# Architecture for Collaborative Business Items

Till Riedel[1], Christian Decker[1], Phillip Scholl[1], Albert Krohn[1], and Michael Beigl[2]

[1] TecO, Universität Karlsruhe (TH), Vincenz Prießnitz Str. 3, 76131 Karlsruhe, Germany
{riedel,cdecker,scholl,krohn}@teco.edu
[2] IBR, Universität Braunschweig, Mühlenpfordtstraße 23, 38106 Braunschweig, Germany
beigl@ibr.cs.tu-bs.de

**Abstract.** Sensor network technology is pushing towards integration into the business world. By using sensor node hardware to augment real life business items it is possible to capture the world and support processes where they actually happen. Many problems of the business logic running our world can be efficiently implemented "on the item". In order for these smart items to couple back to the virtualized world of business processes it necessary to design a uniform system abstraction for enterprise systems. Service oriented architectures are the tool to describe functionality apart from its concrete implementation. This paper describes a system and the experiences made integrating wirelessly networked smart business items into high-level business processes.

**Keywords:** Wireless sensor networks, service oriented architecture, business logic, distributed systems.

## 1 Introduction

The complex and interwoven business logic demands more and more information sources to enable reliable, flexible and efficient processes. Information has become one of the key values of today's business world. At the same time technologies emerge that can lead to a ubiquity of information sources. Especially sensor nodes are a promising platform for enabling the digitalization of our environment. Augmenting real life business items like goods or people with tiny wirelessly connected computing and sensing platforms creates a broad range of possibilities.

However, the enormous amount of unfiltered information arising from a steadily growing amount of sensors can soon become a problem of scalability. The need for continuous evaluation of sometimes-unreliable information often contradicts the goal of reliable, flexible and efficient business processes. We think that only *integrating sensor networks into business logic* (as in [2]) will thus fall short of a scalable solution. However, *integrating business logic into sensor networks* we believe can become a future direction of computing systems.

The information processing capabilities of sensor networks can implement business logic in a collaborative fashion without pushing unnecessary information to backend systems.

P. Lukowicz, L. Thiele, and G. Tröster (Eds.): ARCS 2007, LNCS 4415, pp. 142 – 156, 2007.


**Fig. 1.** Smart chemical drums equipped with sensor nodes

The following motivating scenario, for which we build a trial installation in a BP chemical plant in Hull, UK, shall outline sensor network capabilities in the context of business processes. In this example containers of chemical goods – the business items – are equipped with wireless sensor nodes (see Fig. 1) in order to enforce pre-defined storage conditions. The regulations are encoded in backend enterprises systems as part of the workflow with the business items. However, the supervision on the regulation is delegated to the business items themselves. In this scenario, the nodes on the items communicate with each other and collaboratively supervise the environment around the containers. Business logic on the nodes utilizes the sensor information on presence on number of other containers, chemical content, and environmental conditions, e.g. temperature, as input to collaboratively reason on eventual violation of storage regulations. If detected, the nodes in this scenario are able to warn locally by triggering a visual alert signal and report this incident back for purposes of logging to a backend system. In contrast to technologies like Radio Frequency Identification (RFID), the process on detection and enforcement is completely distributed among the participating business items. This enables a continuous, very accurate in-situ supervision with no additional information overhead for back-end systems.

Sensor networks already have the means for executing such business logic (e.g. in [1]). However, in order to ensure that backend end sensor network seamlessly work together a technical framework is needed that provides interconnectivity between sensor networks and server based enterprise systems. Both systems have optimized ways of processing and communicating data. All assumptions about increasing efficiency by combining both only hold if we don't lose their original efficiency along the path of integrating both systems. Based on the implementation of the business logic of the smart drum scenario we show how efficient an unwanted and potentially dangerous storage combination can be identified on item level and how this can increase the reliability of systems where logic is coupled with physical entities.

