# Specification and Verification of Complex Location Events with Panoramic

Evan Welbourne, Magdalena Balazinska, Gaetano Borriello, and James Fogarty

Computer Science & Engineering
University of Washington
Seattle, WA 98195 USA
{evan, magda, gaetano, jfogarty}@cs.washington.edu

**Abstract.** We present the design and evaluation of Panoramic, a tool that enables end-users to specify and verify an important family of complex location events. Our approach aims to reduce or eliminate critical barriers to deployment of emerging location-aware business activity monitoring applications in domains like hospitals and office buildings. Panoramic does not require users to write code, understand complex models, perform elaborate demonstrations, generate test location traces, or blindly trust deterministic events. Instead, it allows end-users to specify and edit complex events with a visual language that embodies natural concepts of space and time. It also takes a novel approach to verification, in which events are extracted from historical sensor data traces and then presented with intelligible, hierarchical visualizations that represent uncertainty with probabilities. We build on our existing software for specifying and detecting events while enhancing it in non-trivial ways to facilitate event specification and verification. Our design is guided by a formative study with 12 non-programmers. We also use location traces from a building-scale radio frequency identification (RFID) deployment in a qualitative evaluation of Panoramic with 10 non-programmers. The results show that end-users can both understand and verify the behavior of complex location event specifications using Panoramic.

## 1 Introduction

Intelligent behavior in location-aware computing is driven by *location events*. Applications detect events by dynamically evaluating spatio-temporal relationships among people, places, and things. For example, a location-aware to-do list might detect simple events like "Alice is near the library" to trigger reminders. In contrast, many new applications for real-time location systems (RTLS) rely on *complex events* that contain sequences of interactions [1, 27]. For example, a hospital workflow tracker may log a "cardiology exam" whenever a patient is detected exiting the hospital after meeting with a cardiologist and then spending time with a nurse and an electrocardiogram machine. With the RTLS market expected to soon exceed $2 billion US [15], support for complex events is crucial.

A fundamental part of support for complex events is *event specification*; users must be able to specify new events to meet their evolving needs. Applications achieve this by leveraging event detection systems (e.g., context-aware computing infrastructures) that allow new events to be formally specified in some manner. However, while existing systems allow developers to specify new events using low-level APIs [3, 30, 33] or a declarative query language [10, 14], dependence on developers is a costly inconvenience for both individuals and organizations. Indeed, a recent survey of location systems in hospitals cited the cost of tuning vendor software to site requirements as a critical barrier to deployment [34]. A compelling alternative is to allow direct specification of complex events by end-users. This is challenging, however, because it requires translation of high-level concepts into conditions on diverse, low-level, and uncertain sensor data. Unfortunately, existing systems for end-user event specification are either limited by inexpressive interfaces [17, 24] or require iterative and potentially unfeasible training demonstrations for machine learning models [8].

It is equally important that end-users be able to *verify* event specifications and debug those that do not work. Verification is difficult because it requires a system to produce high-level evidence of a specification's behavior over sensor traces that may be too complex for an end-user tool to generate. Moreover, because bugs can occur at the sensor level (e.g., calibration errors) or in the specification design, users must be able to understand detected events and evaluate their relationship to sensor data. This is impractical or impossible when events are specified with inscrutable machine learning models or when they do not represent uncertainty. As such, existing systems for end-user event specification are limited by inadequate support for verification and debugging.

We present Panoramic, a web-based tool that enables end-users to specify and verify complex location events. Panoramic does not require users to write code, understand complex models, perform elaborate demonstrations, generate test traces, or blindly trust deterministic events. Instead, it offers an intuitive visual language for specification and an intelligible verification interface that uses readily available historical sensor data. Specifically, we contribute:

1. A significant upgrade to our existing event detection system, Cascadia [37]; we facilitate event specification and verification by integrating Lahar [28], a new type of probabilistic event detector (Section 3).
2. Extensions that prevent errors and increase the expressive power of Scenic, our existing event specification tool. Our changes are guided by a formative user study with 12 non-programmers (Sections 3 and 4).
3. A novel approach to verification that leverages both historical sensor traces and a user's knowledge of past events while explicitly but intuitively representing the probabilistic nature of sensor data and events (Section 5).
4. Verification interface widgets that provide end-users with an intelligible and hierarchical visualization of context. The widgets also allow users to distinguish sensor errors from errors in a specification's design (Section 5).
5. A qualitative evaluation of event verification in Panoramic with 10 non-programmers. The study uses actual radio frequency identification (RFID) location traces collected from a building-scale deployment (Section 7).

