First Steps Towards an Event-Based Infrastructure for Smart Things

Marc Langheinrich, Friedemann Mattern, Kay Römer, Harald Vogt

Distributed System Group, Department of Computer Science
ETH Zurich, Swiss Federal Institute of Technology
8092 Zurich, Switzerland
{langhein,mattern,roemer,vogt}@inf.ethz.ch

ABSTRACT

In this paper we examine requirements for an infrastructure that supports implementation and deployment of smart things in the real world. We describe a case study (*RFID Chef*) where kitchen items and ingredients, equipped with remotely accessible electronic tags, drive an interactive context-aware recipe finder through the use of an event-based infrastructure.

Keywords

Ubiquitous computing, RFID tags, events, infrastructure.

INTRODUCTION

The emerging field of *Ubiquitous Computing* [4] aims at making computers available throughout the environment, while rendering them effectively invisible to the user. One of the main goals is to incorporate computing power into everyday objects in order to create *smart things:* real-world objects that provide novel ways of accessing information, react to their environment, or provide new emergent functionality when interacting with other smart things.

Research efforts in this field can roughly be categorized along three axes, as depicted in figure 1: Diversity, scale, and interaction complexity. Research projects so far have usually traded diversity (i.e., types of artifacts) for scale and vice versa, effectively limiting the potential interaction complexity of the overall system to simple one-to-one interactions.

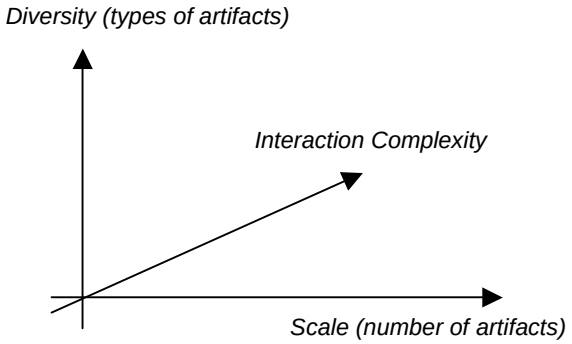

Figure 1: Applications involving s*mart things* can be characterized along three axes: Diversity (the number of different types of artifacts supported), scale (the number of instances of each artifact type), and interaction complexity (the number of different modes of inter-artifact and cross-artifact interactions).

Truly *smart environments* that feature a large number of smart things will require both large scale and high diversity, thus creating much richer interaction schemes, where many-to-many interactions both between artifacts of the same type as well as between artifacts of different types need to be addressed.

So far, research projects typically used ad-hoc approaches when enabling smart things to interact with their environment. However, with the increase in interaction complexity, the need for a well-founded *interaction infrastructure* arises, which will support much richer interaction models.

In the following sections, we want to examine requirements for such an interaction infrastructure, propose one solution based on a hierarchical event model, and describe a sample application that we use to examine our design in a practical setting.

INFRASTRUCTURES FOR SMART THINGS

Smart things need - mostly independent of concrete application scenarios - several supporting services (e.g., they typically want to interact with the environment or with other smart things in their proximity). Also, facts about the environment smart things perceive have to be represented in such a way that the information can easily be processed and made available to other smart things (perhaps to modify their own behavior). This, and many other similar tasks, should not be done by the things themselves, but by a supporting *infrastructure* residing in the environment.

Virtual Counterparts of Real Objects

One observes that more and more objects of the real world are represented in databases or are mirrored in World Wide Web (WWW) servers, which are - at least in principle - linked together by the Internet. Thus, a large and fast growing information space - a “one virtual world” - begins to emerge that contains various information items about real-world objects and in some sense reflects the real world. Such a global information space about “almost all” everyday objects seems to promise very interesting new opportunities. Unfortunately, however, this *virtual world* is largely unstructured, and it is difficult or often impossible to directly link an artifact with its corresponding information items in the virtual world.

This would be simpler if each real-world object had its dedicated representation in the virtual world, and if these virtual counterparts were organized such that they reflect in some sensible way the structure and relations of the real-world objects they stand for (fig. 2). The virtual counterparts could be WWW pages (as it is done in the *Cooltown* project [5]), thus associating each artifact with its homepage, or any other data items giving things some electronic identity.