In this paper we describe a service-oriented architecture for seamless integrating business logic executed on sensor networks. The goal of the architecture is to delegate parts of the business logic from resource intensive backend systems to thin sensor node technology. In the next section we analyze the general structure of business logic and the requirements for mapping them to sensor networks. Furthermore we discuss the technical and structural challenges for an integrated architecture. We continue describing a pragmatic architecture for enabling sensor supported business logic, the CoBIs Gateway Architecture. It links services executing logic on top of the sensor network through UPnP (Universal Plug and Play) proxies to the logic running in backend systems. In the last section we present our experiences gathered while applying this technology within a real world trial.

## 2 Analysis

A business process describes the transformation of resources in order to achieve a measurable business relevant outcome. Business processes can be split into process tasks, which can be delivered by different service providers. The sequence of such tasks can e captured and modeled in so-called business logic.

The term *business logic* is used to distinguish the processing part from presentation and storage within classical 3-tier architectures. The term "functional process logic" might be a better term describing the same thing. Classical business logic is built on top storage layer most commonly called CRUD (Create, Read, Update and Delete). Because business logic components still share a lot of intimate knowledge about the data layout CRUD layer parts of the reusability of business logic is often limited. Services oriented architectures have recently been used to abstract from this rather tight coupling to the data layer by defining functional interfaces across all components.

In our view (see [3]) a service consists of a well-defined functionality and a well-defined interface for accessing this functionality from a client. To distinguish the service abstraction from others we demand that the client is independent of the concrete implementation of the service and that the each service is independent from the internals and the state of any other service. The discriminating factor of service-oriented architectures is the loose coupling between services. This enables exchanging functional entities and gives us the possibility to seamlessly integrate new technology.

### 2.1 Collaborative Business Items

Business logic acts on a virtual representation of physical world. Every business item that is part of a process is modeled to have an equivalent in the virtual world. Business rules and workflows describe how to act on those objects. Business logic often relies on user or machine interfaces to update their state according to the real world process in parallel. The state between the real world and the virtual world is thus only synchronized at certain checkpoints in space and time.

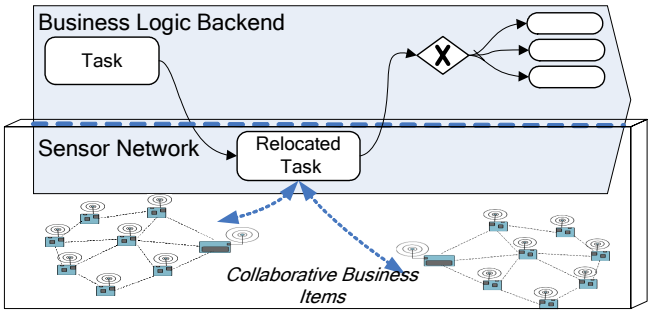


**Fig. 2.** Relocated business logic

Collaborative Business Items (CoBIs) push the interface between real world processes and business logic out onto the business item itself (see Fig. 2). This allows us close the gap between the virtual and physical world. Sensor nodes attached to business item as the drums in the introductory example can directly act as service providers. CoBIs add the following key capabilities normal business items (also compare [3]):

- Computing
- Data Storage
- Monitoring
- Controlling
- Communication

Wireless sensor nodes such as the Berkley Motes [6], Ambient uNodes [7] or Particle Computers [8] can be used as enabling platforms for embedding those features into physical business items.

### 2.2 Logic on the Item

Functionality implemented on a single sensor node is subject to many constraints. For instance, the micro-controller is often a resource-restricted 8-bit processing unit, providing typically up to 512KB Flash memory for programs and only a small amount (around 2KB) of RAM for volatile data.

However, in spite of the limited amount of data space and computation power on sensor nodes, they can provide complex services to business applications by in network collaboration. There are two key properties of business logic that can compete with and outplay server-based alternatives.

The first is the distributability of business logic. Because business logic is item related it can often be split on item level. The processing power then can scale linear with the number of collaborative business items. While backend systems often need complex strategies to scale with an always-increasing number and speed of inputs, scaling is an intrinsic feature of sensor networks.

