive a pallet with TV devices. The pallet is shipped within a smart container, equipped with several sensors measuring and logging acceleration, humidity and temperature throughout the whole transport. Once the container is received by the local warehouse, the pallet shall be unpacked and the devices be loaded directly onto a truck to deliver them to the customers, but only in case the sensor values logged during the transport do not exceed a specific limit. In this case, the trading company needs to send a mechanic into the warehouse to check if the devices are damaged.

To be informed about the state of delivery, the trading company registers several event triggers with the warehouses sensor network middleware. States of interest are: a) The pallet does not arrive in time b) the pallet arrived, but sensor values imply a possible damage and c) the pallet arrived in time and devices are going to be loaded onto the truck.

4.2 Overview

Inspired by common event driven architectures [7] and existing RFID systems [5, 10, 22, 26] we also follow a three-layered system architecture comprising the *Application Layer*, an intermediate *Network Layer* and finally the *Network Edge*. The latter connects to the event sources (e.g. WSN base station, RFID reader, etc.) and is responsible for infrastructure management as well as low-level event filtering. The Network Layer's primary role is the higher-level processing of events and the creation of *Application Level Events* (ALE) as required by the Application Layer. In our architecture the Application Layer requires no mandatory base components, hence even resource-constraint devices may make use of the subordinated Network Layer's functionality. Moreover, as our middleware shall be independent of specific kinds of event sources, the sources are not part of the architecture, but one can imagine the sources to reside in an additional layer below the Network Edge (as found in [5, 22, 28]). A detailed introduction of each layer is given in the following.

4.3 Application Layer

The Application Layer (also referred to as the Network Core, cp. [22]) represents the highest layer in the middleware stack. This layer only offers additional services for monitoring and debugging functionalities. It hence contains no mandatory components as to even allow applications running on resource-constrained devices (e.g. mobile phones) to be part of the infrastructure. This layer can be seen as a logical entity as it is not a core in physical means, but may be distributed over several physical locations. It is even possible to have multiple cores, e.g. each organization has its own core, but all of them are operating on the same middleware layers below. In fact, the relationship between Application Layer and Network Layer is an n:m relationship, as it is also possible to have one Application Layer, operating on multiple Network Layers.

Fig. 1 depicts the components residing in this layer as well as the actions a developer has to accomplish in order to connect the Application Layer respectively an application with the Network Layer. The overall goal is to receive meaningful Application Level Events (ALE) from the Network Layer once one or more specific low-level events occur. For this purpose a developer first has to create an event filter for filtering out events of interest. As the event filter is processed by a complex event processing engine (e.g. *Esper* [9]) in the Network Layer, also complex patterns (allowing content-based filtering in contrast to current standards [25]) as well as causal and temporal relationships may be detected in a stream of events [18].

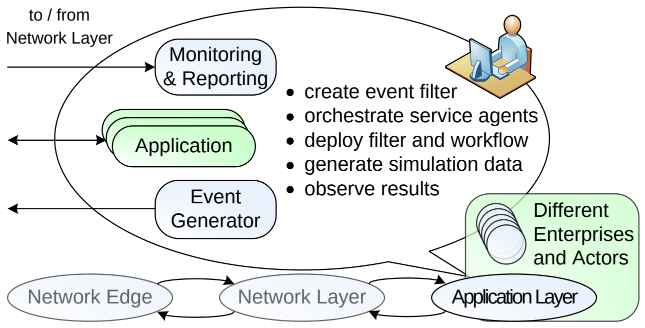

Figure 1: Application Layer

In a second step, the developer may choose how an event passing the filter shall be processed by the lower layers. In RFID systems for example, events may be aggregated, grouped, translated and enriched with additional context information. For this purpose, the Network Layer offers a yellow page service which may be queried by the developer to find and orchestrate the service agents she needs in order to create a tailored ALE. Such an orchestration is similar to a business process orchestration, in which activities, their processing endpoints and additional parameters may be specified. For example, one may create such a workflow by specifying that events shall be first translated from representation A into representation B and then be enriched by querying a certain context service (e.g. an EPC information service). In case the service agents registered with the yellow page service do not match the needs, a developer may also choose to provide its own service agent for a dedicated task, for example to translate an event into the company's proprietary representation.

