An Agent-based Sensor Middleware for generating and interpreting Digital Product Memories

Christian Seitz and Thorsten Schöler
Siemens Corporate Technology
Intelligent Autonomous Systems
Munich, Germany
[ch.seitz|thorsten.schoeler]@siemens.com

Jörg Neidig
Siemens Industry Sector
Advanced Technology and Standards
Nuremberg, Germany
joerg.neidig@siemens.com

ABSTRACT

Automatic identification technology such as the Radio Frequency Identification (RFID) technology has significant value for production processes, supply chain management, and inventory systems. For future purposes the idea of storing a simple ID must be extended to a Digital Product Memory. This memory provides a digital diary of the complete product life cycle that is embedded in the product itself, using smart wireless micro-sensor technology. It records all relevant ambient parameters in digital form. Sensors note where and when an interaction with a product takes place and the necessary data is added to the memory. A key problem with digital product memories is their cross-domain nature, i. e. new relevant data must be added to the memory from various stakeholders during the complete product life cycle. In this paper we propose an agent-based architecture for appending sensor data to a digital product memory in a generic way. Additionally, methods are presented how a digital product memory can be analyzed and evaluated to simplify business processes, to ease maintenance, or to bring benefit to the end-user.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent System; D.2.11 [**Software Architectures**]: Data abstraction

Keywords

Digital Product Memory, Sensor Abstraction, Sensor Modelling

1. INTRODUCTION

Automatic identification technology such as the Radio Frequency Identification (RFID) technology has significant value for production processes, supply chain management, and inventory systems. For future purposes the idea of storing a simple ID must be extended to a Digital Product Memory. This memory provides a digital diary of the complete product life cycle that is—at least to some extend—embedded in the product itself using smart wireless micro-sensor technology. It records all relevant ambient parameters in digital form. Sensors note where and when an interaction with a product takes place and the necessary data is added to the memory.

There are already some consumer application which describe a simple RFID-based form of a product memory. The authors in [19] present the Digital Sommelier, an interactive wine shopping assistant that provides general product information. Wine bottles sense their state via attached wireless sensors and detect user interaction over RFID and acceleration sensors. Multi-modal interfaces allow the access of information either through physical interaction with products or a natural language interface. With such a solution a buyer of a product can check if the product was aging adequately and is still in a good condition.

The Smart Shopping Assistant [20] application observes the users' interactions with products in a supermarket. On displays mounted to the trolleys, context-dependent user support is provided by the display. The user is provided with detailed product information or a list of recipes that could be prepared with the selected ingredients. For each product, information including the product name, a textual description, the price, and a set of physical and technical properties is manually entered into a SQL database. The information is in turn related to the object through a globally unique identification number.

A key problem with digital product memories is their cross-domain nature, i. e. new relevant data must be added to the memory from various stakeholders during the complete product life cycle. This paper introduces an agent-based middleware for creating, analyzing, evaluating, and interpreting digital product memories. Additionally, applications and scenarios from the automation domain are presented, which show the benefit of a digital product memory and of our sensor middleware approach.

The paper is organized as follows. The next section presents existing approaches for sensor middleware architectures. Chapter 3 defines our requirements, it is followed by a description of the agent-based middleware with Chapter 4. Chapter 5 presents the implementation and Chapter 6 shows some applications for the sensor middleware in combination with the digital product memory. The paper concludes with a summary and a future outlook.

2. RELATED WORK

Various middleware approaches for sensor networks, with the goal to provide a unified access to sensors, are existing. The authors in [17] discuss challenges for sensor middleware

Cite as: An Agent-based Sensor Middleware for generating and interpreting Digital Product Memories, Christian Seit, Thorsten Schöler, Jörg Neidig, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 61 – 68

approaches, but their analysis is limited to sensor networks. Our approach must also include sensors which are part of existing systems, e. g. automation systems.

Kumar et al. [16] present a sensor middleware with autonomous services. These services deal with basic sensor network functionalities like a status monitor or monitoring quality of services issues. Although a uniform access to heterogeneous systems is part of the system, no sensor modelling is included in their approach, which would have made their approach even more flexible.

In addition to creating an interface that realizes a uniform access to get sensor values, the configuration of sensors has also taken into account. Grace et al. [10] consider reconfiguration of the network layer. We suggest that not only the communication infrastructure needs to be reconfigured, there must also be a possibility to change the sensor parameters (e. g. measuring range) as well.