The second key property of most business logic is that it exposes a high locality concerning their information working set. As an example matching storage regulations against environmental conditions such as temperature or humidity can be

done locally on a single node, as all input is available. At the same time the number of possible matching regulations can be statically evaluated on item basis, so that the logic actually executed on a single node only has linear time and space requirements.

We suggest that instead of only collecting sensor nodes should interpret data and pass on results to the business logic. Executing the process logic close to the data source decreases the amount of data that has to be processed by the backend system. This results in less resource consumption for computation and communication and can in turn increase the responsiveness and the scalability of the whole system considering the amount of data that gets generated by a normal business process.

### 2.3 Architectures for Collaborative Business Items

Considering the growing number of sensor network platforms for services relevant to business logic the backend systems would have to interface a number of different sensor networks hosting different sensor types and using different data encoding.

Uniform service concepts for sensor networks have already been designed in the past however they either see sensor networks only as a data source like [2] or at least restrict the logic which is processed on the item to query statements [10]. Other systems like [9] also suggest new service architectures for backend systems.

In contrast to this related work we see the challenge in fitting sensor networks into existing business service architectures. Seeing a sensor networks as regular service providers frees business developers from the need to develop proprietary connectors and leaves sensor network technology the freedom of optimized implementations. The coupling point between the two worlds is a service interface that has to be provided by a collaborative item architecture.

## 3 Key Design Challenges

In this chapter we describe the key design challenges of a service-oriented architecture (SOA) integrating collaborative business items.

### 3.1 Interfaces

Supporting standard RPCs to the services running on the nodes could lead to a painless integration with backend systems. However, it can be shown that RPC is not the right abstraction to directly support on sensor nodes (see [4]). RPC is very restrictive about the calling semantics, which makes stub generation easy, but proves to be inefficient for the communication requirements of sensor networks.

Data packets in sensor networks are often built in a way to support cross-layer protocol optimizations. For this purpose the data is encapsulated in an efficient encoding that can easily be parsed by the system. Because of different protocol stacks and operating systems, this leads to very different presentations of structured information in a packet. As an example the Particle platform use a tuple-oriented data format enforcing strictly typed information. Motes use Active Messages allowing a direct mapping to the component interfaces of TinyOS. A client would have to support all different encoding in such a heterogeneous environment.

Even if we can understand the message encoding, we will still fail to extract sensible information from the data. If transport container and environment have both humidity sensor embedded, it is not possible to make a statement about neither absolute nor relative humidity, because both sensors will most likely have different sensitivity or different resolutions.

The goal has to be that interfacing sensor network services are not more complex than using backend services. All domain knowledge needed to understand interfaces to sensor network services needs to be made explicit in standardized service interfaces.

If we acknowledge the necessity of proprietary protocol layers, this means that sensor network messages need to convert to a uniform message encoding. We propose the use of "active" service descriptions. Additional to the interface descriptions they can contain information to generate transformation stubs implementing endpoints for both sensor network and backend service communication.

### 3.2 Addressing

Besides interfaces, addressing can be identified as one of the most essential parts about service interaction. It is closely coupled to topics like transport and routing, which are also key elements of wireless sensor network research [12,13]. For us it is important not to impose implications on concrete implementations of such protocols and algorithms via the addressing scheme. From the application's point of view, however, a common method for addressing is necessary for a truly service-oriented view on the network. Because the semantics of an address is determined by the concrete implementation, we need means for address translation in our system.

Arbitrary resolving scheme can be applied to describe the different needs for addressing. Like in a DNS system semantic hierarchies can be constructed across an address space. Once resolved, the client can use the address to communicate with the system hosting a service using the target address space.

In highly dynamical systems like item networks (in contrast to rather static sensor networks) this generates consistency problems. To illustrate this we may think of a temperature service in out smart drums scenario. Getting the temperature can easily be executed on any sensor node in the network. Typically higher-level logic is, however, not really interested in the temperature of some node *A* but in e.g. the temperature at a location *L*. If node *A* is in location *L* we can nonetheless execute the service there by resolving *L* to *A*. If node *A* physically moves, however, this leads to an obvious problem when calling the service again.

