Infrastructure for Virtual Counterparts of Real World Objects

Kay Römer, Friedemann Mattern, Thomas Dübendorfer, Jürg Senn

Department of Computer Science
ETH Zurich
8092 Zurich, Switzerland

Abstract. Based on our experience with a collection of prototypical ubiquitous computing applications, we have identified a number of common basic tasks which led to the design of some simple mechanisms that we have found useful for structuring and implementing such applications. Building upon these mechanisms, we have created a software infrastructure to support the development of tag-based ubiquitous computing applications. Our framework is targeted at applications where objects are tagged with Radio Frequency Identification tags (RFIDs), a simple but powerful means of bridging the gap between the physical and the virtual world. A central feature of our infrastructure are virtual counterparts. They form augmented representations of real world objects and encapsulate their state and behavior. This paper presents the rationale, design, and implementation of our prototype infrastructure.

Keywords: ubiquitous computing, infrastructure, virtual counterparts, RFID, radio tags

1 Introduction

Recent technological advances have enabled parts of our world to be turned into so-called smart environments, where the physical environment is augmented in an unobtrusive way with useful IT functionality [20], thus heralding the vision of ubiquitous computing [21].

Radio Frequency Identification (RFID) systems are a technique for bridging the gap between the physical and the virtual world [18]. By attaching small passive *RFID tags* to physical objects, these objects can be identified when brought into the vicinity of an antenna connected to a device known as a *tag reader*. State-of-the-art RFID systems such as Icode [25] allow the simultaneous detection of multiple tags within a space of up to approximately one cubic meter. Tags not only hold a unique ID, but also provide a limited amount (typically about 60 bytes) of non-volatile read/write memory.

Such RFID systems enable the implementation of a wide range of novel ubiquitous computing applications. During the last two years we have developed a number of such RFID-based applications. Our first prototype systems were implemented from scratch, the only piece of software they had in common being the driver software for the RFID system. Based on our experience with these applications, we identified a number of tasks common to this type of application, which led to the design of some simple

generic mechanisms which we found useful for structuring and implementing applications using tagged physical objects. Based on those mechanisms we designed and implemented a software infrastructure to support the development of tag-based ubiquitous computing applications. As a proof of concept we re-implemented one of the earlier applications using this infrastructure.

In this paper we present our software infrastructure. We begin with a short overview of some of the applications we developed, go on by pointing out common tasks and pertinent support mechanisms, and then present some details of the infrastructure. In-depth descriptions of the work presented in this paper can be found in the theses [3] and [16].

There are a number of other research projects that aim to provide a software infrastructure for smart environments. Many of these projects such as Nexus [5] try to provide solutions for a very broad class of applications. Others grew out of infrastructures for more traditional distributed systems, such as Hive [10], which is essentially a mobile agent system. Moreover, many infrastructure approaches are motivated by mere speculation as to what infrastructure support an application might need. These issues typically make it difficult to develop applications based on such infrastructures, because the mechanisms provided either do not properly fit the needs of a real application, or are too generic to be used without much adaptation effort in real applications. Our experience with CORBA-based systems [22] shows that there is a trade-off between genericity and usability, which often causes developers to avoid a "bloated beast" infrastructure by re-implementing the required functionality from scratch.

We tried to avoid these pitfalls with our infrastructure by providing support for a rather focused set of Ubicomp applications, based on our experience with prototypical, but concrete applications.

2 Selected Ubicomp Applications

In this section we outline the type of applications we intend to support with our infrastructure by sketching some of the RFID-based ubiquitous computing applications we have developed over the recent years. Note that all the applications are based on multiple interacting tagged physical objects. These applications will serve as a basis for identifying common tasks that should be supported by a generic Ubicomp infrastructure.

RFID Chef Grocery items are equipped with RFID tags (instead of the barcodes that are commonly used today). When placed on the kitchen counter, a nearby display suggests dishes that can be prepared with the grocery items available, or shows missing ingredients. The suggested dishes not only depend on the available ingredients, but also on the preferences of the cook, who might for example prefer vegetarian or Asian dishes. To implement this functionality, the cook is identified by an RFID tag attached to his wristwatch, such that the tag enters the range of the kitchen counter RFID antenna when grocery items are placed on the counter. Figure 1 shows a screen-shot of the user interface. [8] contains a detailed description.

Fig. 1. RFID Chef user interface

Smart Playing Cards Ordinary playing cards are equipped with RFID tags. An RFID antenna mounted beneath a table monitors the game moves of the players. A nearby display calculates the score, determines the winner, displays a cheat alarm if one of the players does not follow suit, and gives hints to beginners by assessing the players' moves. This is implemented by having each card remember the contexts in which it has been used and whether the trick in question was won or lost. Figure 2 shows a screen-shot of the user interface. [12] contains a detailed description of the system.

Smart Agenda Agendas are equipped with RFID tags. If two or more people want to make an appointment, they place their agendas on the "appointment table", which is equipped with an RFID antenna. A nearby display shows possible dates that are compatible with the schedules of all the participants.

