Embedding Semantic Product Memories in the Web of Things

Christian Seitz and Christoph Legat
Intelligent Autonomous Systems
Siemens AG, Corporate Technology
Munich, Germany
Email: [ch.seitz | christoph.legat.ext]@siemens.com

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

***Abstract*—Today, RFID is used to identify a wide range of work pieces or individual products for tracking their movements through the logistics chain. For future purposes the idea of storing only a single ID must be extended to a Product Memory. This memory stores data of the complete product life cycle. This paper introduces an architecture and an implementation for integrating data in product memories. Our contribution encompasses software modules for a uniform sensor access, a sensor data ontology and web interfaces for product memory applications.**

I. Introduction

This paper introduces web approaches for Semantic Product Memories. A Product Memory is part of Intelligent Products and a next possible step in product identification.

A. From Price Labels to a Product Memory

Some years ago, price labels were attached to products in supermarkets. But these labels had some major drawbacks. If the prices of a product had changed a tedious manual re-labeling process was necessary. Manual activities are both, expensive and error prone. Additionally, the prices labels were not machine readable in an easy way. Sophisticated character recognition methods combined with expensive camera equipment would have been necessary.

For that reasons the price labels were replaced by bar codes. A bar code in its simplest form represents data in the widths (lines) and the spacings of parallel lines. With a bar code a specific product in a data base is associated. The benefit is, if the price of a product changes only the price attribute in the data base needs to be changed and not each single instance of a product must be updated. Bar codes are machine readable with e. g. laser scanners.

Bar codes in supermarket identify a specific product, but have still the disadvantage that single product instances cannot be addressed. Additionally, it is not possible to change the data of the bar code without replacing it. Therefore, in the last few years for expensive products the bar code technology was replaced by the RFID (Radio Frequency Identification) technology. RFID tags are readable and (re-)writable, bulk reading - reading to a set of tags at once - is also possible. With RFID individual data can be stored on a product. Currently, RFID tags are e. g. used for asset management or tracking of goods or animals. Unfortunately, the storage capacity of RFID tags is limited (currently some kilo bytes) and the data rate is also low.

Our vision, which we call Semantic Product Memory, goes even one step further and enhances the product identification with RFID. A product is equipped with an additional memory and optional sensors. This memory stores selected product interactions during the complete product-life-cycle. For instance, laptops are already equipped with acceleration sensors. A product memory could store these data for a specified time period and could be consulted by service staff for warranty issues. Additionally, the product memory is active, i. e. it can communicate with its environment or other product memories, e. g. with wireless radio technologies like ZigBee, Near Field Communication, or Bluetooth.

Getting back to the supermarket scenario with a semantic product memory, it is not necessary to manually change the price of a certain product or even product instance. The product determines its price by evaluating its own condition and by comparing this status with its environment. A prerequisite to achieve this behavior is a set of strategies which are part of the initial product memory.

B. A Road map for Product Memories

The vision of Product Memories cannot be realized in more or less sophisticated forms. The first and easiest realization is a RFID based product memory, as it already exists. A product is equipped with a RFID tag that stores product life cycle relevant data. Alternatively, a link can be stored on the RFID tag. This link can be a web resource and could be accessible with a browser.

Since the storage capacity of RFID tags is limited, the next step is a RFID based system with an additional flash memory. The content of the RFID Tag is constantly moved to a large additional memory.

RFID data throughput is low; therefore it is relevant to replace it with a faster wireless radio technology. But a new huge problem arises - the power supply. Larger batteries are necessary or energy harvesting techniques must be applied. Since product memories must interact with the environment, standards must be used for information exchange. But complex communication stacks (e. g. TCP/IP) and abstract content languages (e. g. XML) need a further increase of the processing power of the product memory hardware which is in opposition to a miniaturization, low energy consumption, and of course the costs.

978-1-4244-5328-3/10/$26.00 ©2010 IEEE

From software aspects another facet exists. The product memory can be autonomous; i. e. it interacts with other product memories and its environment. Thus, intelligent products [1] can be realized.
Our focus is an autonomous product memory. For a first prototype we use a wireless sensor node. Although, this hardware is today very costly, we believe that in about five year the price will be reduced and it will be affordable to endow products with an autonomous product memory.
The paper is organized as follows. Section II is about existing work. After this, the system architecture of our approach is discussed. This is followed by the presentation of product memory scenarios. Section V presents our implementation. The paper finishes with a future outlook.