Another kind of service mobility leads to the same problem: migrating a service to a neighboring node. Once again the service cannot be reached using A's address. Those problems occur, because we once again replicated part of the service state (namely the current host) in the backend system. Exactly this we stated as a problem before, as now the problem is keeping virtual and physical world consistent. Constantly requesting new addresses leads to an immense overhead on the network traffic.

Once we also push the logic of matching a concrete address into the sensor network, we see this problem disappear. An efficient implementation could e.g. involve location-based routing algorithms. We can easily imagine other routing

schemes. Therefore we propose to introduce service proxies in order to hide the addressing information behind an IP-based addressing scheme. Those proxies represent a "service running on an address". This way we can avoid pushing the routing functionality into the client.

### 3.3 Discovery

By saying that we map service addressing and interfacing to IP-based proxies we only shifted the problem of service binding to IP technologies. However, the problem of service discovery can be handled fairly efficient in those networks. Two main approaches can be identified here: infrastructure-based and infrastructure-less approaches.

An example of an infrastructure-based service binding and discovery approach is described the Web Service Interoperability Standard, namely the UDDI registry (www.uddi.org). Without going into detail about the specific up- and downsides of specific discovery implementations, it can be said that infrastructure based approaches are useful for rather static service landscapes and often have a broader scope than just announcing functionality. They also may represent a bottleneck in a distributed system or at least need special care to setup.

The service enabling of sensor networks should be a rather "plug-and-play" oriented approach that simplifies the use of a specific technology and should not generate additional infrastructure dependencies. For those purposes infrastructure-less discovery seems to be the better choice. This kind of discovery either uses multicast announcements and queries or distributed hash tables to announce services throughout the network. Because service mapped to real world items via wireless interfaces can easily disappear or appear, services may move with the nodes. Therefore announcement-based discovery interfaces also provide monitoring functionality for the liveliness of a service. Multicast announcements can provide a powerful means to program dynamical business logic, that acts on the availability of a service and also can take trigger necessary steps if a service is unavailable.

### 3.4 Lifecycle Management

The first thing to do if some logic fails to discover and bind necessary service functionality is to try to deploy this functionality to the sensor network. Because services can be mapped to multiple nodes within the network it makes sense to install one lifecycle management interface per network. The whole sensor network acts as a container for services.

Creating container interfaces for all service hosts would create the need to generate proxies for all addressable entities. Those are not known beforehand, as they can be arbitrarily defined. In order to solve this problem we use deployment descriptions that can have the power of selection statements of any query language. Those descriptions are used to instantiate the service on the target host generating a local proxy. Addressing the service instance is implemented by the sensor network routing services, thus the expressiveness of deployment description is also directly depended on their addressing modes.

We assume precompiled binaries called *service executables* as the input for the sensor network, which are then passed to the single nodes. As stated before we are totally agnostic about the implementation language and content of the service executables. The lifecycle management service only needs a counter part on each node in the network to forward the binary to. Once the deployment request was issued it is left to sensor nodes to execute it and initiates a *service instance*. The success of the deployment can be verified end-to-end using service discovery. Functions for removal and temporary disabling of a service are added to the service interface of each proxy.

## 4 CoBIs Gateway Architecture

In this chapter we describe our implementation of a service-oriented architecture for enabling sensor networks to run business logic. The Collaborative Business Item Gateway Architecture implements an UPnP to sensor network gateway. We chose the Universal Plug and Play (UPnP) standard because of its lightweight, infrastructure less and yet complete approach. The Standard includes the Simple Object Access Protocol (SOAP), the Simple Service Discovery Protocol (SSDP) and the General Event Notification Architecture (GENA). Because UPnP uses Internet technology it can easily be included in most business applications. The implementation, however, is not UPnP specific but can be easily ported to e.g. Web Services.