Smart Tool Box Tools are equipped with RFID tags, and the tool box contains a mobile RFID system. The tool box issues a warning if a worker attempts to leave the building site (or a sensitive maintenance area such as an airplane) while any tools are missing from his box. The box also monitors how often and for how long tools have been in use. Based on this information, tools can be replaced before they wear out. Additionally, the tool owner can charge for tool rental based on actual tool usage.

3 Common Tasks

Based on the applications sketched in Section 2 we can highlight a number of tasks common to these applications.

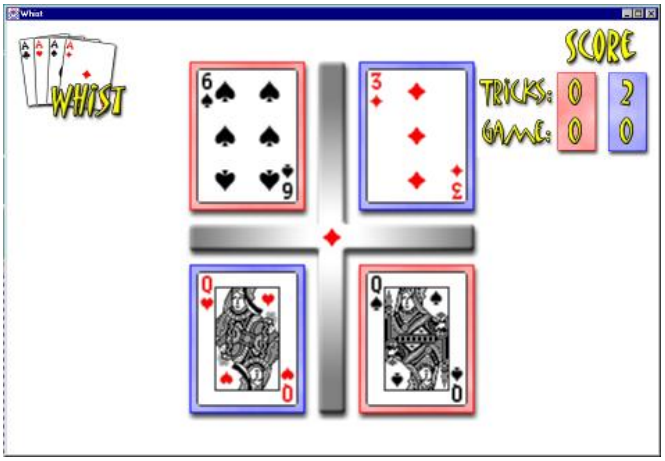

Fig. 2. Smart Playing Cards user interface

Events An RFID system can be seen as a special type of sensor which detects two types of real world events: a tag X entering the range of the RFID antenna (`enter(X)`), and a tag X leaving the range of the RFID antenna (`leave(X)`). There is usually a need to notify an application about these real world events, as is the case for all applications in Section 2. Also, applications need a way of expressing their interest in a subset of all possible tags, since a single antenna might be used by multiple applications at the same time.

Note that the RFID system and the application may run on different systems and platforms, as for example in the Smart Tool Box application, which consists of a mobile RFID system in the tool box cooperating with a stationary system at the workshop.

Event Generation Although from an abstract point of view the RFID system generates enter and leave events, matters are complicated by the actual low-level interface provided by the RFID system. The Icode system [25] for example periodically scans (typically at sub-second intervals) for present tags by sending a short RF pulse and waiting for answers from the tags. When receiving the pulse, a tag waits a random number of discrete time slots before answering, in order to avoid time-consuming collisions with other tags sending concurrently. The maximum number of time-slots N a tag may wait before answering influences both the time needed for a single scan and the expected number of collisions. A small N value results in fast scans (down to 60ms according to [17]) but many collisions, whereas a large N value results in slow scans (more than one second) but few collisions. The optimum performance can be achieved by selecting N to be slightly greater than the actual number M of tags in the range of the antenna. However, M is typically unknown. Therefore, nontrivial algorithms – which are not part of the driver software provided by the RFID vendor – are required for reading all present tags in a minimal amount of time.

The resulting list of present tags has to be converted to enter and leave event notifications, for example by calculating the differences between consecutive scan results. Note that imperfect tag detection algorithms (i.e., schemes that do not detect all present tags in every scan) can cause spurious leave events. The latter can result in event flickering: the fast generation of alternating leave and enter events for a tag that is in fact present all the time. Filters which cancel out spurious leave/enter events are required for such imperfect tag detection algorithms.

Context Typically the application's action when a tag enters or leaves the antenna's range not only depends on the identity of the tag, but also on the presence or absence of other tags during this event – which we call the context of the event. Consider for example the RFID Chef application: the dishes that have to be displayed when a new grocery item is placed on the kitchen counter not only depend on the grocery item itself, but also on the cook. In the Smart Playing Cards application, the action taken when a playing card enters or leaves the antenna's range depends on the other playing cards lying on the table. Note that this notion of context – the presence or absence of tagged objects – is a specific instance of a more general concept [14].

Often applications are only interested in events with a certain context. Consider the Smart Playing Cards for example, where the application only wants to be informed when the last of four players has played his card in the trick. This is equivalent to an enter event with the context of three other cards present. Applications need a way of expressing the context they are interested in, so that they are not pestered with irrelevant information.

Location An RFID system provides only a very simple notion of location – the reading range of an RFID antenna. A tagged object is at this location if its tag can be read by the antenna. Though very simple, this notion of location is useful for many applications, because it serves as a means of grouping tagged objects by their location. In the Smart Tool Box application, for example, all the tools in the range of the tool box antenna belong to the same tool box. An embodiment of such a notion of location therefore is a natural place for queries like "which tools are currently in the tool box". There is a need not only for an adequate representation of such primitive locations, but also for concepts to build higher-levels of location information.