After creating the workflow (representing the service agent orchestration) and the corresponding event filter, these need to be registered at the *High-Level Event Filter* (cf. next section) of the Network Layer. As the Network Layer may already be a productive system, testing and debugging the filter as well as the workflow without interfering others may be a problem. For this reason, developers may use the *Event Generator* as well as the *Monitoring & Reporting* components, to create suitable simulation data and monitor the service agents executing their tasks.

In our example scenario the trading company wants to be informed once one of three possible delivery states is achieved. Therefore, three different event filters need to be created. Additionally, at least one workflow description has to be specified, instructing the middleware how the events shall be processed. This way, the trading company may provide e.g. its own aggregation function to be executed on sensor data, specify to call specific external services (e.g. call a mechanic to check the devices) to be executed depending on the aggregated values and enrich an event with additional context data helping the mechanic to bring the right tools. Once event filters and the workflow have been deployed, the company may simulate different scenarios to assure the right operations are performed by using the Event Generator and monitoring the processing of generated events.

4.4 Network Layer

The Network Layer is responsible for mapping low-level events received from the Network Edge to Application Level Events (ALE) that are finally forwarded to the Application Layer. Several tasks may be performed at this stage (cf. [22]):

- Events may be aggregated (e.g. just counted), grouped (e.g. build event sets with respect to a specified attribute) or translated (e.g. change content encoding).
- Additional information based on the event source or a specific event attribute (e.g. EPC) may be retrieved from external sources, e.g. an *Object Naming Service* (ONS) and an *EPC Information Service* (EPCIS) [8]. This task may involve multiple service invocations (e.g. ID resolution, context retrieval, ontology lookup, etc.).
- Every event source may allow to propagate data coming from applications towards a specific sensor. This way applications are able e.g. to write into the memory of an RFID tag or to send instructions to a WSN.
- An application may also specify that an external agent has to be called once a specific condition occurs. This way the processing chain may be easily extended to include arbitrary services.

As already mentioned in Section 3, these tasks may be extended in future middleware generations. Additionally, the interfaces, as standardized by e.g. EPCglobal, may be changed according to further demands [8]. To deal with these circumstances, we propose to encapsulate every task in a dedicated agent type (see Fig. 2). Once an interface changes or new roles are introduced, the corresponding agent types simply need to be refactored or newly created. For backwards compatibility one may decide to additionally keep the 'old' agent working. In order to execute the activities of a workflow (sequentially or in parallel) agents coordinate among each other by exchanging messages in a standardized communication manner.

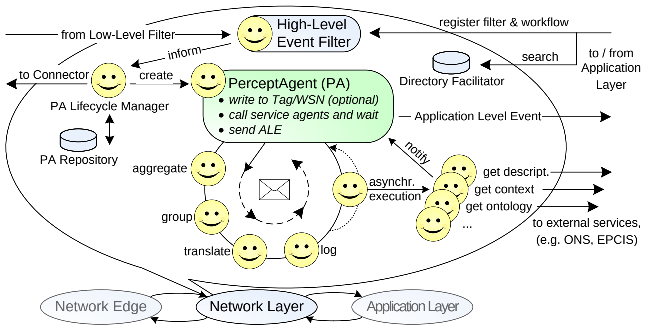

Figure 2: Network Layer

For an application to be informed about specific events, it has to subscribe at the *High-Level Event Filter* (HLEF) providing a (possibly complex) event pattern as well as a workflow description, containing orchestration instructions for the above mentioned service agents (cf. Section 4.3). Events coming from the Network Edge (cf. Section 4.5) are processed by a complex event processing engine in the HLEF and filtered with respect to the registered high-level filters. Once a low-level event passes the filter (meaning an application subscribed for that event) a so called *PerceptAgent* (PA) is instantiated and further on responsible for coordinating the processing of the workflow corresponding to that event. Note, that for a single low-level event, multiple PAs may be instantiated (one for each registered high-level filter). Processing the workflow means sending a message containing event information to a service agent as specified in the workflow and wait for an answer. Processing is finished once either all involved service agents notified the PA of completion or if the event is subject for being aggregated or grouped. In this case, superordinated dedicated *Aggregation-* and *GroupAgents* (not depicted) take over the responsibility for the event.