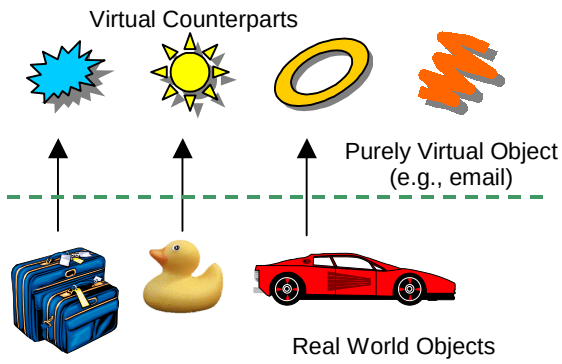

Figure 2: Real-world artifacts are linked through the infrastructure with their *virtual counterparts*, enabling them to interact in the virtual world. Note that not all objects in the virtual world necessarily have a corresponding object in the real world.

For our ubiquitous computing infrastructure for smart things we postulate that the virtual counterparts are organized along the *object-oriented paradigm* - that is, virtual objects may be structured into classes, and objects interact by

sending messages to each other which may evoke state-changing methods at the receiving side. The clear advantage of using an object-oriented representation model is that relevant aspects of the real world can easily be modeled - in some sense, the virtual world *simulates* the real world more or less directly with respect to relevant actions (Note that object orientation was in fact invented with the language *Simula* some 35 years ago to facilitate modeling and event-based simulation of real-world situations!).

Of course, the physical world of artifacts and the virtual world populated by abstract objects have to be linked together. We deliberately chose to concentrate on *events* as the connecting entities gluing the two worlds together: Relevant state changes in the real world have to be detected and signaled to the virtual world by the generation of appropriate events. Events will then trigger the virtual counterparts to simulate the state change.

To automatically detect state changes in the physical world, one needs sensors. However, low-level sensory information is often not directly useful. Typically, raw sensory input is therefore processed and combined to form higher-level events that model real-world actions and thus need to be modeled accordingly in virtual space.

Smart Things and Their Virtual Proxies

How do everyday things become smart, that is, get an electronic identity and some information processing functionality in addition to their classical, mundane function? One possibility is to equip them with embedded processors, memory, sensors and possibly some output device. However, often it is sufficient to put the smartness into a virtual "proxy" object that resides somewhere in the virtual world - maybe on some Internet server that has an abundance of storage and processing capabilities.

Also, instead of being integrated into the things themselves, sensors may reside in the environment. These sensors then have to detect the presence of objects, and perhaps also any changes in their state, and communicate this to the corresponding virtual counterparts in the virtual world. For that, it might be sufficient to tag each real-world thing with a remotely readable label that represents an URL-like pointer. Via that pointer not only an almost infinite amount of information may be associated to the object, but also potentially demanding *services* which are then in fact realized by the virtual proxy object.

Clearly, merely communicating events that signal the presence of things to the virtual world is not sufficient in all cases. Sometimes it might be necessary to connect a data stream from the real-world object's sensor to an information processing process residing in its virtual proxy. Our model does not preclude, but also not support such data stream linking.

Figure 3: Typical form factors for RFID tags. The particular tags shown come in a glass casing in order to allow attaching them to metallic objects or to insert them below the skin of animals. (Image Source: Portolano Project [13])

Instead, we concentrated on a pure event propagating architecture and used electronic labels (i.e., *radio frequency identification or RFID tags*) attached to the objects in our prototype implementation.

RFID tags feature a small integrated circuit (IC) connected to an antenna and are capable of storing data (usually a numeric ID) in the IC's memory. Tags can be read (and sometimes written) using a magnetic or electromagnetic field emitted by a fixed-position or mobile reader device over distances ranging from several centimeters up to a few meters. Since RFID tags collect the energy to operate from the field emitted by the reader device, they don't require an internal power source and can be produced both with very small form factor and at a very low price [6].