According to [12] four classes of sensor middleware solutions exist. Each of the classes solves just a subset of problems. Database inspired solutions provide access to sensor networks similar to databases, i. e. they are integrated into a schema and can be accessed using a SQL-like query language. Several solutions support query optimization (e. g. energy-efficient query processing) or sensor configuration. Most of the solutions are limited to a homogeneous sensor infrastructure, like in [9]. Solutions based on tuple spaces are comparable to database-inspired approaches. Data from different sensor networks are integrated into a virtual storage space. Systems like TinyLime [8] are limited to local sensors, whereas a global tuple space solution would be very challenging because of scalability. Event-based systems adopt the publish/subscribe paradigm for sensor infrastructures (e. g. [23]). They especially support operators for event correlation. The fourth class of service-discovery-based solutions as well as event-based systems address just one aspect of creating a global sensor infrastructure, i. e. the event propagation or the discovery of sensors. The approach of Sensor Web Enablement (SWE) of the OpenGIS consortium [7] aims at the integration of sensor devices into an open sensor infrastructure which is accessible in a web-based manner for arbitrary services. The solution consists of a set of specifications divided into three XML document descriptions (SensorML, Observations & Measurements Schema (O&M) and TransducerML). The solution provides a unified access to geographically dispersed sensors, but the processing steps for abstracting sensor data is left to the clients. While the SWE approach is rather general one, our solution is currently focused on the industry domain and integrates raw sensor data and results of hierarchical processing steps into a unified infrastructure which allows the dynamic discovery and sharing of all these components.

The creation of product memories is discussed in [15]. The authors suggest to split the memory in a short-term and a long-term memory. The short-term memory contains the raw data and mechanisms exist to filter and aggregate the data. This higher level data is then the long-term memory and stored in a database. The focus of the paper is more the exploitation of memories than the definition of a generic middleware concept. Wahlster et al.[22] describe an extension of this approach and present the SmartKitchen project. A semantic cookbook is created by monitoring persons which are using the kitchen. The recorded information can be shared locally or globally over the Internet.

3. REQUIREMENTS

The goal of our research was to develop a comprehensive cross-domain approach for creating digital product memories. We assume a distributed, heterogeneous sensor infrastructure or other relevant information sources like user input, databases, or web services. Because of the dynamic character of pervasive environments, the solution should be flexible and extensible. That requires especially independence from the used sensor types, the structure of the sensor networks, and of the different real world situations which could occur. Also the solution should not depend on a specific application domain, because the product memory must be accessed during the complete product life cycle. Furthermore, scalability and modularity is required, i. e. the solution should not restrict the amount of deployed sensors. For better reuse, it is necessary that the system is designed in building blocks reflecting the different abstraction steps of the conceptual framework. Additionally, performance has to be considered to address the requirements of different applications. Based on the assumption of a heterogeneous and distributed sensor and service infrastructure the self-description, discovery of building blocks of the infrastructure is required to create an open sensing infrastructure. While sensors are usually bound to a certain location, the provision and handling of location information is of special importance. The solution should also enable the sharing of any component by multiple applications. Thus, all services and sensor access components should be configurable to the needs of different applications. Moreover, several applications could use the same sensor device at the same time in a concurrent manner, e. g. a sensor can add information to more than one product memory at a time. For later processing, as well as for legal requirements, sensed and processed data should be persistently stored for configurable periods of time. Storage functionality should be integrable at all steps of processing of data in a dynamic and configurable manner. Even if the infrastructure should be open, security and privacy have to be supported. Thus, the solution has to provide means for restricting access to sensors and further infrastructure components to authenticated and authorized users or systems only.

4. ARCHITECTURE AND FRAMEWORK

In this chapter we present the architecture and generic framework of the sensor middleware for creating and interpreting digital product memories. The architecture is shown in Figure 1. It consists of a sensor infrastructure layer, infrastructure services, an agent system and the application layer. The building blocks are explained in the following sections in detail.

4.1 Sensor Infrastructure

The sensor infrastructure layer (see Figure 1) contains the physical sensors and a proxy concept. The proxy concept is the foundation concept of the sensor middleware. It can be seen as an abstraction for wrapping heterogeneous sensing devices and providing unified access to sensor measurement and configuration, hiding the details of raw data acquisition and sensor configuration from the higher-level components.

The proxy concept, is shown in more detail in Figure 2. A proxy represents either a single sensor or a collection of sensors, e. g. from a sensor network. A collection of sensors may even be heterogeneous. In that case, the Sensor

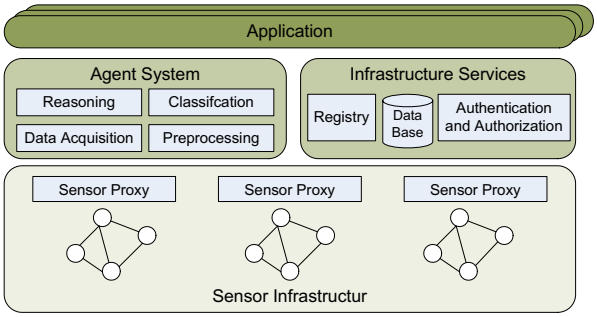

Figure 1: Overall middleware architecture

Manager integrates all devices, manages them internally and represents them to the outside as one logic sensor network. The proxy manages meta information about its sensor devices (e. g. sensor type, unit, accuracy and location) and uses this information to register the sensors with an external Registry to enable the discovery of the sensors. This meta information can be requested from the proxy as well. To enable a unified access to heterogeneous sensing devices, the sensor proxy provides methods for accessing raw sensor data, for configuring individual sensor devices and for monitoring their state. Access to sensor data is supported in two ways. First, data can be requested from the proxy. Second, other components can subscribe to changes of sensor measurements. Because the proxy should be usable by multiple applications or components at the same time, it has to coordinate competing data requests and subscriptions internally. In addition, potentially large amounts of raw data should be processed in an application independent and configurable way.