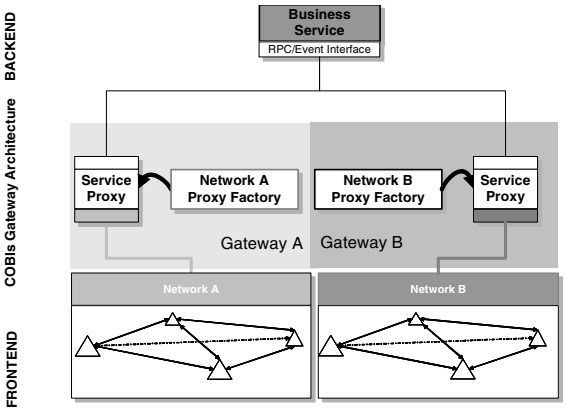


**Fig. 3.** The CoBIs Gateway Architecture

Key element of this architecture is the dynamic instantiation of service proxies. Proxies can be accessed like native UPnP devices, providing detailed service descriptions for the implemented functionality. The proxy itself, however, only exists as a virtual representation of the service interface. Request issued to the service proxy are transformed by the gateway to sensor network messages and vice-versa. From the backend the gateway itself is only visible for deploying new services to the sensor

network. Multiple gateways can be instantiated simultaneously to include several sensor networks into the architecture. The general architecture is depicted in Fig. 3.

### 4.1 Gateway Devices

The platform gateway is a application level bridge that handles all aspect of communication with the backend system. This includes all protocol levels beginning with the physical layer bridging from Ethernet to the wireless network ending with the application level bridging for service interaction from proprietary interfaces to RPC-style service interfaces. The idea of platform gateway is that it capsules all domain knowledge needed for communication with a sensor network platform. Because the interfaces to the backend system is always of the same type this allows exchanging the platform as well as using multiple sensor network platforms in parallel.

Because we handle the platform gateway as a monolithic software component in our architecture this does not mean that it is a single machine. The system has been designed to be able to both different levels of bridging as well as proxies to different services distributed on multiple machines. The only interface to the gateway from the backend system is the lifecycle management service that enables pushing new service binaries to the network.

This and all interfaces dynamically created service proxy are announced in the network. UPnP handles Service discovery natively once a proxy is initiated. All proxy instances provide a pointer to their description via GENA when requested by some client. For the gateway this means that proxy services have to be instantiated whenever a new service is provided by the sensor network and destructed when the service becomes unavailable. The proxy service instance itself is a only dispatching path associated with a SOAP request URL pointing to the gateway and a service description including a the interface transformation (see below).

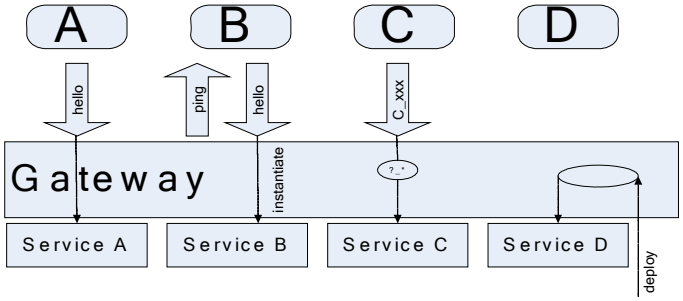


**Fig. 4.** Discovery methods supported by the gateway

Fig. 4 shows four means to initiate a service instantiation. They are provided by Discovery Services running on the gateway and can be exchanged. In the first case the gateway parses predefined “hello” and “bye-bye” packages (Fig. 4A). Querying actively instructs all services to issue hello packets by sending broadcast pings (Fig. 4B). Passive discovery omits special hello packets completely by adding a service identifier to all packets (Fig. 4C). Proxies can also be permanently installed on

deployment, requiring manual removal (Fig. 4D). This however disables support for liveliness monitoring.

### 4.2 Interface Transformations

We integrate the platform specific semantic transformation for the RPCs into the XML-based UPnP service description. Because the UPnP demands flexible XML parsing we can use the same description for providing the client description to the control point. Leaving the transformations inside the descriptions allows easy debugging of the transformations and allows the management services to analyze the running system.

The descriptions are automatically parsed by UPnP stack implementation, which already provides the UPnP RPC dispatching and eventing facilities. We integrate our transformation logic into the device instantiation that is guided by the XML description. This allows a direct coupling between transformation and interface.