Attaching RFID tags to physical objects allows linking them to arbitrary online information, very much in the same manner as traditional printed barcodes allow suppliers and vendors to track production, inventory, and sales of goods. However, since RFID tags can be made much smaller than barcode labels, and because no line of sight is required for reading (the magnetic and electromagnetic fields only require relative proximity), such links can be integrated into everyday objects in a much more unobtrusive manner.

The Necessity of an Infrastructure

While electronic tags create novel ways of linking everyday objects with online information, most approaches have so far been limited to rather simple scenarios: a single tag is brought into the vicinity of a tag reader and triggers some sort of action on a server. As we begin to imagine densely tagged environments where hundreds or thousands of such objects are present and move in and out of the vicinity of dozens of tag sensors (both mobile and fixed-position), we need to move away from a simple action-response model of early prototypes towards a much richer representation.

We are convinced that our object-oriented model of virtual entities allows the realization of more complex event-based information structures that can effectively represent such situations. For example, virtual objects may filter or combine events, and generate new "super events" that are then propagated to other virtual objects. It is easily possible to implement virtual objects that have no direct counterpart in the real world, but provide nevertheless some very useful function within the virtual world (e.g., event clustering), thus realizing and incorporating some abstract *concept*. Virtual proxies of real artifacts may interact and form relationships (e.g., when the corresponding real-world artifacts become related in some concrete or abstract sense), and even *pure virtual objects* may be generated dynamically to represent new abstract facts. Furthermore, virtual entities, such as Web pages, email messages or word documents are easily integrated in such an architecture.

However, an infrastructure for smart things should not only consist of an architecture to represent objects and events, but also provide various *services*. Smart things (or their virtual proxies) may need location information, they want to discover services in their physical proximity, and they may want to communicate to other (possibly remote) physical objects. While up to now we only realized some basic services for event handling, we postulate that an infrastructure for future populations of smart things requires such services. Here again we think that an object-oriented virtual

world that mirrors the physical world with respect to relevant events is well suited for such services.

Beyond the concepts mentioned here and our current (and still rather initial) realization of an infrastructure for smart things, there are other interesting challenges - both on the conceptual level and on a more pragmatic implementation level. For example, it is not a priori clear what a single object in the real world should be: Two objects might be bound together (and have two RFID tags) such that they might also appear as a single object - this is an example of a modeling issue one has to deal with. Dynamic creation and mobility of artifacts together with the geographic distribution of the supporting infrastructure gives rise to a different set of problems and design issues on the implementation level (e.g., should the virtual proxy of a mobile artifact also be mobile and follow the artifact to another infrastructure domain?). Finally, considering time as another dimension where local proximity matters yields even more challenging issues.

However, as our case study described further down shows, even a rather small subset of an ideal infrastructure for smart things is already very helpful.

The Role of Event-Based Systems

We consider events, and services associated with events, to be of prime importance for an infrastructure for smart things. An event is a message that is generated by an *event source* and sent to one or more *event consumers*. The arrival of an event at an event consumer triggers some action in response to that event. Associated with an event may be arbitrary data, called *event arguments*.

An event-based system is characterized by the set of possible events and the argument data domain. One can differentiate amongst untyped and typed event spaces. For example, an untyped approach is used in the *Jini* system [19] in order to guarantee interoperability, while the CORBA event service [17] supports both typed and untyped spaces.

Usually, a fixed set of basic events is identified. These events are directly or indirectly generated by some sensory hardware (e.g, the appearance of an RFID tag, as in our case study example). However, the most interesting events are aggregations of one or more basic (or other aggregated) events. Such “combined” events are often called *context events*. Since they play such an important role, event-based systems have to provide a means to describe context events by providing direct access to basic events and operations for the description of aggregated events. Examples are the event translation language contained in the Xt programming toolkit [18] and the simple language for describing weather conditions within the Tacoma weather alarm system [16].

In an infrastructure for ubiquitous computing environments, several features for describing events could be helpful. This includes multi-level context events (where context events are part of the description of other events), event sequences, access to event arguments, and extensions of the event space. Furthermore, continuous context information such as time or geographic location could be incorporated within event descriptions.