If a service agent specified in the workflow does not accept a task or does not respond within a specified time interval, the PA may decide to ask the yellow page service (called *Directory Facilitator*) in order to find another agent instance capable of execution. This way dynamic binding and hence adaptation to network failures may be achieved. Once a workflow is completed, an ALE is created and finally sent to the Application Layer for further processing.

Data may not only be read from lower layers, WSN as well as RFID systems also allow to write data. For example, an application may want to send processing instructions, queries, or a new power management configuration to a node inside a WSN or data, e.g. values coming from a sensor network, shall be stored into the memory of an RFID tag. In the case of writing data, two problems arise: 1) how to address a single sensor or tag from an application and 2) what happens if a sensor or tag is currently not in range so that writing temporarily fails? Regarding the first problem, an application shall never address entities in the lowest layer directly, as details of the concrete addressing scheme must be known by the application. Therefore, the middleware needs to abstract from low-level addressing details and must provide a mapping between an abstract and a concrete addressing. In our architecture this is achieved by the PAs and the *PA LifecycleManager* (PALM). An application may send messages to the PALM addressing a PA using a standardized scheme. The PALM in turn creates a new PA instance, which caches the data as long as the corresponding sensor or tag is sensed again, implying that it is also in writing distance and then dispatches the writing request to an appropriate *Connector* (cf. Section 4.5) in the Network Edge, responsible for writing to a tag or sensor. This way, also the second problem stated above may be solved. Similar approaches also exist in other projects, but the involved component roles as well as the naming are slightly different (cf. *virtual counterpart* [20], *virtual sensor* [1], *virtual tag* [10]).

Coming back to our example scenario: as already described, the trading company needs to register event filters and a workflow description with the HLEF. Once the container is being unloaded, the sensor log-files are read and corresponding low-level events from the Network Edge are forwarded to the HLEF. The complex event processing engine, when trying to match a registered filter against a stream of events, caches the relevant sensor data by itself, so that additionally storing the data in the warehouse is not necessary (although reasonable). After reading the sensor logs, an RFID reader scans all tags that are attached to the TV devices and subsequently generates appropriate events, which are again forwarded to the HLEF. Depending on the sensor data either the trading companies event filter for possibly damaged devices or the event filter indicating that everything is fine matches (the third event filter triggers after a specific time interval only if no corresponding TV devices are sensed). As a consequence, the PALM is informed, looking into its repository if corresponding PAs (with pending write instructions) exist, and finally instantiates a (new) PA. The PAs are provided with the trading company's workflow and coordinate the execution of activities by sending messages to the corresponding service agents. After being notified that all agents completed their work, data necessary for constructing an ALE is gathered and the ALE is finally forwarded to the trading company's application.

4.5 Network Edge

The Network Edge (depicted in Fig. 3), as the name indicates, separates the non-IP from the IP segment of the system. Event sources transmit event streams or batches to protocol-specific *Connectors* and further on to a *Low Level Event Filter* (LLEF) before they are finally forwarded to the Network Layer's HLEF.