Time Some of the applications require a notion of time. The Smart Tool Box, for example, has to determine the amount of real time that has elapsed between removing a tool from the box and replacing it. The Smart Playing Cards application knows which player played which card by means of the temporal order of the card enter events. In general, there is a need to time-stamp enter and leave events. In the case of multiple tag readers, the time stamps of events originating from different readers should be comparable, even if some of the readers have been offline during event generation. Since events are typically triggered by human actions, a timing precision of 50ms is often sufficient.

State and Behavior Applications typically assign state and behavior to physical objects. In the Smart Tool Box application the state of a physical object (i.e. a tool) consists

of its usage pattern. In the Smart Agenda application, the state of an agenda consists of a schedule. However, there are also stateless applications such as RFID Chef, where the reaction or "output" of the application depends only on the tags currently present.

The applications also differ in the way they assign behavior to physical objects. In the RFID Chef application, for example, all the grocery items and the cook have a "common" behavior – the display of a list of dishes. In the Smart Tool Box application, physical objects have a more "individual" behavior – raising an alarm if they are missing, and calculating tool usage. Moreover, a single physical object can contribute to the behavior of more than one other physical object. In the Smart Playing Cards application, for example, a single card contributes to the "usage context" of all the other playing cards on the table.

A flexible mechanism is therefore needed for assigning state and behavior to physical objects.

History Some applications not only react immediately to entering and leaving tagged objects, but are also queried about their history later on. Consider the Smart Tool Box example, where tools can be queried regarding how long they were used in which tool box on which building site. Therefore, a generic mechanism for logging and querying the history of physical objects seems appropriate.

Communication Infrastructure All the applications we have developed require a TCP/IP-based, Internet-like communication infrastructure. However, there may not always be global connectivity, as in the case of the Smart Tool Box application. The tool box contains a mobile RFID system and an associated computing system, which are able to operate offline. The toolbox is only connected to the background communication infrastructure when it is returned to the workshop. Such disconnected operations must also be supported.

4 Infrastructure Concepts

Having identified a set of common tasks in Section 3, we now elaborate on some of the infrastructure concepts that are intended to support these tasks.

Virtual Counterparts The central concept of our infrastructure is the *virtual counterpart* (VC). A virtual counterpart encapsulates the state and behavior of a tagged physical object. Figure 3 shows four different types of virtual counterparts. The simplest form appears in the playing card VC, which is denoted as a rectangle in the figure. There is a one-to-one mapping between tagged physical objects and their VC. A virtual meta-counterpart, on the other hand, implements "collective" state and behavior for a group of tagged physical objects; that is, it implements an n-to-one mapping of physical objects to state and behavior. Virtual meta-counterparts are depicted as a rounded rectangle, such as the grocery meta-counterpart in Figure 3.

There is a special kind of VC representing a location such as the range of an RFID antenna. These VCs are called virtual locations (VLs) and are depicted as diamonds.

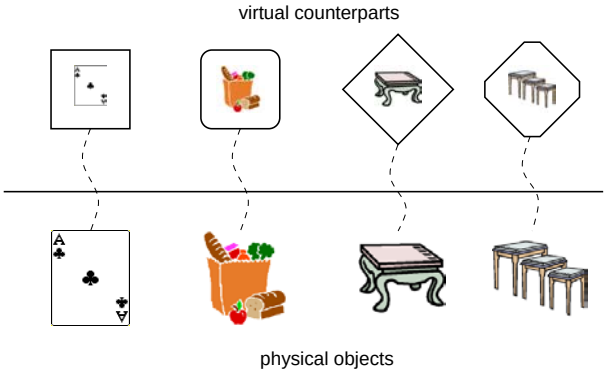

Fig. 3. Virtual Counterparts

Analogous to meta-counterparts, there are also virtual meta-locations, which represent a whole set of locations. Virtual meta-counterparts are denoted by rounded diamonds. We will use the term "counterpart" as a genus for virtual (meta-) counterparts and locations.

Counterpart Events In Figure 3, dashed lines indicate a link between tagged physical objects and their virtual counterparts. As noted in Section 3, information flowing from the physical world to virtual counterparts consists of enter and leave events caused by tags entering and leaving the range of an RFID antenna [1]. A natural approach for informing virtual counterparts of these events is the use of a Virtual Counterpart Event Service (VCES).

The VCES consists of producers generating events, consumers receiving events, and a component that forwards events from producers to consumers. Producers advertise the types of events they are going to generate. Consumers subscribe to the types of events they are interested in. Based on advertisements and subscriptions, the VCES delivers events only to interested consumers.

In our context, producers correspond to software components that generate enter and leave events from the RFID scans, which we call event driver (ED). Consumers correspond to the various types of virtual counterparts. Since multiple counterparts can subscribe to events originating from the same physical object, we can implement an n-to-m mapping between physical objects and virtual counterparts by using the VCES.

In order to support context as described in Section 3, the event service can be programmed by the counterparts. By means of a rule-based program, the application can tell the system the context in which the application is interested in events. If an event and its context match a rule, then the event service generates a context event, which contains context information as well as the information from the triggering event.