II. Related Work

There are already some consumer applications which describe a simple RFID-based form of a product memory.
The authors in [2] 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. 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 [3] 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, and the price is manually entered into a SQL data base. The information is in turn related to the object through a globally unique identification number.
In [4] a frozen pizza box is equipped with a simple RFID-based object memory. The authors report on requirements regarding technology and memory content, and describe our memory framework implementation as well as two hardware demonstrators from the smart home domain.
The creation of product memories is discussed in [5]. The authors suggest splitting 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 data base. Wahlster et al. [6] 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.
The existing approaches are all RFID based or data base centric and do not implement autonomous product memories. Therefore, our proposal can be seen as an enhancement of existing work.

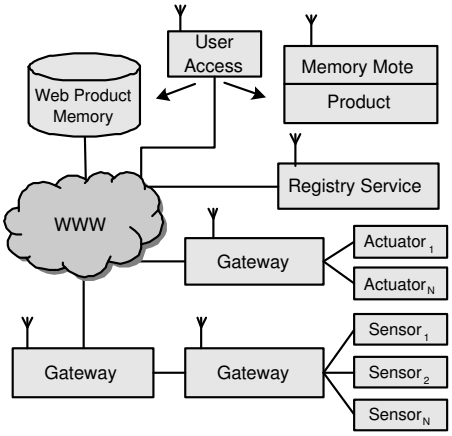

Figure 1. Semantic Product Memory System Architecture

III. System Architecture

In this section our proposed architecture is described. The major building blocks are depicted in figure 1. A product is equipped with a memory mote, i. e. the hardware which is necessary to realize semantic product memories. This hardware integrates a (wireless) communication facility in order to get in touch with the product memory. Since the product memory acts autonomously and searches actively for memory content another infrastructure element is necessary - the Gateway. It can be seen as an abstraction for wrapping heterogeneous sensing and actuator devices and providing unified access to sensor measurement, configuration and control, hiding the details of raw data acquisition and actuator control from the higher-level components. Therefore, sensors and actuators can be attached to Gateways, either for accessing data or to control the environment. Gateways are connected to other Gateways or to the Internet. Data in the Internet or data from data bases can be seen as virtual sensors. Gateways publish information about attached sensor and actuators at a Registry Service. This service is used by the product memory when certain information is needed or action must be executed. Finally, product memory data can be replicated in the Internet, just because a product memory is not accessible at any time through its mobile character. The User Access module allows the detection and access of surrounded product memories as well as the access of associated web product memories.
In the next section the architecture of the two most important components - the Memory Mote and the Gateway - are described in detail.

A. Memory Mote Architecture

Figure 2 shows the generic architecture of a single product memory. It consists of five major components: (*i*) a *Communication Interface* to send and receive information; (*ii*) an optional sensor module for accessing sensors on the product; (*iii*) a *Memory Agent* which coordinates the communication activities; (*iv*) a *Product Memory* containing

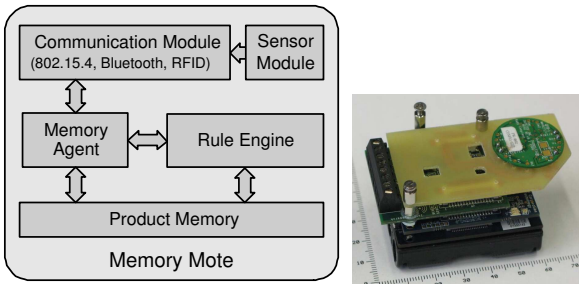

Figure 2. Generic Product Memory Architecture

all information stored on the memory; (*v*) and finally a *Rule Engine* for evaluating the stored information and inferring required actions. In the following we discuss each of the components in more detail.