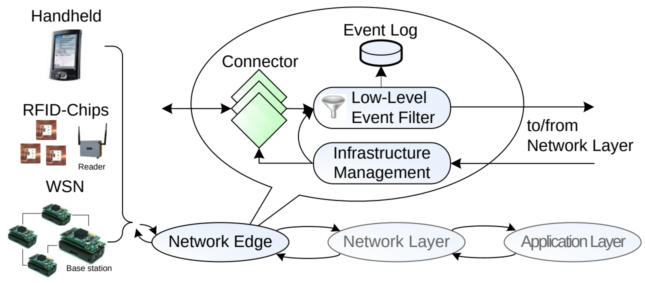

Figure 3: Network Edge

The Connectors are responsible for bridging the protocol gap, as the middleware components communicate over standard Internet protocols, but this must not necessarily hold for sensor networks, RFID readers or other event sources. In the case of RFID for example, low-level protocols for the communication between tags and readers have been specified by EPCglobal, but for the communication between readers and the Network Edge only data formats are standardized. Hence in practice readers communicate using a multitude of interfaces, e.g. Bluetooth, (W)LAN, IrDA or serial RS-232. As a consequence, one or more Connectors (depending on the provided data format) for each technology are required.

Once a Connector receives an event from an event source, contents are extracted and passed further on to the LLEF. This filter corresponds to an event filter as found e.g. in RFID systems and is used to filter out duplicates and incomplete, malformed or unknown events (but it has to be pointed out that the LLEF is no substitute for the filter integrated into RFID reading devices, as these work on an even lower layer). Events passing this stage are handed over to the HLEF residing in the Network Layer (cf. Section 4.4). An *Infrastructure Management* component is used to configure sensors as well as the event filter. At this layer we do not necessarily employ agents as the components effectively processing events in this stage are already in use by several other projects (e.g. *Fosstrack* [3]) and may simply be reused. Although it is possible to additionally wrap the functionality by agents to achieve a coherent addressing.

In our example scenario, the data from the container's sensor logs are read by a specific base station and passed on to a Connector. This Connector in turn forwards the data to the LLEF. Depending on the configuration of the filter, some sensor information may be discarded as it is of no interest. The remaining data is further transmitted to the HLEF. Once the container is unpacked, the TV devices are read by an RFID reader and the information is again passed on to the Connector and further up the stack to be finally processed by the HLEF.

5. DISCUSSION

In Section 3 we discussed challenges for future middleware systems and identified some non-functional design goals. In this section we now want to highlight how our proposed system architecture meets these design goals and where possible problems may arise.

We followed the paradigm of agent-oriented software engineering, which is well suited and approved for developing complex software systems in distributed and dynamic environments [4]. Among the reasons are that agents are autonomous, which means they are able to decide for themselves what they want to achieve, and they are capable of sensing their environment and hence able to adapt their behavior to changing conditions [16]. If for example an agent called an external service, but does not get any answer, it may decide for itself to follow different behavioral strategies, e.g. wait, call service once again or look for another service. In our architecture, *robustness* is achieved by using a yellow-page service for finding required service agents hence the failure of one agent can be compensated by requesting another instance at runtime and moreover by service agents being stateless (a new instance for every event-subscription pair is created) failures do not necessarily affect subsequent processings. *Adaptation* to network and load changes may be achieved by dynamically choosing processing nodes on which appropriate service agents reside or by migrating mobile service agents onto these nodes.

Moreover, agents are able to communicate and cooperate, which allows to divide a complex task into multiple simple tasks being executed by individual agents (cf. our service agents in the Network Layer). Communication is done by exchanging messages, which may be processed asynchronously, therefore tasks may easily be executed in parallel. Standardized infrastructure components for agent platforms, e.g. a yellow page service, may be used to achieve loose coupling as an agent may decide at runtime with whom to communicate based on its current senses. This also fosters the possibility to easily extend our middleware by simply introducing new agents, replacing or cooperating with existing ones. Additionally, agent-oriented programming is even more abstract than e.g. the object-orientation, allowing developers to focus on functionality instead of dealing with low-level details like communication and threading issues for example. All these arguments argue for agent-oriented software engineering achieving the *Flexibility & Extensibility* design goals.