We suggest a simple template-based transformation, we have successfully used for creating interfaces for Particle Computers and Ambient uNodes. Templates work as bi-directional transformation. Fig. 5 shows the model of the templates organized as an ingress filter tree that is loosely connected via a listener pattern to the UPnP protocol stack. When a RPC call is received each outgoing argument is serialized via the template the UPnP state variable and incoming arguments are parsed by inverting the process: the template is matched against the packet.

A template entry can be of three types of data, wildcard or don't care. If a matching template is found the information under the wildcards is extracted/serialized together with any optional static data. UPnP typing information contained in the interface description can be used for de- and encoding. This suffices for most standard integer data, but the scheme can be extended to support more explicit interfaces without the increasing message size. In cases more complex parsing is needed one may specify platform specific extensions for message decoding.

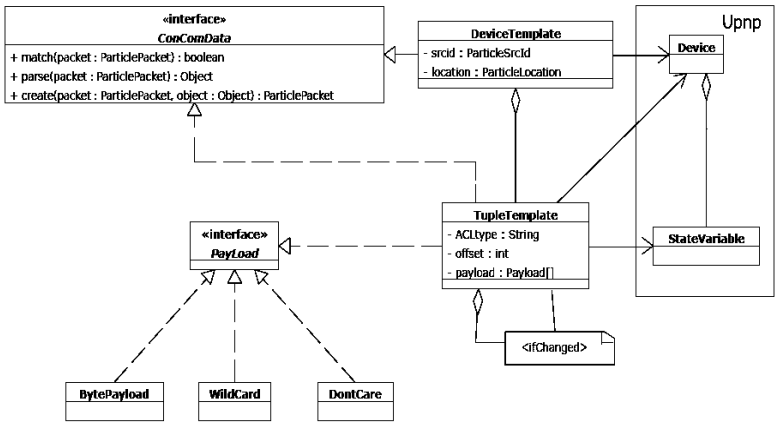


**Fig. 5.** Mapping message templates of RPC arguments

### 4.3 Message Primitives

We support seven message types for communication between sensor node services and backend-based services that can be accessed via the RPC interface:

- Non-blocking Send, Receive and Call
- Blocking Send, Receive and Call
- Callback

The classification of blocking and blocking messages concerns the behavior on the gateway not on the issuer as RPCs are always blocking. The non-blocking send can be used if no return types are given. It sends out a message to the sensor network and does not wait for an answer.

The non-blocking receive uses replicated services states on the proxy to answer the RPC. Here no function argument can be provided to RPC. An example would be a *getTemperature* action providing the most current temperature data. Caching data in the gateway allows the services on the sensor node to run in a duty cycle without loosing accessibility. The non-blocking call is a combination of the message types described before. It does not wait for data to arrive, but returns cached data. It can be used to signal the service that its data was received and consumed by the backend. The blocking call can be used to implement real RPC for a service. It sends out data and awaits an answer before returning. It can also be used to provide support for acknowledged that means blocking sending support as well as blocking receive by omitting either in or out parameters. For blocking operation we wait for a message matching the associated return argument. Timeout values may be specified individually for each function.

The callback provides support for a node service to asynchronously trigger the Business Logic in the backend. An example would be an alerting service that needs to result in rapid action within the backend. UPnP's General Event Notification Architecture (GENA) handles the callback subscription. Subscriptions are thus handled by the gateway.

## 5 Real World Trial

The gateway implementation described above was installed for a one-month trial at one of the BP's largest acetyls production site. The trial was conducted in a declassified storage area and included 21 chemical drums equipped with Particle Computers. The goal was to model storage regulation of chemical substances stored in two different stored with multiple storage locations inside the same store (see Fig. 6).