Consider for example the Smart Playing Cards application. It only needs to be notified of completed tricks, i.e. when all four players have put a playing card on the table.

[1] An obvious generalization is the propagation of sensor values for objects tagged with sensors as we pursue in the Smart-Its project [23].

Therefore the application programs the event service with a rule that says “notify me of an enter event only if there are already three cards on the table”. If this rule is matched, the event service will generate a context event which contains the identities of all four cards.

Counterpart Life-Cycle Management If a tagged object is detected for the first time, the counterpart associated with that object has to be instantiated. The counterpart state should be saved and the counterpart destroyed if the tagged object leaves the vicinity of an RFID antenna for a long period of time. If the object shows up later at a different location, its counterpart must be re-instantiated using the saved state. These and other issues are what we call counterpart life-cycle management.

A component called the Virtual Counterpart Manager (VCM) is responsible for the tasks of life-cycle management. A VCM is associated with one or more RFID systems and listens to enter and leave events from these systems. The events are assumed to contain the identities of the associated tags and the detecting RFID system. When an enter event occurs, the VCM first checks whether instances of the associated virtual counterpart and virtual location are already active. If this is not the case, the VCM maps the tag and location identities contained in the event to code executables for their respective counterparts and executes them. The mapping service is provided by a separate component called the Virtual Counterpart Repository (VCR).

Multiple VCMs may cooperate in order to support the migration of virtual counterparts. If the same tag shows up in a different RFID system managed by another VCM, the VCM can optionally migrate the counterpart to the new location. This functionality enables the offline operation of mobile RFID systems as required for example by the Smart Tool Box application.

Artifact Memory In Section 3 we pointed out the need for persistently storing the state and event histories of virtual counterparts. The Artifact Memory (AM) fulfills this task. It can store attribute value-pairs belonging to an object (or, more precisely, its VC) in two different places: the RFID tag’s read/write memory and a database system with persistent storage. While only a very limited amount of state can be stored in tag memory[2], this method has the advantage that the saved state is available wherever a tag is. Larger amounts of state and history are stored in the database.

In order to store state in the tag memory, the counterpart event service VCES is used. For this, the AM publishes `store` events to the event service, to which the event drivers subscribe. Upon receiving such a store event, the event driver instructs the RFID system to write to a tag’s memory.

The AM provides a simple interface for queries such as “which VCs were at a particular location at a certain time” or “where is a certain VC now” based on the event histories stored in the AM.

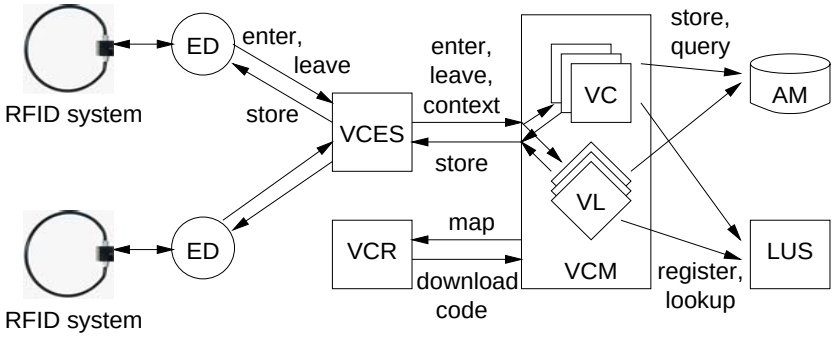

Fig. 4. Infrastructure overview

5 Infrastructure Implementation

We have implemented the concepts outlined in Section 4 in a Java-based infrastructure for virtual counterparts. Figure 4 shows an overview of the system architecture. RFID systems are connected to event drivers (EDs), which generate enter and leave events from periodical tag scans. The EDs act as producers for the counterpart event service (VCES). The VCES delivers events to the counterpart manager (VCM), and specific counterparts (VC, VL). The VCM acts as an execution environment for counterparts. Upon the first sighting of a tagged object or location, it consults the counterpart repository (VCR) to obtain counterpart executables for the tag or location. Counterparts register with the look-up service (LUS), so that cooperating counterparts can find each other. The artifact memory (AM) acts as a place for persistently storing and retrieving counterpart state and event histories. Small amounts of state can also be stored in the tag memory by sending appropriate store events to the VCES.

Event Driver Using the RFID driver software[3], the ED periodically executes tag detection rounds. Each detection round consists of multiple tag scans with a carefully selected maximum time-slots parameter N, which is obtained by using a mechanism we developed in [17]. The latter provides a technique for estimating the actual number of tags present from a scan result, which consists of a list of detected tags as well as the number and type of collisions. Thus, we start with a small value for N, estimate the number of tags N' from the scan result, and re-scan with N'. This process is repeated until all tags have been detected with a probability p. Note that even if p is very close to 1, due to the probabilistic nature of the anti-collision scheme there is a slight chance of one or more tags not being detected.