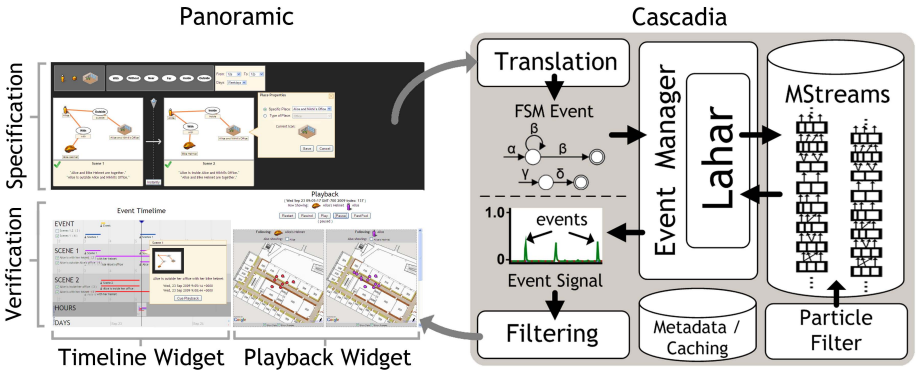


**Fig. 1. The Panoramic system architecture. Events are iteratively specified in Panoramic, detected over historical location traces by Cascadia, and then displayed in Panoramic's verification interface.**

## 2 The RFID Ecosystem

We design and evaluate Panoramic using RFID traces, an increasingly common type of location data [29, 34]. Our deployment, the RFID Ecosystem, models an enterprise RTLS deployment using 47 EPC Gen-2 RFID readers (160 antennas) installed throughout our 8,000 $m^2$ building. In addition, more than 300 passive (e.g., unpowered) tags carrying unique identifiers are attached to people and objects. When a tag passes an antenna it may wirelessly transmit its identifier to a reader, which in turn creates and sends a timestamped *tag read event* of the form `(antenna location, tag ID, time)` to a server for storage and processing. However, our antennas are only deployed in corridors (not inside offices), and past work has shown that factors like RF-absorbency and mobility in an everyday environment may prevent tags from being read [36, 39]. Thus, like many location systems, the RFID Ecosystem may produce sporadic, imprecise location streams.

## 3 Specifying and Detecting Events

In this section, we describe Cascadia, Scenic, and extensions we make to support Panoramic. We also present a taxonomy for the events Panoramic can specify.

### 3.1 Integrating Lahar into Cascadia

Cascadia is a system for specifying, detecting, and managing RFID events [37]. It accepts declarative event specifications and detects the specified events over incoming RFID data, producing one event stream per specification. Cascadia copes with uncertainty by transforming intermittent RFID streams into smoothed, probabilistic *Markovian Streams* (MStreams) [19] that capture both the uncertainty of a tag's location at each time step (e.g., a distribution over which rooms the tag might be in) and the correlations between a tag's possible locations (e.g., distributions over entire paths through a building). MStreams abstract away the complexities of sporadic and imprecise data to expose a more uniform model

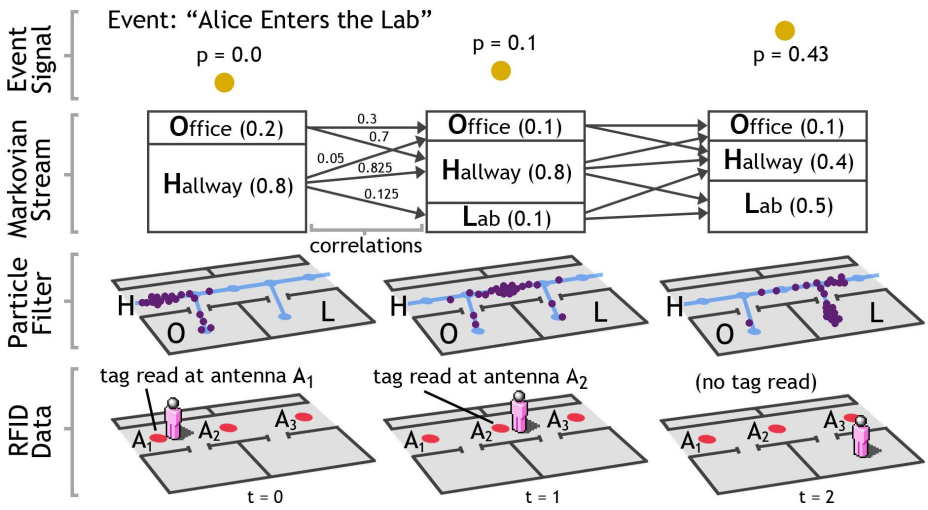


**Fig. 2. Cascadia transforms raw, uncertain location data into smoothed, probabilistic Markovian streams over which Lahar detects complex events.**

of location over which event specifications are expressed, thus considerably simplifying the requirements of an event specification language. At the heart of Cascadia is the PEEX event detection engine, which uses an SQL-like event specification language and extracts probabilistic events from MStreams.

We upgrade Cascadia by replacing PEEX with the Lahar event detection engine. Lahar's query language is based on regular expressions that are represented internally with finite state machines (FSMs). The language's pattern matching constructs, together with standard query predicates, offer a more intuitive way to express sequential spatio-temporal events. This streamlines translation from Panoramic specifications into Lahar queries and provides end-users with an easily comprehended mental model of the event detection process. Lahar produces a single probabilistic query signal for each MStream it processes. The query signal for an event specification, or *event signal*, consists of timestep-probability pairs <t,p> which indicate that the event occurred at timestep t with probability p (see Figure 2). Each probability p is derived from an MStream using well-established query answering techniques for probabilistic databases [19, 28].