An important issue in an event-based system is the mode of addressing event consumers. There is a wide variety of possibilities to choose from and it is largely dependant on the kind of applications which modes should be supported. For example, within the X11 windowing system, a hierarchical distribution scheme is used, while in the Jini system event consumers can register for arbitrary events. Adequate mechanisms for large populations of smart things have yet to be identified, but proximity-based and type specific addressing schemes seem to be useful.

Difficulties arise with the remote distribution of events, in particular in dynamic contexts and wireless environments that are typical for populations of smart things. Generally, in-order delivery is very costly and is justified only by certain applications. In particular, the event distribution mechanism has to deal with partial network failures or temporarily unavailable event consumers. It is therefore desirable that event consumers can specify how events destined for them are handled.

RFID CHEF – A CASE STUDY

In order to examine some of the issues described above, we designed and implemented a scalable case study, called *RFID Chef*. Though our primary focus is on gaining experimental insights by providing an easily accessible scenario, we also tried to position the project as being part of a realistic environment in the not too distant future.

RFID Chef is situated in the ordinary household kitchen, where it attempts to facilitate and improve the everyday tasks of supply keeping and cooking. We anticipate the kitchen environment to be a fertile exploration setting for the following reasons:

- **Non-technical environment:** In contrast to classical office environments where people have long since accepted the presence of technology, a kitchen is a social space where so far manual labor has played a dominant role. In order to be accepted, technology needs to be combined with tangible interfaces [7] such as knives, pots and pans, or counter tops.
- **Variety of artifacts:** A kitchen offers a large diversity in artifacts, both in the real world (e.g., ingredients, cooking utensils, appliances, and last not least,

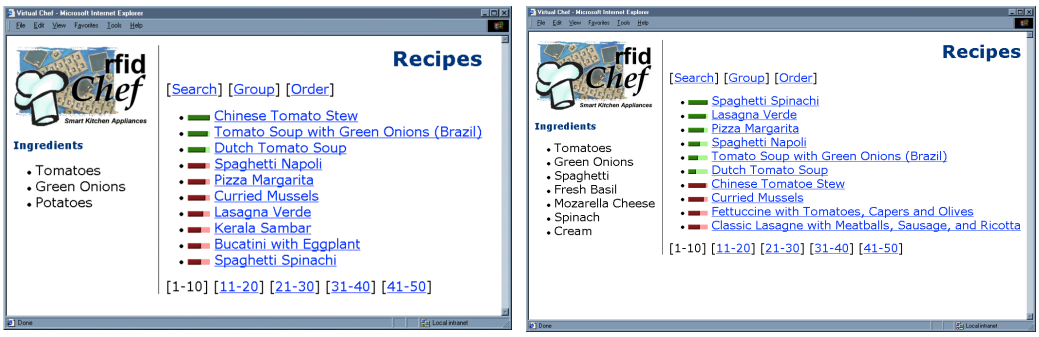

Figure 4: Sample screenshot of the *RFID Chef* prototype implementation. On the left side, three ingredients have been placed on the counter: tomatoes, green onions and potatoes. The system suggests a Chinese tomato stew. After the potatoes have been removed and five additional ingredients have been put on the counter, the recipe list dynamically adapted to suggest Spaghetti Spinachi instead.

people) and as purely virtual entities (such as recipes).

- **Large number of artifacts:** Each type of artifact described above comes in relatively large number of individual items, such as different spices, various sized pots & pans, the individual members of a family, or even different guests.
- **High interaction complexity:** People use a number of ingredients in order to prepare a particular recipe, employing different tools and appliances in the process.

In short, a kitchen should offer the complexity of interaction patterns that we are interested in, while at the same time being a challenging environment for incorporating technology into everyday objects. For similar reasons, Minar et. al. started work on a project called *Counter Intelligence* [12].

In the following paragraphs, we will first describe the current state of our prototype system, and then continue with a description of its underlying software architecture. A brief overview on the utilized hardware and software concludes the description of our case study.

Prototype Description

In our first implementation step, we have targeted the kitchen counter and extended it in order to support the user in selecting a recipe given the list of ingredients at hand.