1) Communication Interface: The Communication Interface is responsible for receiving data (e.g. from manufacturing machines) as well as for sending data to outside components (e.g. in order to adjust a machine or to trigger an external event). The Communication Interface has to be adapted to the applications area. We are focusing on the industry automation domain, where 802.15.4 communication, Bluetooth or RFID-based communication infrastructure is already in place. In general, a wide range of different technologies and protocols can be supported. Sending and receiving activities are controlled by the Memory Agent.

2) Sensor Module: A product memory can optionally contain onboard sensors. Products are mobile and therefore it is possible that no communication with external sensors is possible. But many applications for product memories must be constantly provided with sensor data, e. g. with temperature data. The sensor module consists basically of device drivers for the sensors.

3) Memory Agent: The Memory Agent is responsible for forwarding information to the product memory once it is received by the Communication Interface. Currently, we assume that the input data is already semantically annotated in a standardized way - this is done by the Gateway. As a language with standardized syntax and semantics we rely on a restricted fragment of the W3C Web Ontology Language OWL 2 [7], called *OWL 2 RL profile*. The major difference of the profile in terms of expressivity compared to OWL 2 DL are a limited set of supported axioms (no disjoint unions of classes, no reflexive object property axioms, no negative property assertions) as well as restrictions on the use of certain constructs (e.g. no existential quantification on the right side of an axiom allowed). Although not fully expressive, the fragment has two major advantages with respect to embedded applications: First, all standard reasoning tasks (such as checking for ontology consistency, class expression satisfiability, class expression subsumption, and instance checking) are tractable, i.e. they can be solved by a deterministic algorithm in polynomial time which is absolutely crucial for the real world embedded systems. Second, OWL 2 RL can be implemented using traditional rule-based systems and we can thus easily realize a reactive behavior (i.e. knowledge base updates may directly trigger new actions). The Memory Agent therefore has to translate the received data into rules and facts before storing them in the product memory.

4) Rule Engine: Each time a change in the Product Memory occurs, the Rule Engine verifies whether the current facts fulfill the conditions listed in the body of a rule (antecedent). This is repeated for all rules until no rule fires anymore. Note that an obvious limitation of the current OWL 2 RL translation to a forward chaining rule language is the possible loss of the declarative model. In real applications the declarative models could be important – particularly if different not coordinated sets of rules are added to the product memory by different parties.
The rule sets must be transferred a priory to the Memory Mote. The rule set is not static, i. e. it can change during the product life cycle.

B. Gateway Architecture

The Gateway represents sensors and actuators. The Gateway integrates all devices, manages them internally and represents them to the outside as one logic unit. The Gateway registers its sensors with the Registry Service (see figure 1). To enable a unified access to heterogeneous devices, the Gateway provides methods for accessing raw sensor data, for configuring individual sensor and actuator devices and for monitoring their state. The components of the Gateway can be seen in figure 3.

1) Communication: The Communication subcomponent is responsible for handling all communication of the Gateway with the "outside", i. e. other components of the system infrastructure. Thus, the Communication component provides the unified interface for sensor and actuator access and configuration. It handles multiple, potentially concurring requests and subscriptions and provides the means for

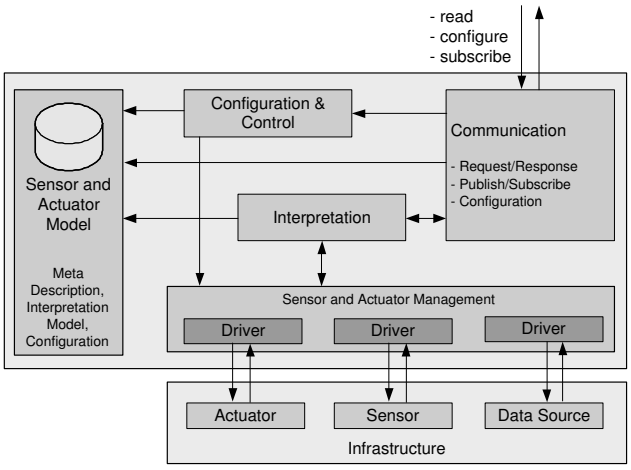

Figure 3. Architecture of the Gateway

sending messages to remote components, e. g. notification messages to registered subscribers or messages to register sensors at the registry component.