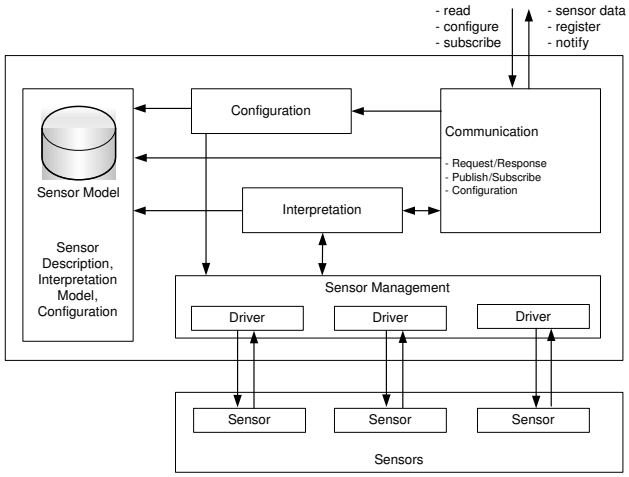

Figure 2: Architecture of the sensor proxy in detail

4.1.1 Communication

The Communication subcomponent is responsible for handling all communication of the sensor proxy with the "outside", i. e. other components of the system infrastructure. Thus, the Communication component provides the unified interface for sensor access and configuration. It handles multiple, potentially concurring requests and subscriptions and provides the means for sending messages to remote components, e. g. notification messages to registered subscribers or messages to register sensors at the registry component.

4.1.2 Sensor Management

The Sensor Manager subcomponent maintains the low-level concerns of the management of sensors and sensor access inside the proxy. It manages a driver for each sensor or sensor network and maps requests from the higher-level components of the gateway proxy to the physical sensor devices. The driver abstraction provides interfaces for requesting sensor values, registering for sensor value updates and for the configuration of sensor devices and data access. Thus, it represents the link between the abstractions for data access and configuration at the one hand and the sensing hardware at the other hand. The driver abstraction hides all implementation details from the proxy developer and provides a convenient way to integrate heterogeneous sensor devices into a proxy.

4.1.3 Sensor Model

The Sensor Model contains meta data about all sensor devices managed by the sensor proxy. There are various ways to describe sensors.

The Sensor Model Language (SensorML) provides an XML schema for describing sensors systems and processes, it provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of taskable properties [7]. SensorML is rather complex and a sensor model can get quite large. There is a first version of the SensorML specification available, but the complete sensor web enablement (SWE) modelling stack is still not coherent.

Additionally, we evaluated IO-Link [5]—a communication standard which was developed by the IO-Link research group of the PROFIBUS & PROFINET International organization. The communication standard below the field bus level enables a central diagnosing and locating of errors down to the sensor/actuator level, and facilitates the commissioning and maintenance, by permitting parameter data—directly from the Programmable Logic Controller (PLC)—to be dynamically changed. The standard uses an XML-based device description, which comprises the most important characteristics of a sensor (e. g. vendor, sensor type, parameters, unit). Unfortunately, no meta information like the location is considered in the standard.

Both modelling approaches seem very promising. SensorML is pushed by university organizations, whereas IO-Link will become an industry standard. Both approaches complement each other, because each modelling technique has its certain application domain. Nevertheless, both approaches are in an early stage, i. e. no mature tools are available to create a sensor model and no query languages exist. We believe that these tools will soon be available.

In the meantime we use our own simplified meta format. This meta data comprises a logic ID per sensor, the physical addresses of the sensors, sensor type information, the unit of the values provided by a sensor, the accuracy of sensed data, configuration parameters, the location of sensor devices and a method for interpreting raw sensor data (among others). These data is also entered in the registry component, which

make an efficient sensor discovery possible.

4.1.4 Configuration

The Configuration subcomponent is responsible for interpreting configuration requests of higher level entities and for carrying out the configuration of the underlying sensors accordingly. One important aspect is to resolve conflicting configuration requests and to make sure that the sensing components perform their task in a resource efficient (energy, bandwidth, etc.) manner. While the conflict resolution mechanisms strongly depends on the type of the particular configuration parameter, the Configuration subcomponent has to be tailored to the sensor devices managed by the sensor proxy.

4.1.5 Interpretation

The Sensor Manager subcomponent provides raw sensor data according to particular data requests and subscriptions. This raw data may be delivered directly to the requesting instance or could be interpreted inside the proxy to reduce the amount of data which has to be transmitted via the network. The interpretation subcomponent is responsible for this task. Components that access the infrastructure can request raw sensor data or interpreted sensor data. If interpreted sensor data is requested, the Interpreter subcomponent is invoked. It refers to the Sensor Model to access the interpretation model for the requested sensor value and performs the interpretation.