The basic interaction pattern is as follows: The user places grocery items (equipped with RFID tags) such as cheese or tomatos onto the kitchen counter. Items can be placed individually or in groups, e.g., in a shopping bag. An LCD screen next to the counter dynamically displays the current list of ingredients placed on the counter and updates a recipe list in order to show dishes that can be prepared with the ingredients present. A picture of the setup in our lab is shown in figure 5, a more detailed screenshot of the display can be seen in figure 4.

Figure 5: Experimental setup of RFID Chef in our lab. A standard office desk functions as our kitchen counter, an 18" LCD monitor acts as our display. Several individual ingredients have been placed on the table, where an RFID antenna (here shown in its casing, next to the ingredients) detects them and reports their presence to a PC that controls the display. Note that the antenna is usually mounted underneath the table.

Red and green bars act as visual aids to the user in order to show how "close" a number of certain ingredients are to the list of required ingredients for a recipe. Red bars indicate that the recipe requires further ingredients not yet placed on the counter. The larger the filling of the red bar, the closer the recipe is to the available ingredients. Once all required ingredients for a recipe are available, its bar turns to green and its semantics change

from “how many ingredients are missing” to “how many ingredients are not needed for this recipe.” A perfect match of ingredients is thus shown as a full green bar (compare with the top entries in the pictures in figure 4). If, however, one or more of the available ingredients are not needed for a recipe, the green bar is shortened proportionally.

Recipes are ordered according to the color of their bar (green bars sorted first, followed by red bars) and the size of the bar (closer matches first). Using the mouse as a pointing device, a user can click on an interesting recipe and call up the recipe homepage for detailed cooking instructions.

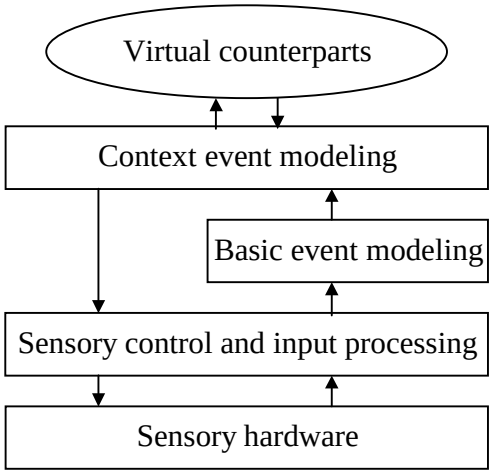

Figure 6*:* Layerd event-based architecture. After processing sensor input from the hardware, basic events are created and passed to the context event level, where events are held back, aggregated or released depending on context information.

Software Architecture Design

The main task of the RFID Chef software infrastructure is to provide a model for supporting the large variety of interaction styles mentioned in the first section of this paper. Both the flexibility as well as the efficiency of the system should be as high as possible. To that extent, we have implemented a layered event architecture that feeds sensory data into higher-level event structures, which can then be exploited by a simple script.

Being in its initial stages, it represents only a fraction of an ideal infrastructure as discussed in the previous sections. However, it should serve as a good sample environment for validating some of our infrastructure assumptions.

At the lowest level of the RFID Chef system, the *Sensory Control- and Input-Processing Module* polls sensors continuously. In the current prototype, only a single sensor exists – the RFID reader. The same module is also responsible for sensory hardware configuration and control, which can in turn be affected by higher-level context events. The module outputs the status of the RFID reader, including any RFID tags encountered in the vicinity (i.e., their ID’s), at discrete time intervals to the Basic Event Modeling Module layered on top.

The *Basic Event Modeling Module* is responsible for creating basic events that can be computed directly from sensor input, such as the appearance of an artifact inside the sensor range. For that purpose, the module keeps a short-term record of identified tags and compares the list of currently detected tags with those detected in the previous time interval. Should the two sets differ, corresponding appearance events and disappearance events are triggered and propagated to the Context Event Modeling Module.