Of course performance issues have to be discussed when processing thousands or millions of events [5]. And by using agents and message-based communication an additional overhead has to be considered. But in general agent-based software scales very well as the agents may be easily distributed among several hosts and by using yellow page services loose coupling and hence a dynamic binding can be achieved. Our middleware is currently implemented using the Jadex V2 agent system [19] which ships with a high-performance agent platform and is capable of executing very lightweight micro agents. Additionally, Jadex also allows BDI (Belief-Desire-Intention) agents to be executed, which may be of interest for designing more complex agents with reasoning capabilities for special tasks (e.g. intelligent adaptive routing). For these reasons we argue that agent-based applications naturally scale very well thus achieving the *Scalability* design goal.

6. RELATED WORK

In the past, different middleware platforms have been proposed for RFID systems, sensor networks and combinations of both. The EPCglobal consortium has published several standards for the processing of RFID data, including the *EPCglobal Architecture Framework* [8]. As this framework only proposes abstract standards, several projects aim at building concrete system architectures adhering to these standards, among these are for example [10, 20, 22, 26].

But the examples lack a flexible design making an adoption of the overall architecture to new demands and standards quite laborious. Moreover, they concentrate on RFID data only and in most cases do not allow multiple applications and organizations to take part in the event processing. To the best of our knowledge, few projects use agent technology as part of the system architecture. For example [27] are using software agents for a manufacturing control system. They embed the functionality of an RFID middleware to a large extent into a single monolithic agent, which is directly interfaced by applications. Another middleware approach using agent technology is [6], which focuses on mobile agents for gaining load balancing in RFID systems.

Although standardization of WSN technology has not made substantial progress, several works propose middleware architectures or guidelines and design issues for the development of such architectures [14, 17, 21, 29, 30]. Some works even make use of software agents [11, 13]. But all approaches mostly concentrate on the ongoings within the sensor network (e.g. data aggregation, routing, etc.) and do not account for the post-processing of sensor data in the backend, which we are focusing on. Sung et al. predict a convergence of RFID and WSN technologies in the future as RFID tags will become more powerful and hence gain the ability of autonomous communication and processing [25]. Additionally, isolated systems may be interconnected in the course of time realizing the vision of the *Real World Web* [15]. Therefore, some research efforts have already been spent in developing an integrated middleware for both WSN and RFID systems [12, 17, 25, 28] as well as for global scale systems (e.g. [1]). As a consequence, a middleware needs to be flexible and generic to abstract from incoming concrete events and outgoing sensor instructions on the one hand, and also should be extensible to easily allow incorporation of new functionality in the future as new demands arise. In our opinion and to the best of our knowledge, none of the systems satisfies these requirements.

All architectural proposals in common lack a future-proof design, rely on centralized infrastructures and/or make use of monolithic building blocks. They mostly concentrate on the state-of-the-art in standardization and apply proprietary solutions if necessary. But as new standards will arise and existing ones be changed, system architectures are confronted with frequent and substantial redesigns and refactorings, putting the architectures to the test.

7. CONCLUSION AND FUTURE WORK

In this paper we argued that the design of future middleware architectures for sensor networks and RFID systems is challenging due to the underlying standards being subject to frequent changes. As a consequence we identified a set of non-functional design goals, which should be considered when developing such middlewares.

By using software agents for engineering the middleware we expect to be able to deal with frequent architectural changes as agents are a means for designing complex software in dynamic and distributed environments. Additionally, agents are from a software engineering perspective the natural answer to scalability, reliability, extensibility and adaptability issues - key concerns for the distributed processing of event streams.

We proposed a three-layered, event-driven architecture following the state of the art middleware designs, in which agents are the main actors responsible for processing the events. From an application's point of view, the processing of events is in this case simply an orchestration of service agents, which collaborate in order to create tailored Application Level Events for the application.

Our prospects for future work are basically finishing the implementation of our proposed architecture to prove its feasibility. Within that scope we will develop exemplary applications of different scale addressing certain aspects of the middleware. Looking further into the future, standardization progresses as well as the evolution of existing middlewares will be closely followed in order to adapt our architecture to changing conditions and demands.

8. REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Intern. Conf. on Mobile Data Management*, pages 198–205, 2007.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38:393–422, 2002.