4.2 Infrastructure Services

The Infrastructure Services provide mechanisms which are necessary but generally optional. The services can be used by the agent system and by the application as well. The currently integrated services are explained in the following.

Registry: In our current framework an attribute-based registry is used for discovering sensor proxies. It provides its service under a well known URL. Components that should be discoverable can register with the registry based on a unique identifier, a component type and a set of attributes in the form of name/value pairs. Especially, sensor type, quality information of sensors, and the location of components are represented as attributes.

Database: A database component is responsible for managing historical data of the digital product memory. Parts of the product memory can be attached to the product itself, e.g. with a bar code or RFIDs. Product memories can be large and may not fit on these storages, or a product memory cannot be accessed at all times because of a lack of communication infrastructure (e.g. during transportation). Therefore, an additional storage is needed which either stores parts of the product memory or which is responsible for memory replication.

The storage mechanism (e. g. a database, a RDF triple store, or a file) is encapsulated in a Storage Manager who is responsible for storing, delivering or synchronization of the data. A request frequency can be provided, which determines the period in which data is requested from that source, i. e. the granularity in which historical data is available. Moreover, the configuration contains a maximum age of data to control the deletion of old values.

Authentication and Authorization Service: The Authentication and Authorization Service is responsible for authenticating users in the system and for granting access to system components to authorized users only.

4.3 Agent System

The agent system provides mechanisms to get the data from the sensor proxy, to perform an optional preprocessing, and finally to interpret the data accordingly. For each mechanism, a single component is responsible that can be attached to an agent to achieve the functionality.

Sensor Data Acquisition: The agent system must be able to get the data from the sensor proxy. This module is responsible for discovering the right sensor proxy and must be able to obtain a single sensor value as well as to subscribe to a sensor proxy in order to be provided with sensor data continuously.

Preprocessing and Aggregation: If agents get sensor data from a sensor proxy, not all the data needs to be stored in a product memory or a special format is needed. For that reason the preprocessing components comprise mechanisms like outlier extraction, noise reduction filtering, or data transformation. The next step is the data aggregation, when several agents need data from more than one sensor proxy. The sensor data often must to be combined in a special way to increase expressiveness. This is achieved by several aggregation functions like average, maximum and minimum, or summing up.

Classification: Classification implements certain algorithms for grouping items that have similar feature values into groups, e. g. assigning raw sensor data provided by a sensor proxy to a certain value class. Algorithms to be adopted are for instance crisp limits or fuzzy sets. Thus, a classifier requires a single value as input on which it applies the implemented classification algorithm, configured by the classifier settings (e. g. classifier ranges), and thus, produces one resulting fact representing the classified sensor value.

Reasoning: The most sophisticated mechanism is a reasoning component, that makes it possible to generate new higher-level data from pure sensor data. It requires a set of lower-level facts as input and produce a higher-level fact about the current situation as output. For example, data of two light barriers can be used as a direction sensor or event, provided that a time stamp is available, the velocity of an object can be calculated. This is accomplished by using rule engines (e. g. CLIPS, Jess, or Drools) and OWL Reasoner (e. g. Racer Pro, Pallet or Fact++).

4.4 Communication

The communication framework is a general building part of the generic framework which is reused in most of the components for implementing their communication capabilities. It is not explicitly mentioned in figure 1, because it is more the glue between the components. It is an abstraction layer which hides communication technologies regarding to protocols and message formats from the programmer and the components into which it is integrated. The major functionality it provides is for sending and receiving messages to and from remote components. It represents a component as an addressable communication endpoint in the system. Moreover, it provides hooks to implement component specific functionality for handling and sending messages. The main component of the communication framework is the

Communication Manager. It contains the implementation of the basic functionality for encoding and decoding messages and encapsulates the concrete technologies and protocols for sending and receiving messages. In our framework we currently use Jetty, an open-source, full featured web server, which can be embedded into Java applications.

Thus, components can access functionality and data of other components and offer their functionality and data in a web-based manner and are addressed with a unique URL. The communication protocol is HTTP 1.1. The communication framework defines a set of message types representing messages, which belong to a certain interaction type (e. g. request message, subscribe message, etc.). Messages are identified by a unique ID and a particular message type. Each message contains a set of data which is specific to the message type. This data is encapsulated in a message body. Currently, messages are defined in XML format. The messages could easily be implemented also in another, maybe binary format. Nevertheless, advantages of XML are easy parsing based on available parsing library implementations, easy-to-implement XML-schema-based validation of messages, easy extensibility, and a human-readable structure. The major drawbacks of XML-based messages—high overhead and slow parsing—are avoided by defining a very simple message format with short identifiers.

4.5 Design issues

The proposed architecture and framework structure has several consequences on the performance and scalability of the systems build on it, as well as on the reusability of framework code and components. Therefore, in the following sections, the major design issues are discussed and the design decision are made explicit.