We implemented logic for multiple types of storage regulation:

- per chemical storage limit
- incompatibility classes of chemicals stored in the same location
- environmental constraints ( maximum/minimum temperature)
- maximum time in storage

The Business logic for the storage regulations was modeled in SAP's EH&S (Environment, Health and Safety) system. The system was used to parameterize a

so-called hazardous goods service on the nodes. This service needed input from a location service, which was implemented using a simple infrastructure-based infrared location system. A temperature monitoring service gave input to check environmental storage regulations. Additional management functionality (voltage, duty cycle) was provided to manage the networks functionality from Smart Items Management Console developed by SAP Research. The gateway software was run on two 200 MHz embedded Linux MIPS systems in combination with a with 2GHz Intel Windows XP server also running the monitoring logic.

**Fig. 6.** Storage location enabled with Collaborative Business Items

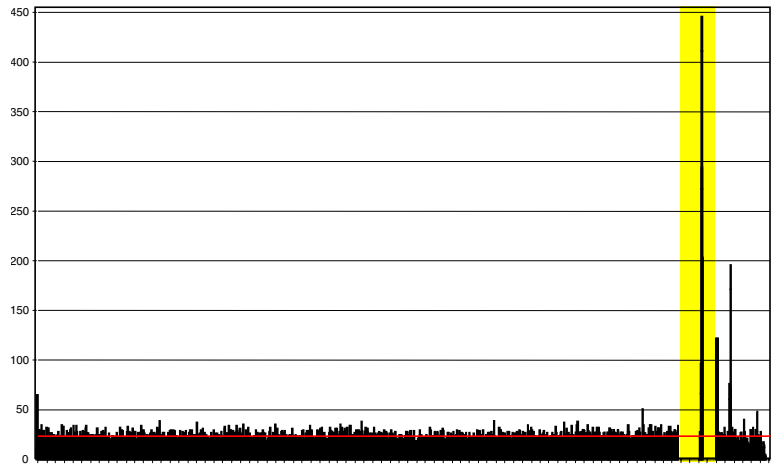


**Fig. 7.** Message load per minute to backend system

### 5.1 Trial Evaluation

Surprisingly most of the problems encountered on site were related to technologies not specific to our architecture. Using Internet technologies the hope beforehand was

to seamlessly integrate into any existing network using those technologies. As BP provided us with a wireless 802.11 based infrastructure we were able to connect our gateways without the need of complicated wiring on site. Probably because of the rather humid weather conditions close to the cooling towers the 802.11 network showed a high packet loss at times. This packet loss did not very much affect TCP traffic but unacknowledged traffic like UDP. UPnP uses UDP multicast as part of its discovery scenes and also the DHCP based dynamic addressing for the gateways used unacknowledged transport for discovery.

The logic pushed to the sensor network worked reliable. Additionally to the month long trial we performed an extensive test set at the end of the trial to confirm the correct behavior of the system. Both the business software running on an SAP application server as well as the sensor network performed their services by specification. The average message load to the business logic was only about 23 messages per minute (see Fig. 7), mostly resulting from voltage monitoring need for the management application. We were however able to put our system into an overload situation (right part of Fig. 7), when simultaneously generating alerts from all business items. This was due to the GENA implementation of the used UPnP stack, which set up new HTTP connections for each event and each subscription.

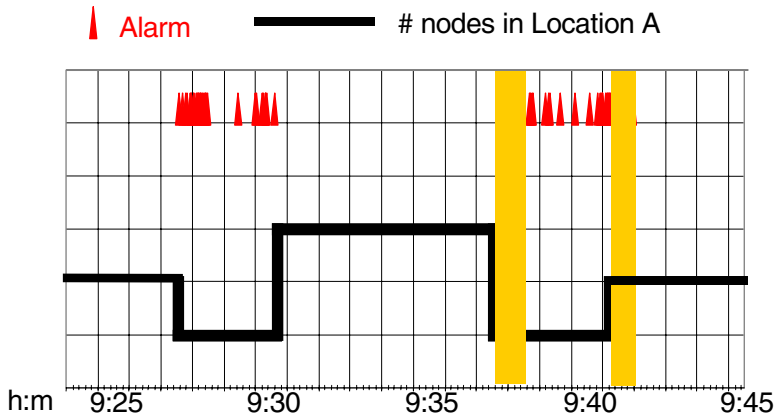


**Fig. 8.** Delayed issuing of alarm action

Only functions carrying state changing configuration information actively injected messages into the sensor network. All other services running on the nodes ran autonomously, only actively communicating to the backend system on events subscribed by the business logic. This scheme made the reactions of the sensor network especially in critical situations very robust as the message load was in some sense predictable within the system and message prioritization could be decided within the node network. This at times (due certain amount of message loss in unacknowledged traffic) leads to an inconsistent view of the system between sensor network frontend and backend (business logic/management) systems. Most events were retransmitted repeatedly so that the error was temporal and the correct logic was executed with a delay (compare Fig. 8).