[3] AutoIDLabs. Fosstrak. www.fosstrak.org, 2009.

[4] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.

[5] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing rfid data. In *Proc. of the 13. Intern. Conf. on Very Large Data Bases*, pages 1189–1195. VLDB Endowment, 2004.

[6] J. F. Cui and H. S. Chae. Mobile agent based load balancing for rfid middlewares. *Advanced Communication Technology, The 9th International Conference on*, 2:973–978, Feb. 2007.

[7] J. Dunkel, A. Eberhart, S. Fischer, C. Kleiner, and A. Koschel. *Systemarchitekturen fuer verteilte Anwendungen*. Hanser Fachbuch, September 2008.

[8] EPCglobal. The epcglobal architecture framework, final version 1.2. Technical report, Sept. 2007.

[9] Esper. Event stream intelligence. esper.codehaus.org.

[10] C. Floerkemeier and M. Lampe. Rfid middleware design: addressing application requirements and rfid constraints. In *Proc of the conf. on Smart objects and ambient intelligence*, pages 219–224, USA, 2005. ACM.

[11] C.-L. Fok, G.-C. Roman, and C. Lu. Agilla: A mobile agent middleware for sensor networks. Technical Report WUCSE-2006-16, Washington University in St. Louis Dept. of Comp.Sc.and Eng., 2006.

[12] M. J. Franklin and et al. Design considerations for high fan-in systems: The hifi approach. In *CIDR*, pages 290–304, 2005.

[13] D. Georgoulas and K. Blow. Intelligent mobile agent middleware for wireless sensor networks: A real time application case study. *Advanced International Conference on Telecommunications*, 0:95–100, 2008.

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

[15] G. Inc. Gartner's 2006 emerging technologies hype cycle. www.gartner.com/it/page.jsp?id=495475.

[16] N. R. Jennings and M. Wooldridge. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[17] T. S. Lopez and D. Kim. Wireless sensor networks and rfid integration for context aware services. Technical report, Auto-ID Labs, 2008.

[18] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, May 2002.

[19] A. Pokahr and L. Braubach. From a research to an industrial-strength agent platform: Jadex v2. In to appear, editor, *9. Internationale Tagung Wirtschaftsinformatik*, 2009.

[20] K. Römer, F. Mattern, T. Dübendorfer, and J. Senn. Infrastructure for virtual counterparts of real world objects. Technical report, ETH Zurich, 2002.

[21] K. Roemer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):59–61, 2002.

[22] G. Roussos. *Networked RFID: Systems, Software and Services*. Springer Publishing Company, Inc., 2008.

[23] M. Sgroi and et al. A service-based universal application interface for ad hoc wireless sensor and actuator networks. In *Ambient intelligence*. Springer Verlag, Berlin, 2005.

[24] K. Sohraby, D. Minoli, and T. Znati. *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley-Interscience, 2007.

[25] J. Sung, T. S. Lopez, and D. Kim. The epc sensor network for rfid and wsn integration infrastructure. In *Proc. of the 5. IEEE Intern. Conf. on Perv. Comp. and Comm. Workshops*, pages 618–621. IEEE, 2007.

[26] F. Thiesse. Architektur und integration von rfid-systemen. In E. Fleisch and F. Mattern, editors, *Das Internet der Dinge*, pages 101–117. Springer, '05.

[27] P. Vrba, F. Macrek, and V. Mařík. Using radio frequency identification in agent-based control systems for industrial applications. *Eng. Appl. Artif. Intell.*, 21(3):331–342, 2008.

[28] W. Wang, J. Sung, and D. Kim. Complex event processing in epc sensor network middleware for both rfid and wsn. In *Proc. of the 11th IEEE Symp. on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 165–169. IEEE, 2008.

[29] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *Network, IEEE*, 18(1):15–21, 2004.

[30] L. Zhang and Z. Wang. Integration of rfid into wireless sensor networks: Architectures, opportunities and challenging problems. *Grid and Cooperative Computing Workshop*, pages 463–469, 2006.