By calculating the difference between successive detection rounds, a list E of entering and a list L of leaving tags is determined. In order to avoid event flickering as

[2] We expect the amount of memory available in tags to increase over the next few years, however.

[3] We re-implemented the Icode driver software for Linux, which can be downloaded from www.inf.ethz.ch/∼roemer/. Java wrappers for this C library can be downloaded from www.inf.ethz.ch/∼vogt/.

mentioned in Section 3, we do not post events for tags in E and L to the VCES until the next tag detection round has been completed and new lists E' and L' of entering and leaving events have been calculated. In order to avoid spurious leave events, we remove from L and E' all tags that are contained in both L and E'. Now for each remaining tag in L (E) a leave (enter) event is posted to the VCES.

Both enter and leave events contain a tag ID, location ID, and a time stamp. Enter events can carry additional data from the tag memory. The ED also subscribes to store events from the VCES, which contain a tag ID and data that is to be written to the memory of specific tags. Thus the combination of store/enter events can be used to store/retrieve state to/from the tag memory.

Support for the time stamping of events usually requires clock synchronization of all computing devices involved in event handling, such as the event driver, the event service, and the counterparts. Since clock synchronization is problematic during offline operation of certain subsystems – as with the Smart Tool Box application – we developed a time stamping mechanism [13] that eliminates the need for clock synchronization. Instead, time stamps are generated using the unsynchronized local clocks of the computing devices. Whenever an event is sent to a different computing device, its time stamp is transformed to the local clock of the receiver using a simple computation. There is no need for extra synchronization messages. However, time stamp transformation cannot be done precisely, but generates a lower and upper bound for the actual point in time. Therefore, time stamps are represented as intervals. Simple interval arithmetic [13] is used to compare time stamp intervals. Thus, any node can easily compare locally-generated time stamps with time stamps contained in events received from arbitrary remote event sources – even if the latter were offline during event generation.

Virtual Counterpart Event Service Producers and consumers advertise and subscribe to the VCES by specifying the types of events they want to generate or receive. Based on this information, the VCES forwards events to interested subscribers only. The VCES can tell producers bot to produce events if nobody is interested in them.

Subscriptions can optionally contain a rule for specifying context events. Such a rule consists of event declarations and a program. Consider the following example from RFID Chef, which generates context events for grocery items that have been placed on the kitchen counter by a cook.

```
 1: in {
 2:   enter(int id);
 3:   leave(int id);
 4: }
 5: internal {
 6:   cook(int id);
 7: }
 8: out {
 9:   grocery_enter(int groceryid, int cookid);
10:   grocery_leave(int groceryid);
11: }
12: program {
13:   enter[100 <= enter.id & enter.id < 200]:
```

```
14:     add (cook(enter.id));
15:   leave & store(cook)[leave.id == cook.id]:
16:     remove (cook);
17:   enter & store(cook)[200 <= enter.id & enter.id < 300]:
18:     grocery_enter (enter.id, cook.id);
19:   leave[200 <= enter.id & enter.id < 300]:
20:     grocery_leave (leave.id);
21: }
```

The rule declares input events in lines 1-4, events for internal use in lines 5-7, output events in lines 8-11, and a program in lines 12-21. The program converts enter and leave events into grocery_enter and grocery_leave events, to which the RFID Chef virtual meta-counterpart subscribes. In addition to the grocery ID, grocery_enter events contain the ID of the cook.

The program consists of four rules separated by semicolon. Each rule consists of a condition part (to the left of the colon) and an action part (to the right of the colon). Square brackets contain Boolean expressions relating to event arguments; "&" denotes a logical AND.

We assume that tag IDs 100-199 correspond to various cooks and tag IDs 200-299 to various grocery items. The first rule (lines 13-14) expects enter events for cooks and stores an intermediate cook event by using the add command. The second rule (lines 15-16) expects leave events that match a stored cook event and deletes the stored cook event by using the remove command. This means that the stored cook event always corresponds to the current cook. The third rule (lines 17-18) expects enter events for grocery items, retrieves the current cook from the event store and generates a grocery_enter event containing the grocery and cook IDs. The last rule (lines 19-20) expects leave events for grocery items and generates corresponding grocery_leave events.

The VCES context engine provides a number of useful operators for event expressions, Boolean expressions relating to event arguments, and an event store (add, store, and remove commands). A more detailed description can be found in [16].

Virtual Counterpart Manager The VCM acts as an execution environment for the various types of virtual counterparts. It is also responsible for counterpart instantiation, migration, and destruction. For this purpose, the VCM monitors tagged objects by subscribing to enter and leave events.

If the VCM receives an enter event, it first consults the look-up service for matching counterpart instances. If no counterpart exists, the VCM consults the counterpart repository, which maps tag and location IDs to URLs. The URLs point to Java archive (JAR) files, which contain code, resources, and arbitrary additional data for the respective virtual counterparts. The VCM downloads this code, executes it in a separate thread, and registers the counterpart with the look-up service.