Sensor proxy granularity: A major goal for the design of the proxy abstraction is to introduce a unified interface for accessing sensor data and for configuring sensors, no matter what kind of sensor is represented by a particular proxy. In practice sensor devices can have different granularity (e. g. a single sensor vs. a sensor board with multiple sensors attached). Thus, a proxy could either represent only a single sensor or it could handle a set of sensors by definition. We decided for the latter, to ensure a unique granularity of the proxy interfaces and a more natural and efficient modelling of sensor networks. Thus, to address a sensor within a proxy, always a unique ID for each sensor is required.

Sensor proxy vs. Agent system relation: The sensor proxy abstraction covers sensors providing raw data. Several types of sensors (e. g. a microphone or an accelerometer) produce a large amount of data. The agent system processes such a stream of raw data, resulting in facts and reducing the amount of data significantly. We decided to detach the sensor proxy from the agent system in order to support a clear separation of concerns. To avoid the transmission of large amounts of raw data, agents should be placed locally to sensor proxies, either on the same host or on a host with a fast connection to the proxy host.

Sensor proxy vs. database: From the performance perspective, sensor proxy and storage should be closely coupled to optimize storage structures and data access. This would increase the complexity of the proxy abstraction. The other option is to separate the storage from the proxy. We decided for the second option again to support a clear separation of concerns and a simple proxy solution. With the introduction of the storage abstraction a uniform access to data history can be provided at all levels of the architecture, which also reflects the idea of a digital product memory. For the drawback of that approach, see last paragraph.

5. IMPLEMENTATION

This section describes briefly our implementation of the sensor middleware. It explains which sensors are currently integrated and which agent system is used.

5.1 Sensor Infrastructure

We attached various information sources to the sensor proxy. For each of them a driver component was developed within the proxy, see figure 2.

We started with integrating a Crossbow Imote2 module. The Imote2.NET is an advanced wireless sensor node platform. It is built around the low-power PXA271 XScale CPU and also integrates an 802.15.4 compliant radio [4]. It has 32MB SDRAM and 32MB of FLASH memory at its disposal, which is enough for a lot of entries of a digital product memory. The sensor board contains a three-axis accelerometer, an advanced temperature/humidity sensor, a light sensor and a four channel A/D converter The Imote2 is integrated via an 802.15.4 interface to the sensor proxy.

Additionally, driver exist for serial and parallel communication with sensors, IP-based communication is supported as well.

We do not only concentrate on physical sensors, other information sources can be added as well. There is the possibility to use databases, files or URLs as data sources for the product memory.

5.2 Agent Platform

As already mentioned, we need another kind of infrastructure on top of the sensor proxy, namely a multi-agent system, which is responsible for preprocessing and interpreting the data, that is provided by the sensor abstraction layer.

5.2.1 JADE vs. Cougaar

We have gained a lot of experience in the Agent Platform JADE (Java Agent DEvelopment Framework) in the last years and developed a lot of prototypes, mainly in the mobile computing domain. JADE is a software framework fully implemented in the Java programming language. It simplifies the implementation of multi-agent systems that complies with the FIPA specifications and through a set of tools that supports the debugging and deployment phases. JADE is relatively suitable for simple agent applications and developers, that require FIPA compliance. JADE's debugging and monitoring tools, and its support for FIPA ACL message format are a sound foundation for systems interactions that require cross-platform interoperability [11]. JADE can be distributed over several hosts, resulting in a distributed system that seems like a single platform from the outside. White and yellow pages services are available. Additionally, ontologies are supported and a large number of plug-ins and 3rd party software is available [6].

We additionally started an evaluation of the Cougaar (cognitive agent architecture) platform [3]. Cougaar is a Java-based architecture for large-scale agent systems. Cougaar has been used in distributed logistics planning and replan-

ning systems. Cougaar is an open source project and is the product of a multi-year DARPA research project. The Cougaar architecture is based on a node/container concept, where each agent is implemented from a basic agent and additional plug-ins. The agents communicate via a publish/subscribe-based blackboard system. Cougaar itself is described as a highly configurable, robust and scalable architecture. It is designed and tested for scalability, performance and robustness. The plug-in concept adds greater flexibility and performance in more complex agent designs. An automated military logistics planning and execution system was built with Cougaar, in which more than 1100 agents were involved. Additionally to the open source version, there is a commercial version of Cougaar, called ActiveEdge which is offered by Cougaar Software, Inc. ActiveEdge provides all features of Cougaar with key extensions to simplify application development, increase agent functionality, and provides enhanced system capabilities. ActiveEdge is designed to provide a sound real-time picture of enterprise operations. Additionally, ActiveEdge provides advanced execution monitoring and collaborative decision support [2].