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

3) Configuration and Control: The Configuration and Control subcomponent is responsible for interpreting configuration requests of higher level entities (e. g. the product memory) and for carrying out the configuration of the underlying sensors and actuators accordingly. Additionally, this component is in charge of controlling actuators, e. g. switch on/off of machines.

4) Interpretation: The Sensor and Actuator 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 Gateway 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.

5) Sensor and Actuator Model: The communication and interface heterogeneity of sensors and actuators can be overcome by abstracting with a middleware-like architecture, the semantic and schematic heterogeneity of delivered data can not be solved in this way. The research of information integration has identified the use of ontologies as appropriate remedy to integrate different data of different sources and enables the handling of the latter two heterogeneity problems by the explication of implicit and hidden knowledge [8]. The precise and formalized nature of ontological knowledge encapsulation enables the generation of additional knowledge not explicitly given within the knowledge base which can be used directly by the product memories reducing the effort of their mostly limited computing resources.

Currently we are focusing on sensor models, the integration of actuators is planned for the near future. We use

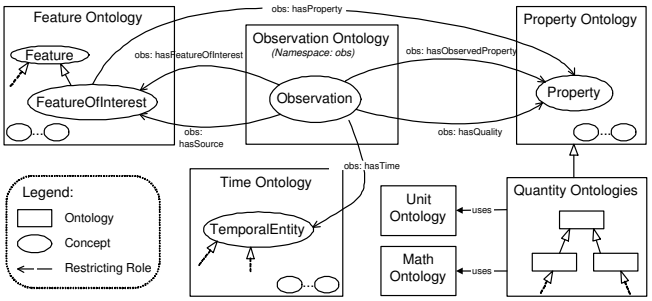

Figure 4. Structure of the upper ontology layer

an OWL DL description logic based two-layered sensor model architecture consisting of a domain-independent upper ontology layer and subjacent domain ontology layer to balance interoperability between different domains and often highly domain-specific formalization of knowledge. The upper ontology layer is modularized in a sum of ontologies of different concerns as shown in figure 4 to enable a flexible design and reducing the complexity for development and reasoning. The OWL DL logic is used for the registration service. Since the product memory is currently only able to process OWL 2 RL, the non OWL 2 RL expressions must be filtered, before it the expressions are transferred to the device, see section III-A3.

The macro-structure of the upper ontology described in the following is embossed by a simplified but expressive view on the physical world. Entities like robots, machines or the product itself are faced to their imputable aspects like temperature or speed. The observation procedure to detect this relationship is done by sensors. This circumstance can be expressed with the help of the Observation Ontology which is inspired by [9]. It forms the center of the upper ontology layer defining the *Observation* concept and its relationships. An observation relates an entity called feature of interest to one of its aspect called property by the roles *hasFeatureOfInterest* and *hasObservedProperty*. A feature of interest is a feature being the object of interest of an observation in accordance to standards. The Feature Ontology encapsulates knowledge about features and enables time dependent and independent reasoning on features with standard reasoning engines to support the detection and generation of additional knowledge. A semantically exhaustive description of properties achieved by a generic meta-model is the concern of the Property Ontology. It is expanded by a summary of ontologies for quantifiable properties (quantities) which are separated by concern of science into different ontologies in a recursive hierarchical manner. The strong mathematical grounding of quantities assisted by the Math Ontology enables their automated computing and precise distinction e.g. between speed and velocity. The ontology for physical quantities is the most important of them to describe low-level sensor information and is completed by an ontology for units of measurement

which enables reasoning on the different dimensions of units. With a view to the effort of abstracting sensors by focusing on the information provided by them, some additional information beside the description of features of interest and their properties are necessary to represent all facets of an observation: the quality, the source and the time. The qualitative information is indispensable if sensors should be abstracted because e.g. correctness, granularity or resolution of observed data are highly dependable on the sensor hardware and has to be presented within the data. This enables reasoning about inconsistencies and the detection of faulty sources as well as provides a selection criterion for data requests. Quality aspects of an observation are integrated with the help of the *hasQuality* role linking to a property because it can be seen as property of an observation and therefore can be modeled in the same way as properties of a feature of interest. The annotation of an observation with a description of the source providing the respective observation enables on the one hand to detect faulty sources. On the other hand it offers the possibility to describe processes used to compute an information e.g. by a transducer or a smart sensor node to provide the possibility to reason relationships about information correlations and enable the detection of possibly damaged sources. Sources either physical or abstract one are entities within the physical world and can therefore be modeled in the same way as any other entity as feature of interest. This is expressed by role *hasSource*.