If on the other hand the look-up service already contains matching counterpart instances executing in a different VCM instance, the VCM asks the counterpart to migrate to the new location. However, the counterpart may choose to disregard this request.

If the VCM receives a leave event, it asks the respective counterpart to clean up and exit. As with migration, the counterpart may choose to disregard this request.

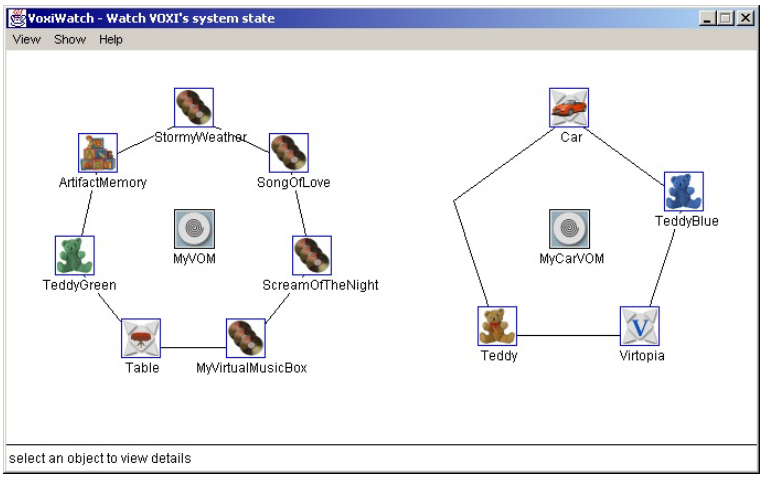

Fig. 5. VCM monitor screen-shot

Once a counterpart is up and running, it can subscribe to events, program the VCES for context events, use the LUS to look-up cooperating counterparts, or store and retrieve state using the artifact memory. Counterparts are Java objects that provide an event API and a set of interface methods to the VCM. Counterparts cooperate by using events or Java RMI.

Note that it is possible to implement *abstract* virtual counterparts which have no physical equivalent by selecting an unused tag ID and manually sending enter/leave events with this ID to the VCM.

We have implemented a graphical VCM monitor that shows the counterparts and locations currently managed by a VCM. Figure 5 shows a screen-shot of the user interface. Icons at the vertices of the polygons denote counterparts managed by the VCM that is depicted at the center of the polygon. By selecting icons with the mouse, additional information on the selected object can be obtained.

Virtual Counterpart Repository The VCR consists of two components, a mapping facility that maps tag and location IDs to URLs, and an HTTP server for downloading the counterpart executables. By mapping multiple IDs to the same URL, we can implement a meta-counterpart (or meta-location), which corresponds to multiple physical objects (or locations).

Look-up Service The LUS is somewhat similar to the VCR in that it maps location and tag IDs to virtual counterparts. However, in contrast to the VCR it returns pointers to executing counterpart objects. Again, meta-counterparts (locations) are implemented by mapping multiple IDs to the same counterpart (location).

Artifact Memory The AM stores state information in the form of attribute/value pairs and event histories. It is implemented as an abstract virtual counterpart. Other virtual counterparts can send predefined events (store state, retrieve state, store event, query events) to the AM. The query event can be used to issue queries to the AM regarding multiple events, such as "which objects were at location X at time T".

The AM internally uses JDBC to open a connection to an SQL relational database. The AM creates one table for persistent state and one table for each event type in the database. The persistent state table has two columns, an attribute column and a value column. The table for a particular event type has one column for each parameter of this event type. The enter event table, for example, has four columns, since enter events have four attributes (tag ID, location ID, time stamp, tag memory contents).

The AM query language is plain SQL, which is passed through to the database unmodified. However, the AM also provides a set of query macros for some common tasks similar to [19]:

- find(TAG, TIME): the location of TAG at TIME
- with(TAG, TIME): tags at the same location as TAG at TIME
- look(LOC, TIME): tags at location LOC at TIME
- history(TAG): list of recent locations visited by TAG

Offline Operation As mentioned earlier, the mobile RFID system and associated mobile computing devices in the Smart Tool Box also have to operate while disconnected from the background communication infrastructure. We call such a mobile computing and RFID system a *sub-network*.

The support for offline operation had all sorts of consequences for the design of our Ubicomp infrastructure. The counterpart life-cycle management must be aware that an instance of a counterpart might possibly exist in a temporarily disconnected sub-network. When the sub-network re-connects, the collected data must be synchronized between the instances, and one of the instances must be unloaded. The synchronization can be accomplished with a specialized synchronization event type.

In order to support the offline operation of sub-networks, all vital infrastructure components are replicated in each sub-network. Most of these components are able to operate independently of their replicas in other sub-networks. However, the look-up service and artifact memory are special cases, because their contents can be changed independently and concurrently in different sub-networks. When sub-networks re-join, the contents of the LUS and AM replicas have to be synchronized in a sensible way.