The *Context Event Modeling Module* processes basic events and filters or collates them. In our current implementation, the module is also used for *sensor dampening*: it collates several basic events over time and releases them grouped at a higher granularity. This way, the display avoids the problem of *sensor jitter,* where the addition of several ingredients to the counter would result in the repeated flickering on the screen as the recipe list would be reordered for each additional ingredient. With sensor dampening, only a single event is triggered after the ingredients list has stabilized again, resulting in only a single update of the screen. Similarly, events that signal the *disappearance* of an ingredient are held back for a short amount of time in the context module, in order to wait for an *appearance*-event of the same ingredient a short time later. Often the rearrangement of ingredients on the counter will result in brief periods of time where the reader cannot detect a certain tag (mainly due to the collision avoidance algorithm of the underlying RFID tag identification system). By withholding a disappearance event for only a short time, the context module avoids unnecessary updates of the recipe list.

Implementation Status

In our initial prototype of *RFID Chef,* we did not focus on proper, unobtrusive integration with a regular kitchen environment. As our primary goal is to study and identify software infrastructure issues, our prototype relies more on the imagination of the user in order to pass as a productive kitchen environment.

Hardware Setup

The first generation of *RFID Chef* consists of an RFID reader that is connected using a serial cable to a standard PC. The RFID reader's antenna is mounted underneath a table. The output device is an 18" LCD monitor, a mouse allows for navigation of the recipe display. Figure 7 depicts our hardware setup.

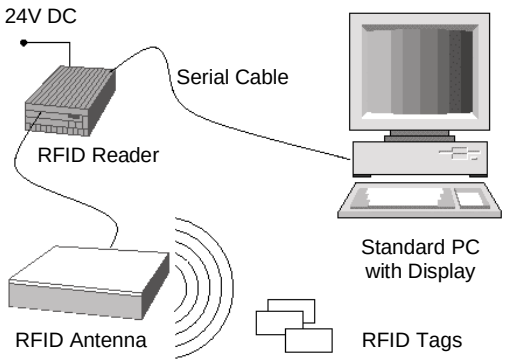

Figure 7: Schematic view of the RFID Chef hardware. The RFID reader antenna detects tagged artifacts in its vicinity and transmits raw sensory information through a reader module to the PC via serial cable.

In an initial registration step, each individual ingredient is tagged using a unique RFID label. In all cases, such a label is placed in the position where the product's barcode is currently located, simulating a future stage of packaging where barcode labels are shipped with RFID tags underneath. Fruits and vegetables are assumed to either come prepackaged or are packaged, weighted, and labeled (using a self-adhesive printed barcode label), which is usually done by either the consumer or the salesperson in the store. In both cases the items thus feature a barcode label (or, in our case, an RFID label).

The RFID infrastructure is based on Philip's *I-Code* tag system [15], which offers 64 bits of read-only memory as well as 384 bits of user-definable read-write memory per

tag. Tags come in two form factors: as a 80x50mm self-adhesive label, and as a rectangular, 55x55mm plastic wafer (shown in figure 8). The I-Code reader device operates at a frequency of 13.56 MHz and is able to read or write up to 30 tags at once using a (proprietary) anticollision protocol. The reader field easily penetrates the (non-metallic) table and is able to read and write tags within a 1x1m area on the table.

Software Status

The software is currently implemented partially in C (sensor library), Java (event library) and Python (application scripting).

The Python script features a small database of recipes, including a detailed, searchable list of ingredients. The script registers its interest in changes in the set of ingredients (including a resolution time in seconds) and reacts to any state change with the display of an updated recipe list (expressed as an HTML page). If the user clicks on any recipe displayed in the list, the script displays the recipe details and sends a context event to the Context Event Modeling Module, informing it about the mode change. Once the user switches back to the list of recipes, a corresponding context event signals the return to the recipe list mode to the context module.

The recipe database currently contains about 50 recipes, which were assembled from various Internet sources by manually extracting their list of ingredients and entering them into our database. The system is operational and has already been used in various public demonstrations.