## 6 Conclusion

In this paper we have demonstrated how a system can be build to easily integrate arbitrary sensor networks into business logic. For this purpose we did not stop at the point where we use this technology as a data source, but rather tried to unleash the computation power that results from deploying networked sensor nodes into real life environments. We showed that service oriented architectures are way to abstract from classical system designs using only backend driven business logic. The proposed system enables sensor networks to actively take part in distributed processes.

The implementation of our UPnP sensor network gateway shows that this technology can easily be used to adapt existing sensor node platforms to a service oriented business architecture. While trying to make the least possible assumptions about functionality of the services executed by sensor network during the design of our architecture, we showed by trial a specific use case that this technology can be successfully applied in real life settings. In spite of the still prototypical nature this system we hope that our experiences can help the deployment and integration of sensor node technology into business applications in near future.

## Acknowledgements

The work presented in this paper was fully funded by the European Community through the project CoBIs (Collaborative Business Items) under contract no. 4270. We further like to thank Paul McCune and the people at BP Chemicals Saltend for their support during the application trial.

## References

[1] M.Strohbach, H.-W.Gellersen, G.Kortuem, C.Kray. *Cooperative Artefacts: Assessing Real World Situations with Embedded Technology*. Ubicomp 2004

[2] C. Bornhövd, T. Lin, S. Haller, J. Schaper, Integrating Automatic Data Acquisition with Business Processes *Experiences with SAP's Auto-ID Infrastructure,* Proceedings of 30th VLDB Conference, Toronto, Canada

[3] Z. Nochta, N. Oertel, P. Spiess. *Relocatable Services and Service Classification Scheme*, CoBIs Deliverable Report, http://www.cobis-online.de/files/Deliverable_D101.pdf, 2005

[4] U. Saif, D. J. Greaves, *Communication Primitives for Ubiquitous Computing or RPC Considered Harmful*, 21st ICDCSW, p. 0240, 2001.

[5] C. Decker, P. Spiess, L. Moreira sa de Souza, M. Beigl, Z. Nochta.: *Coupling Enterprise Systems with Wireless Sensor Nodes: Analysis, Implementation, Experiences and Guidelines*, Pervasive Technology Applied @ PERVASIVE, May 7, 2006, Dublin, Ireland

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, *System Architecture Directions for Networked Sensors* ASPLOS-IX, 2000

[7] P. Havinga, *The Quest for Low Cost, Ultra Low Power Wireless Networks for smart environments,* Ambient Systems white paper, 2006

[8] C. Decker, A. Krohn, M. Beigl, T. Zimmer, *The Particle Computer System* Proceedings of the ACM/IEEE Fourth International Conference on Information Processing in Sensor Networks, Los Angeles, 2005

[9] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, *Hourglass: An Infrastructure for Connecting Sensor Networks and Applications,* Harvard Technical Report TR-21-04, 2004

[10] S. Madden, M.Franklin, J. Hellerstein, W. Hong. TinyDB: An Acqusitional Query Processing System for Sensor Networks. ACM TODS, 2005

[11] *Universal* Plug and Play Device Architecture, Microsoft Corporation, 1999.

[12] C. Intanagonwiwat, R. Govindan and D. Estrin, *Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks.* In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCOM 2000), August 2000, Boston, Massachusetts.

[13] J. Kulik , W. Heinzelman , and H. Balakrishnan, *Negotiation-based protocols for disseminating information in wireless sensor networks.* Wireless Networks, March 2002.