For our further developments, we decided to use the Cougaar platform in our sensor middleware architecture. Not only the availability of semantic technologies (e. g. Minsky's frames [21]) in Cougaar but also the gain in experience with applications of Cougaar (as described in Chapter 6) and its proven features like its maturity, its robustness, its scalability[1] and the additional possibility to switch to a commercial system with little effort—if necessary in the future are backing this decision.

5.2.2 *Sensor Proxy - Cougaar Interaction*

Figure 3 shows the combination of the sensor proxy and Cougaar agents. For a sensor or a set of sensors a software

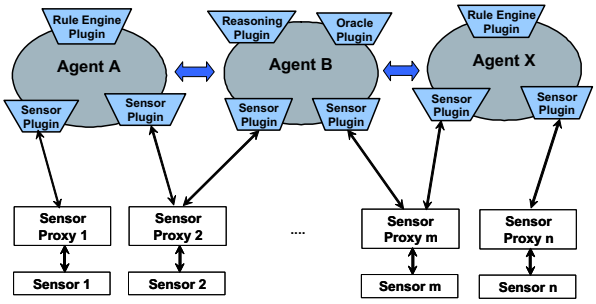

Figure 3: Sensor Proxy - Cougaar interaction

abstraction, a sensor proxy, exists. The sensor proxy is responsible to access the sensor hardware, i. e. the proxy must be able to read the sensor data and performs an optional sensor configuration.

A Cougaar agent's functionality is determined by its plug-ins. In order to obtain values from a sensor the agent must contain a sensor plug-in. This plug-in connects an agent with the sensor proxy via a HTTP connection. By this way an agent subscribes to a certain kind of sensor data, which is put in the agent's blackboard. A Cougaar agent can share its blackboard with other Cougaar agents. The data in the blackboard can be further processed by other plug-ins. We implemented various filter, data transformation and data aggregations plug-ins. Additionally, we made a plug-in for a rule engine, which allow basic reasoning mechanisms. With a database plug-in it is possible to transfer sensor data to a database, which is part of the infrastructure services.

Applications also need be attached via plug-ins. In the next section we present some of them, which we already have successfully realized or which we are currently implementing.

[1]Cougaar's scalability compared against e. g. JADE is also described in a post on the Cougaar mailing list [1].

6. APPLICATIONS

We have used the Cougaar agent platform in a variety of applications. One of them is intelligent system management in the automation system domain. System management can be divided into three major services:

- **Asset management,** providing access to system configurations, hardware/software revisions, and allowing target/actual system configuration comparisons
- **Event management,** providing services for system monitoring, alarming, and event correlation
- **Software management,** providing services for software deployment and patch management

Additionally, we have gained a lot of good experience with Cougaar in the following applications:

- **Decentralized Automation,** the product memory controls and influences the manufacturing process
- **Predictive Maintenance,** detection potential product failures in advance

Applications of the Cougaar agent system for asset management, event management, decentralized automation, and predictive maintenance will be briefly described in the following sections.

6.1 Asset management

The asset management system uses Cougaar agents, deployed on standard PC systems, to collect asset information of automation equipment deployed in various industrial applications. The Cougaar PLC proxy agent (as shown in Figure 4) collects asset information for a set of deployed programmable logic controllers (PLCs). The collected asset information is stored by a PLC asset information plug-in in a hierarchical system description, based on Cougaar's semantic frame set data structures, which is quite similar to the semantic product memory approach. Other Cougaar plug-ins are used to enrich and analyze the collected data. Finally, exporter plug-ins are implemented that provide the collected asset information to external systems (e. g. as XML representation).

The collection and processing of asset information in the described use-case would benefit from a standardized, hierarchically organized system description as envisaged with the semantic product memory. The laborious asset information collection as well as the integration in other asset management systems will be simplified.

6.2 Event management

One current challenge in event management for intelligent system management is how to reduce the event count produced by the managed system. Todays managed systems

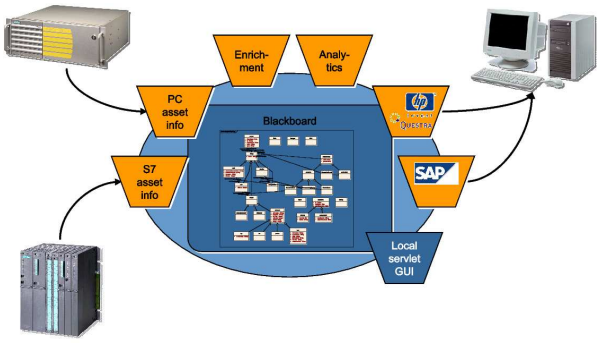

Figure 4: Cougaar agent configuration for asset management use case

e. g. computer tomography systems are based on one or more PC-based subsystems as well as a number of embedded devices. In our scenario, the event information, produced by the various PC or embedded systems, is collected and sent to a remote service center connected via the Internet. Thus, a reduction of the event count will lower communication costs as well as interpretation and analysis costs.

We efficiently reduce the number of reported events by correlating simple events produced by the various subsystem to higher-order events which allow better interpretation.