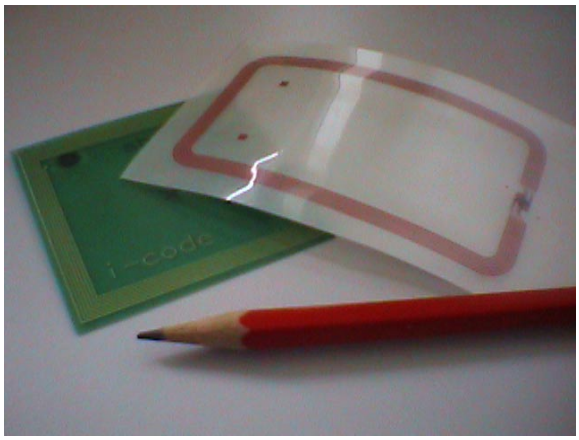

Figure 8: The two different form factors of I-Code RFID labels that are used in our RFID Chef testbed environment.

RELATED WORK

The use of RFID tags as an enabling technology for ubiquitous computing has already been explored in some other research projects. For example, Barrett et al. report creating "virtual floppies" – small disk-like objects that can be associated with files and folders on a computer [1]. Want et al. describe a number of scenarios where RFID tags call up additional information on-screen (such as displaying a person's homepage when holding his or her business card in front of the computer), or trigger automated translation scripts [2]. Minar et al. similarly use RFID tags to create a jukebox, where poker chips dropped on a table trigger playing of an associated song [3]. The coupling of real and virtual objects is also considered by Ljungstrand et al. [20] in their *WebStickers* project where they use barcode labels (instead of RFID tags) to associate a Web page with a physical object. As pointed out in the introduction, most of these projects only investigate novel ways of interactions, rather than addressing requirements for a large-scale deployment of such tags.

Related to our system is also the *Cooltown* project [5], where a Web page is attributed not only to people and places, but also to arbitrary things. The project focuses on

an infrastructure for the Web presence of things, which encompasses device discovery, location awareness, nomadicity support, and general service access. The Cooltown project is largely based on the established Web infrastructure (e.g., URLs, http, Web servers, Web pages) in contrast to our system, which advocates a more general event-based and object-oriented representation of the virtual counterparts of real things. The *Portolano* project at the University of Washington [13] also plans to investigate "architectures for proxies to handle computationally limited, mobile devices" [14]. However, so far no results have been published.

The *Counter Intelligence* project [8,12] of the *Things That Think (TTT)* Consortium [11] at the MIT Media Lab comes closest to our work, in particular with respect to the kitchen environment scenario. The system lets the user manually choose a recipe and then offers step-by-step instructions as it observes the user during preparation. While we have so far largely ignored user interface issues in our scenario and instead focused more on infrastructure and architectural, large-scale design, much of the work in Counter Intelligence has centered on the interaction of the user with the system, such as the suitability of voice output for the recipe preparation instructions.

An infrastructure called *Hive* [3, 9] forms the basis for each of the TTT prototypes. Hive has its origins in a distributed agent system created for distributed Internet applications [10] and thus focuses more on the distribution of a particular application between several mobile agents. Our approach, on the other hand, tries to provide an efficient virtual representation for representing real-world interactions. Also, while we view events and event-hierarchies as an essential building block for effective interaction modeling, Hive uses agent-based messaging.

FUTURE WORK

Our initial RFID prototype has been a first step. Clearly, we now face the task of extending our system to support more complex scenarios and then gradually refine its underlying infrastructure. With the introduction of personalized recipe selections, we will need to introduce multiple types of artifacts. Following recipe selection, an interactive preparation mode would need more complex event support, both in terms of inter-artifact interaction and time-based event handling. Lastly, extending the system towards other tasks in the kitchen such as coffee brewing or reading the newspaper would require us to fully address the requirements of a virtual proxy architecture including mapping and mobility. The ultimate goal of the prototype is to gain experience with experimental infrastructure components for populations of cooperating smart things and to learn about the general requirements of such an infrastructure.

CONCLUSIONS

In the attempt to transform real-world artifacts into smart things that provide easier access to information, facilitate tasks, and interact with other smart things, RFID tag technology presents a valuable alternative to embedding full-featured computing devices. However, in order to scale beyond trivial sample applications, a more sophisticated approach is needed both in terms of object representation and interaction modeling.