The ontology approach supports extensibility. New concepts can easily be added and the integration of new domain ontologies is also possible.

IV. Scenarios

This section describes which scenarios can be realized with the architecture of the previous section and which we are currently implementing. Our focus is the factory automation domain. Therefore, we concentrate on adding sensor data in the product memory to improve quality management or maintenance assistance for operators.

A. Adding Data to Product Memory

In order to get the product memory filled with relevant data it must be a priori specified which data will become part of the memory. The product memory is active; i. e. the product memory agent makes requests to the environment for certain data. We use a rule based approach to specify the content of the product memory:

$$(Temperature(Product) > 25.0°) \rightarrow \\ add\ Humidity\_value(Environemnt)$$

This rule simply specifies that the product memory agent has to provide the memory with an additional humidity value. With such rules the product memory will be successively filled. Sensors attached to Gateways provide local sensor data. But Gateways can be attached to the Internet and also remote sensor data can be integrated in the product memory. This idea is inspired by the work of Sensor Web Team [10]. Finally, product memories can also be data sources for other product memories. When product memories exchange data, the Internet of Thing becomes reality. Currently, in our approach this is possible but only by using the Gateway acting as a data dispatcher.

B. Accessing Data of Product Memories

The customer must be able to read certain values of the product memory. The User Access module is in charge of this task. This module can detect product memories in the environment by sending broadcast messages. Since a product memory can also contain links to web resources, the access module needs internet access. Each product memory consists of a globally unique ID (e. g. a URI). If this ID is known, also replicated Web product memories can be accessed.

We plan to implement the User Access module on mobile devices like PDAs or mobile phones. This enables consumer a ubiquitous access to product memories.

C. Product Memory Controls the Environment

The product memory does not only collect data is also an opportunity to perform control functionality. In the production domain, with a product memory a decentralized production control becomes possible. The goal is an autonomous working station. All data that is needed to assemble the product is kept on the product in the product memory. If a product enters the vicinity of a working station the information is sent from the 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 and the used materials. The product memory can even contain program code for the producing machines. This use case of course needs a completely new infrastructure for shop floor systems. One approach is the architecture in section 1.

V. Implementation

This section describes briefly the implementation of our first prototype of a semantic product memory. Although the product memory is supposed to be used during the complete product-life-cycle, we started to create product memories in the product production step at first.

A. Memory Mote

The hardware basis of the prototype is a Crossbow Imote2 module, see figure 2, on the right-hand side. The Imote2.NET is an advanced wireless sensor node platform. It is endowed with 32MB memory, which is sufficient for a first version of a semantic product memory.

The programming environment for the Imote2 is the .NET Micro Framework. A rule engine is needed for the Memory Mote and since there is no suitable engine available for the .NET Micro Framework in managed code, we decided to

integrate the rule engine CLIPS [11] natively. CLIPS is a production rule engine written in the programming language C based on the rete algorithm. Today, it is one of the most widely used expert system tools because it is fast, efficient and free to use.

In order to improve the machine interaction with the Imote2, it was upgraded with an additional RFID module. It provides a low power, high performance, multi-protocol 13.56 MHz module. The Imote2 in combination with the RFID module is called in the following siTag (smart industrial tag).

This siTag contains the software modules which are introduced in the architecture section III-A and is supposed to be attached to valuable products. However, the product memory must be configured with rules and initial facts. Since many participants interact with the product memory during the product life cycle it is not possible to store all relevant rules initially. Some participants are not willing to hand their rules to other partners. Not to mention, an increasing amount of rules influences the execution time as well as the memory consumption in a negative way. In order to add new rules or facts, the rules must be sent to the memory mote using 802.15.4, RFID, or Bluetooth. The Memory Agent on the siTag checks if new rules are available and shifts them to the rule engine.