Again, the system is Cougaar agent-based, where an event correlation agent (see Figure 5) is deployed on the managed system. It attaches to various event sources (via importer plug-ins), correlates them and exports higher-order events via exporter plug-ins to the event management system, which in turn forwards them to the service center. The Cougaar agent uses a frames-based semantic representation—similar to a semantic product memory—of the managed system. Occurring events are attached to the source subsystem in the frame set. The actual event correlation task is carried out by a rule engine plug-in.

Again, the event correlation/management use-case will benefit from a more generic system or device representation as it will be available with the semantic product memory. Furthermore, the event collection process will be much simpler when there is a well-defined interface to the managed system as provided by a generic sensor middleware as described above.

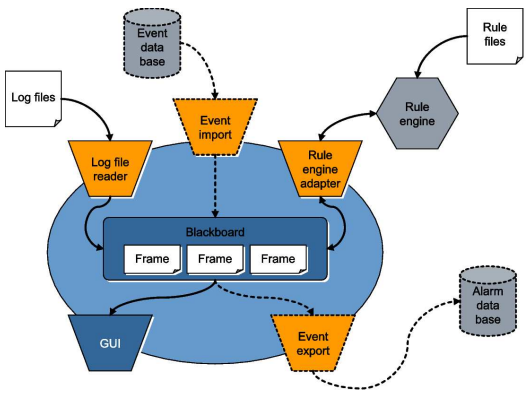

Figure 5: Cougaar agent configuration for event correlation use case

6.3 Predictive Maintenance

Another possible benefit of a digital product memory is the detection of technical issues when the product is already deployed in the field. Since the digital memory contains a lot of sensor data, an analysis is useful for diagnostics and predictive maintenance.

An example is an industrial robot, that moves a work piece from one machine to another. The robot is equipped with an acceleration sensor. While the robot moves the work pieces, the sensor values are continuously (every 20 ms) recorded in the digital product memory of the robot. On basis of this data, an evaluation of the robots product memory can detect, if the robot still works properly or if an execution has occurred. A more sophisticated method is not only to detect errors if they have occurred, but to detect slight changes in the normal behavior. An anomaly detector should report if new measured data of certain sensors have a different pattern or are beyond the normal sensor values. The basis for an abnormality detector are time series of sensor values or other data sources (e. g. from databases), which are element of the product memory. A given time series which describes the normal behavior is the basis for creating a model. This configuration phase is followed by a detection phase in which a detector should detect any candidate of an abnormal behavior. This detection step should be as fast as possible—in many cases even real time processing is necessary.

For the detection of abnormal behavior we implemented two different algorithms—both are integrated into the Cougaar platform by means of plug-ins. The first abnormality detection mechanism is based on the HOT SAX algorithm, [13]. The main idea of the algorithm is to analyze the sensor data history, regarding abnormalities. A parameter specifies how many data are to be extracted from the digital product memory. Thus, if new data arrives, it is compared with the data in the history and the distance is calculated. If a threshold is exceeded, an abnormality has occurred.

The other algorithm is based on the work of Salvador [18] and was extended in [14]. The algorithm transforms the data of the digital product memory into a multi-dimensional feature space, which results in a trajectory. This trajectory is covered with shapes (e. g. boxes). An abnormality is detected if new data is not within these shapes. The size and the location of the shapes are transformed into rules. Thus, an abnormality test can be done by using a rule engine (e. g. Jess, CLIPS, Drools). The algorithm is best suited for periodic events, is highly sensitive and a small rule set is sufficient to detect abnormal behavior. It can also be implemented on resource-limited devices like sensor nodes or PDAs.

6.4 Decentralized Manufacturing Control

The need of products which are tailored to customers' needs, results in a reduction of the lot size and implies a more flexible production and the associated processes. In the course of an increased diversification the changeover time will be a critical cost factor. This essentially needed flexibility is hard to realize with traditional central control architectures that can be found in nowadays automation systems. One solution is a decentralized production control, done by the product itself. The goal is to operate autonomous working stations and all data that is needed to assemble the product is kept on the product in the digital product memory. If a product enters the vicinity of a working station the in-

formation is read from the digital product memory and the station accomplishes the necessary tasks.

Thus, if a product is assembled in multiple steps, the necessary data is written to the product memory when the order is entered into the order system. The data contains the description of the single production steps with all its parameters, e. g. the position of bore holes or welds, the used materials, the size or the color. The product memory can even contain program code for the producing machines. The memory is read by the machine (e. g. via RFID) and the machine is parameterized and set up accordingly. If a production step is finished, the product itself is responsible for the routing to the next station. Depending on its weight and shape either an automated guided vehicle or a conveyer belt can be used. This use case of course needs a completely new infrastructure for shop floor systems. New communication mechanisms (e. g. sensor networks) must be introduced and additionally new software services are necessary. One of them is a sensor middleware in combination with an agent platform (e. g. Cougaar with its facilities for logistics planning and replanning) for the evaluation of the data and control of the working stations.