In our paper, we outlined the approach taken by our group, where we envision treating both active computing devices and tagged artifacts as first class objects having a

corresponding virtual counterpart in a networked computing environment. Such counterparts would not only serve as a data repository for fixed and variable information about an object, but would also facilitate *active interaction* between various artifacts that would otherwise be limited to short-lived request-response cycles, always cut short by the limited time a tagged object remains within the reading range of an RFID reader.

As a first step we have begun implementing a hierarchical event infrastructure, which supports these virtual objects by providing an efficient framework for *abstraction* between the real world and its virtual representation. Raw sensory input is clustered into basic events, which in turn can be the basis for complex and self-referring *context events*. Both multi-event triggers as well as triggers over time will form the foundation for helping the virtual representation keeping track of relevant events in the real world.

REFERENCES

1. Barrett, E., and Maglio, P.: Informative Things: How to Attach Information to the Real World. In *Proceedings of UIST '98*, pp. 81-88.
2. Want, R., Fishkin, K., Gujar, A. and Harrison, B.: Bridging Physical and Virtual Worlds with Electronic Tags. In *Proceedings of CHI '99* (Pittsburgh PA, May 1999), ACM Press, pp. 370-377.
3. Minar, N., Gray, M., Roup, O., Krikorian, E., and Maes, P.: Hive: Distributed Agents for Networking Things. In *Proceedings of ASA/MA '99* (Palm Springs CA, August 1999), IEEE Press, pp. 118-129.
4. Weiser, M.: The Computer for the Twenty-First Century. *Scientific American*, September 1991, pp. 94-101.
5. Kindberg, T. et al.: People, Places and Things: Web Presence for the Real World. HP Laboratories Technical Report HPL-2000-16.
6. Finkenzeller, K., and Waddington, R.: RFID Handbook: Radio-Frequency Identification Fundamentals and Applications. John Wiley & Sons, October 1999.
7. Ishii, H., and Ullmer, B. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *Proceedings of CHI '97* (Atlanta GA, March 1997), ACM Press, pp. 234-241.
8. Kaye, J., Marsakis, N., Gray, M., Wheeler, A., and Hawley, M.: PC Dinners, Mr. Java and Counter Intelligence: Prototyping Smart Appliances for the Kitchen. Accessible online at http://www.media.mit.edu/~jofish/ieee.paper/ieee.cga.jofish.htm
9. Roup, O.: Hive: A Software Infrastructure for Things That Think. Master's Thesis, MIT Media Lab, May 1999.
10. Minar, N.: Designing an Ecology of Distributed Agents. Master's Thesis, MIT Media Lab, September 1998.
11. Things That Think Consortium homepage at http://www.media.mit.edu/ttt/
12. Counter Intelligence Project homepage at http://www.media.mit.edu/ci/

13. Portolano Project homepage at
http://www.cs.wasington.edu/research/portolano/
14. Esler, M., Hightower, J., Anderson, T., and Borriello, G. Next Century Challenges: Data-Centric Networking for Invisible Computing, in *Proceedings of Mobicom '99* (Seattle WA, August 1999).
15. The Philips I-Code System homepage in /identification/products/icode/ at http://www-us2.semiconductors.philips.com
16. Jacobsen, K., and Johansen, D.: Ubiquitous Devices United: Enabling Distributed Computing Through Mobile Code. In *Proceedings of the Symposium on Applied Computing (ACMSAC'99)*, February 1999.
17. Object Management Group. CORBA Services: Common Object Services Specification. 1998. Available at
ftp://www.omg.org/pub/docs/formal/98-12-09.pdf
18. Nye, A., and O'Reilly, T.: X Toolkit Intrinsics Programming Manual. O'Reilly & Associates, Inc, 1993.
19. Edwards, K.: Core Jini. Prentice Hall, 1999.
20. Ljungstrand, P., Redström, J., Holmquist, L.E.: WebStickers - Using Physical Tokens to Access, Manage and Share Bookmarks to the Web. DARE 2000 (Designing Augmented Reality Environments) Elsinore, Denmark, April 12-14, 2000.