Synchronizing the LUS replicas is simple – whenever a look-up fails, an LUS instance looks for other LUS replicas in connected sub-networks and forwards the query to them. Matters are more complicated with the AM, however. A distributed database or database synchronization mechanism is needed to join the information contained in multiple AM replicas. This is mandatory in order to issue queries regarding events collected in different sub-networks. We are currently working on this non-trivial issue.

VCMs in different sub-networks may cooperate in order to support the migration of virtual counterparts across sub-networks. This functionality enables virtual counterparts to follow their respective physical objects.

In situations where we can store some data directly on the RFID tag, we can provide some simple event history or context information to support offline operations. Sometimes this may be the only way of transporting data between different disconnected (sub-)networks.

6 Proof of Concept

As a proof of concept we began re-implementing our earlier prototype applications using the infrastructure we had developed. In this section we compare the original version of Smart Playing Cards (called SPC 1) and the re-implementation based on our infrastructure (called SPC 2). Both SPC 1 and 2 implement the game of Whist [24], a game for four players in two fixed partnerships ("teams").

SPC 1 is implemented as a single monolithic program. Conceptually it consists of three major parts: tag detection, game logic, and user interface. However, the implementation merges these three parts in a rather complex state machine. The program sits in a loop and periodically executes tag scans. From the scan result, lists of leaving and entering tags are determined, and fed to the state machine. Depending on its current state and the input, the state machine triggers an action in the user interface and changes its state.

In SPC 2 we tried to separate the three conceptual parts on the implementation level by using our infrastructure. Tag detection was already provided by the latter, so we did not have to worry about it in the application. The user interface was implemented by an abstract virtual counterpart that subscribed to the following events:

- ShowCard(PLAYER, SUIT, RANK): PLAYER played card with SUIT and RANK, display the card
- RemoveCard(PLAYER): PLAYER removed his card, no longer display it
- UpdateScore(SCORE, TEAM): update the SCORE of the TEAM
- DisplayStatus(TEXT): display TEXT in the status line of the user interface
- ShowTrumps(SUIT): display trump SUIT

In other words, the user interface does not implement any game logic. However, every playing card has an associated virtual counterpart which implements the more advanced parts of the game logic, such as playing hints and the cheat alarm. The more basic parts of the game logic, such as the detection of the trump color and of completed rounds, are implemented as a set of context rules for the VCES. Using these rules, the VCES generates game-related events from the basic enter and leave events for playing cards and sends them to the playing card virtual counterparts. The latter implement the more complex game logic and send events to the user interface.

By using our infrastructure, we achieved a clear separation of the conceptual components of the application at the implementation level, which eases further development of the application and makes is less prone to programming errors than the monolithic approach used in SPC 1.

7 Related Work

There are a number of other projects that aim to provide infrastructure for smart environments. In contrast to these other projects, our infrastructure has been specifically tailored to the needs of multi-object, tagging-based applications based on our experience with these.

CoolTown [7] and the associated CoolBase infrastructure aim to give people, places, and things a Web presence. This Web presence has a similar function as our virtual counterparts. The use of well-known Web technology is both an advantage and a disadvantage. On the one hand, this technology is proven and widely available, but on the other hand we think there is an important difference between Web and Ubicomp applications. Due to its origin, the Web is document-centric. Although it has been augmented with ways to include dynamic distributed applications (e.g. SOAP), it still retains its inherent hypertext nature. On the other hand, Ubicomp applications are more akin to dynamic distributed applications. Therefore the counterpart approach based on computational objects we use seems more appropriate for the kind of Ubicomp application we want to support. It is possible, however, that the emerging XML-based Web infrastructure (i.e. Web services) might give rise to a platform that would also suit our needs.

The Informative Things [1] approach is somewhat similar to CoolTown in that it attaches information to things by using Web technology.

One.world [4] is an infrastructure for pervasive computing. It provides rather basic abstractions and mechanisms for coping with the dynamics of pervasive computing applications. Our infrastructure differs from one.world in that it is much more tailored to the specific needs of tag-based ubiquitous computing applications.

Nexus [5] aims to provide an infrastructure for spatially-aware applications in a very broad sense. Early prototypes are currently being developed. Although the mechanisms provided are rather generic, it can be difficult to adapt them to the more specific needs of tag-based applications. Due to our simple location model, most of the functionality provided by Nexus is not required by the type of application we intend to support with our infrastructure.

Hive [10] has grown out of a mobile agent system. While the agents provided by Hive can be used to implement something like a counterpart, there is no explicit support for linking real world objects to counterparts, location, time, and history.

CALAIS [11] is an infrastructure for smart badges, which is also based on distributed events. Although it supports a much more fine-grained notion of location, it lacks support for counterparts.

The event service and context event support in our infrastructure are related to various other content-based messaging systems such as Elvin [15] and SIENA [2]. Some systems such as GEM [9] also support event aggregation, but the mechanisms provided are insufficient to express complex event contexts.