7. SUMMARY AND FUTURE WORK

This paper presents and agent-based middleware for creating and interpreting digital product memories. A product memory provides a digital diary of the complete product life cycle.

One building block of the middleware is a sensor proxy, which implements a software abstraction of one or more physical sensors. The sensor proxy offers a well defined interfaces for getting sensor values and for configuring the sensor. We use the Imote2 sensor network node for obtaining sensor data.

For each sensor, a sensor model is available that contains its special characteristics (e. g. vendor, type, location, accuracy). The sensor description is stored on a registry component, which is also able to process queries according to the sensors.

On top of the sensor proxy the agent platform Cougaar is used to evaluate the data, which are provided by the sensor proxies. Cougaar agents are responsible for filtering, aggregating and interpreting of the sensor data. A digital product memory is created and can be interpreted with additional mechanisms, e. g. querying or reasoning. Additional services allow storing or replicating the product memories in external databases. We use the middleware approach for applications in the automation domain, e. g. for predictive maintenance, decentralized control or asset management.

In the future we plan to gradually extend the system to other domains to enrich the digital product memory with data from other parts of the product life cycle.

8. ACKNOWLEDGMENTS

This research was funded in part by the German Federal Ministry of Education and Research under grant number 01 IA 08002 G. The responsibility for this publication lies with the authors.

9. REFERENCES

[1] Cougaar Developer Mailarchive. Cougaar's 15,000+ Agents vs. other `http://www.mail-archive.com/cougaar-developers@cougaar.org/msg00090.html`, Accessed 12.12.2008.

[2] Cougaar Software, Inc., white paper. ActiveEdge®Situational Reasoning Framework Technical Overview, `http://www.cougaarsoftware.com/files/CSI_Situational_Reasoning_Framework.pdf`, Accessed 05.12.2008.

[3] CougaarForge. `http://cougaar.org`, Accessed 05.12.2008.

[4] Crossbow Imote2 Datasheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Imote2.NET_ED_Datasheet.pdf`, Accessed 5.12.2008.

[5] IO-Link Homepage. `http://www.io-link.com`, Accessed 5.12.2008.

[6] JADE Homepage. `http://jade.tilab.com`, Accessed 5.12.2008.

[7] M. Botts, G. Percivall, C. Reed, and J. Davidson. OGC®sensor web enablement: Overview and high level architecture. *GeoSensor Networks: Second International Conference*, 2006.

[8] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco. Tinylime: Bridging mobile and sensor networks through middleware. In *Proceedings of the PERCOM 2005.*

[9] J. Gehrke and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 7:10–15, 2000.

[10] P. Grace, D. Hughes, B. Porter, G. Coulson, and G. Blair. Middleware support for dynamic reconfiguration in sensor networks, 2007.

[11] A. Helsinger, M. Thome, and T. Wright. Cougaar: A scalable, distributed multi-agent architecture. In *SMC*, 2004.

[12] K. Henricksen and R. Robinson. A survey of middleware for sensor networks: state-of-the-art and future directions. In *Proceedings of the international workshop on Middleware for sensor networks*, 2006.

[13] E. Keogh, J. Lin, and A. Fu. HOT SAX: Efficiently finding the most unusual time series subsequence. *IEEE International Conference on Data Mining*, 2005.

[14] T. Kopp. Non-probabilistic analysis of time series for detection of abnormal behaviour, Diploma Thesis, University of Marburg, Germany, 2008.

[15] A. Kröner, D. Heckmann, W. Wahlster, and K. Kogure. SPECTER: Building, exploiting, and sharing augmented memories. In *Workshop on Knowledge Sharing for Everyday Life*, 2006.

[16] K. Modukuri, S. Hariri, N. V. Chalfoun, and M. Yousif. Autonomous middleware framework for sensor networks. In *Proceedings of the International Conference on Pervasive Services*, 2005.

[17] K. Römer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communication Review*, 6, 2002.

[18] S. Salvador and P. Chan. Learning states and rules for detecting anomalies in time series. *Applied Intelligence*, 2005.

[19] M. Schmitz, J. Baus, and R. Dörr. The digital sommelier: Interacting with intelligent products. In C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, and S. E. Sarma, editors, *Internet of Things 2008*. Springer, 2008.

[20] M. Schneider. Towards a general object memory. In T. S. A. Bajart, H. Muller, editor, *UbiComp Workshop Proceedings, Innsbruck, Austria*, 2007.

[21] R. Shapiro and J. Zinky. Cougaar frameset overview, 2007. `http://cougaar.org/docman/view.php/17/197/FrameSet-Overview_Mar07.ppt`, Accessed 12.12.2008.

[22] W. Wahlster, A. Kröner, M. Schneider, and J. Baus. Sharing memories of smart products and their consumers in instrumented environments. *it - Information Technology*, 50(1), 2008.

[23] E. Yoneki and J. Bacon. Unified semantics for event correlation over time and space in hybrid network environments. In *IFIP International Conference on Cooperative Information Systems*. Springer, 2005.