The rules define which data are to be integrated in the product memory. For example, the rule in section IV-A is transferred to the Memory Agent that provides the corresponding sensor data. This sensor data is semantically annotated, using OWL 2 RL and is received by the communication module. OWL 2 RL provides a standardized way (W3C) for information exchange, which is necessary for a cross domain application. The data is transformed by the memory agent in the rule language and entered in the product memory. If the new data is entered the rule engine checks if other rules can be fired and the cycle starts anew.

B. Gateway

We implemented the Gateway architecture of section III-B in Java which is installed on industrial PCs. To the Gateways various sensors are attached, e. g. light barrier, distance sensors, gas sensors, and acceleration sensors. For these sensors, device drivers must exist in the Gateway. We implemented simple drivers, e. g. for the serial interface, but also complex access systems, e. g. OPC (OLE for Process Control).

Since these sensors all return raw sensor values (analog current values 4-20 mA, or digital values from 0-4095) the corresponding interpreters are implemented. The output parameters of the sensors are part of the sensor model and are used by the interpretation component For example, a distance sensor returns distance values from 0.2m - 6m. The current interval 4-20mA is linearly mapped to the distance interval. If a similar distance sensor with another interval is used, only the sensor model must be changed. The interpreter transforms these raw values into a sensor typical unit value, e. g. meter or degree Celsius. Additionally, the data is semantically annotated. Currently, unit, quality and location information is added by using the ontology presented in section III-B5. This sensor model is registered with the registry service, which allows the definition of abstract queries of the memory mote for sensor data. For example the memory mote can request the air temperature in degree Celsius in room 12 with a precision of ± 0.1 °C. The registry service returns the ID of gateway(s) that is/are able to answer the request.

We implemented a subscription service within the communication component. With this, a memory mote is able to get sensor data in regular time intervals. The sensor information is sent from the gateway to the memory mote by using 802.15.4 or Bluetooth radio technology.

VI. Conclusion and Future Work

In this paper we presented concepts for generating and accessing product memories. A product memory provides a digital diary of the complete product life cycle.

We described an architecture for autonomous product memories, i. e. the product memory determines itself which data become part of the memory. Local environmental sensor data can be integrated as well as data from the Internet or other product memories. We implemented a first version of a product memory, using a wireless sensor node. We use the OWL DL standard to model sensor data.

For the future we will finalize the communication modules and a web service based infrastructure using DPWS is planned. Additionally, we will integrate actuators in our implementation in order to realize a product driven automation process.

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.

References

[1] G. G. Meyer, K. Främling, and J. Holmström, "Intelligent products: A survey," *Comput. Ind.*, vol. 60, no. 3.

[2] M. Schmitz, J. Baus, and R. Dörr, "The digital sommelier: Interacting with intelligent products," in *Internet of Things*, 2008.

[3] M. Schneider, "Towards a general object memory," in *UbiComp Workshop Proceedings, Innsbruck, Austria*, 2007.

[4] A. K. Michael Schneider, "The smart pizza packing: An application of object memories," in *Proceedings of the 4th International Conference on Intelligent Environnments*, 2008.

[5] A. Kröner et al., "SPECTER: Building, exploiting, and sharing augmented memories," in *Workshop on Knowledge Sharing for Everyday Life*, 2006.

[6] W. Wahlster et al., "Sharing memories of smart products and their consumers in instrumented environments," *it - Information Technology*, vol. 50, no. 1, 2008.

[7] B. Motik, P. F. Patel-Schneider, and B. Parsia, "Owl 2 web ontology language: Structural specification and functional-style syntax," http://www.w3.org/TR/2009/PR-owl2-syntax-20090922/, 2009.

[8] H. Wache et al., "Ontology-based integration of information – a survey of existing approaches," in *IJCAI-01 Workshop on Ontologies and Information Sharing*, Seattle, USA, 2001.

[9] S. Cox, "Observations and measurements – part 1 – observation schema," 2007.

[10] M. Botts et al., "OGC®sensor web enablement: Overview and high level architecture," *GeoSensor Networks: Second International Conference*, 2006.

[11] G. Riley, "Clips: An expert system building tool," in *In Proceedings of the Technology 2001 Conference*, 1991.