8 Conclusion and Outlook

We have presented the rationale, design, and implementation of an infrastructure for RFID-based ubiquitous computing applications. In order to reach a high level of us-

ability, the infrastructure has been designed with a focused set of applications in mind, based on our experience with the development of a set of prototype applications over the last two years. As a proof of concept we re-implemented one of the earlier applications (Smart Playing Cards) using this infrastructure.

In this paper we have given a brief overview of some of the applications we have developed, derived tasks common to these applications, presented concepts to support these tasks, and described the architecture and implementation of our infrastructure. Key concepts of the infrastructure are virtual counterparts, context event service, and artifact memory.

In the future we intend to extend the infrastructure to other, similar kinds of sensing devices and mobile ad-hoc networked environments. With the evolution of RFID tags into smart tags equipped with sensing, computing, and communication capabilities, it is becoming possible to provide a bare minimum of infrastructure services directly on the tag itself. We are pursuing this research direction in the Smart-Its project [6] [23].

The general goal here is to learn about appropriate concepts for a general infrastructure for smart environments and to gain experience by implementing and applying those concepts in typical application scenarios.

9 Acknowledgments

We would like to acknowledge Marc Langheinrich and Harald Vogt for their work on RFID Chef, Matthias Lampe for his work on Smart Tool Box, and Philip Graf, Svetlana Domnitcheva, and Vlad Coroama for their help with Smart Playing Cards.

References

1. E. Barrett and P. Maglio. Informative Things: How to Attach Information to the Real World. In *UIST 98*, San Francisco, USA, November 1998.
2. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. In *Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, Portland, OR, July 2000.
3. T. Dübendorfer. An Extensible Infrastructure and a Representation Scheme for Distributed Smart Proxies of Real World Objects. Master's thesis, ETH Zurich, 2001. Also available as technical report TR-359. www.inf.ethz.ch/vs/publ/papers/TR_359.pdf.
4. R. Grimm et al. Programming for Pervasive Computing Environments. Technical Report UW-CSE-01-06-01, University of Washington, Department of Computer Science and Engineering, June 2001.
5. F. Hohl, U. Kubach, A. Leonardi, K. Rothermel, and M. Schwehm. Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial-Aware Applications. In *MobiCom 99*, Seattle, USA, August 1999.
6. L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.-W. Gellersen. Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In *Ubicomp 2001*, pages 116–122, Atlanta, USA, September 2001.
7. T. Kindberg et al. People, Places, Things: Web Presence for the Real World. In *WMCSA 2000*, Monterey, USA, December 2000.

8. M. Langheinrich, F. Mattern, K. Römer, and H. Vogt. First Steps Towards an Event–Based Infrastructure for Smart Things. In *Ubiquitous Computing Workshop, PACT 2000*, Philadelphia, PA, October 2000. www.inf.ethz.ch/vs/publ/papers/firststeps.pdf.
9. M. Mansouri-Samani and M. Sloman. GEM – A Generalised Event Monitoring Language for Distributed Systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4(25), February 1997.
10. N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes. Hive: Distributed Agents for Networking Things. In *ASA/MA 99*, Palm Springs, USA, October 1999.
11. G. J. Nelson. *Context-Aware and Location Systems*. PhD thesis, University of Cambridge, 1998.
12. K. Römer. Smart Playing Cards - A Ubiquitous Computing Game. In *Workshop on Designing Ubiquitous Computing Games, Ubicomp 2001*, Atlanta, USA, September 2001. www.inf.ethz.ch/vs/publ/papers/ubicomp01-smart-playing-cards.pdf.
13. K. Römer. Time Synchronization in Ad Hoc Networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 01)*, Long Beach, CA, October 2001. www.inf.ethz.ch/vs/publ/papers/mobihoc01-time-sync.pdf.
14. D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *CHI 99*, Pittsburgh, USA, May 1999.
15. B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *AUUG 97*, Brisbane, Australia, September 1997.
16. J. Senn. Eine Ereignis-Architektur f"ur kontextsensitive Anwendungen in verteilten Systemen. Master's thesis, ETH Zurich, 2001.
17. H. Vogt. Efficient Object Identification with Passive RFID Tags. Submitted for publication, February 2002. www.inf.ethz.ch/vs/publ/papers/rfid_obj.pdf.
18. R. Want, K. Fishkin, A. Gujar, and B. Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. In *ACM Conference on Human Factors in Computing Systems (CHI 99)*, Pittsburgh, PA, May 1999.
19. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.
20. R. Want, T. Pering, G. Borriello, and K. Farkas. Disappearing Hardware. *Pervasive Computing Magazine*, 1(1):36–47, 2002.
21. M. D. Weiser. The Computer for the 21st Century. *Scientific American*, pages 94–104, September 1991.
22. MICO: MICO is CORBA. An Open-Source CORBA Implementation. www.mico.org.
23. Smart-Its Project. www.smart-its.org.
24. The Game of Whist. www.pagat.com/whist/whist.html.
25. The Philips I–Code System. www-us2.semiconductors.philips.com/identification/-products/icode.