Lahar is an evolving research prototype that is currently limited to answering event queries over a single MStream. Panoramic, however, creates event specifications that reference multiple MStreams (e.g., multiple people and objects). As such, we developed a new *Event Manager* module for Cascadia that answers queries over multiple MStreams. The Event Manager translates specifications into sets of single MStream queries after which it orchestrates their execution with Lahar and merges the resulting event signals. This module also manages metadata on MStreams and caches intermediate event signals for reuse when answering other queries. Overall, our upgrades make Cascadia more expressive and provide cleaner semantics because all queries map to state machines.

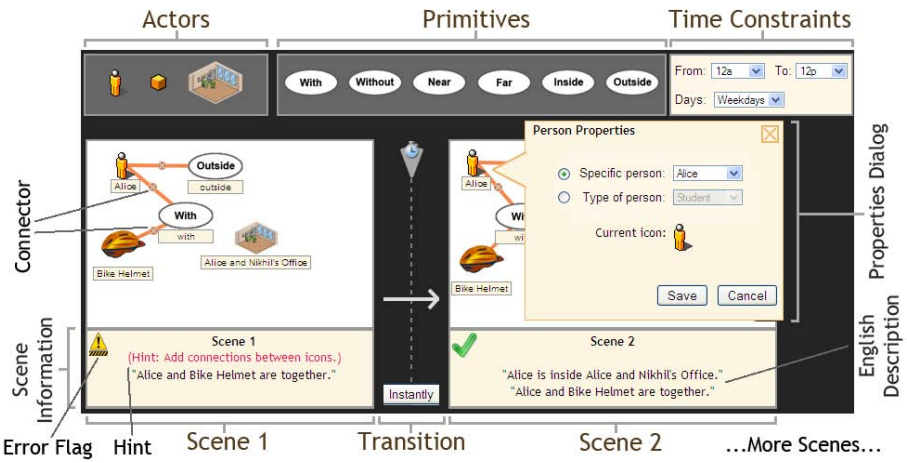


**Fig. 3. The event specification interface employs a storyboard metaphor.**

### 3.2 Enhancing Scenic

Scenic is a tool that allows end-users to specify events for PEEX [37, 38]. It uses an iconic visual language designed to support common location events and their composition through sequencing and conjunction. A storyboard layout describes how people and objects enact an event through a sequence of movements between places (see Figure 3). Users drag and drop *Actors* (people, objects and places) and *Primitives* (instantaneous events) onto *Scenes*. A *Sequence* of Scenes specifies a complex event as a sequence of spatio-temporal *sub-events*. Actors are specified using other end-user tools discussed in prior work [38].

While end-users understood Scenic [37], translation of specifications into PEEX queries was complex and imposed awkward constraints on the interface. For example, only certain combinations of Primitives could appear in a Scene and transitions between Scenes had to be of fixed length (e.g., one second). By designing Panoramic to target Lahar, we were able to remove these constraints and add several new features that increase expressiveness (see Figure 3). We added explicit *Connectors* between Actors and Primitives, enabling any combination of Primitives to appear within a single Scene. We also added *Transitions* between Scenes that are set explicitly by the user as occurring either *instantly* (i.e., in one timestep) or *over time* (i.e., some time may pass before the next scene occurs). Finally, we included simple constraints on absolute time (e.g., "before 12pm on Weekdays") as a convenience.

### 3.3 Supported Events

In a survey of location events in pervasive computing applications [37] we found that six instantaneous events were used most often: *with*, *without*, *inside*, *outside*, *near*, and *far*; and that these events were frequently composed using *conjunction*, *repetition*, and *sequencing*. A large fraction of these events can be expressed in Panoramic and translated directly into FSM queries for Lahar. We now describe and illustrate the set of location events Panoramic supports.


**Fig. 4. A Single Scene event: "*I'm in my office*", translates into a single state FSM that enters the accept state whenever the user is inside his office.**

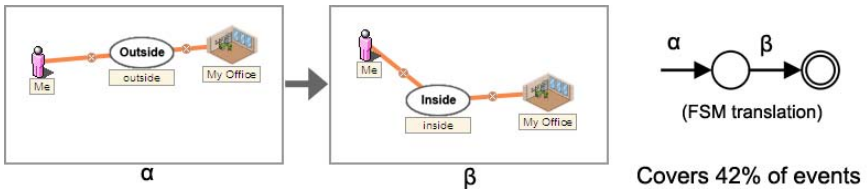


**Fig. 5. A Consecutive Sequence event: "*I enter my office*", translates into a linear FSM that enters accept state when Scenes are satisfied consecutively.**

**Single Scene Events** Single Scene events use connected Primitves and Actors to describe a set of instantaneous events that occur simultaneously. Scenes including one person or object (i.e., one MStream) can be translated into a FSM query having a single accept state and a single incoming edge that is satisfied when the Scene's Primitives are true (see Figure 4). When the Scene includes multiple people or objects, the Event Manager breaks the query into multiple single-state FSM queries, answers each, and merges the results by assuming their mutual independence and computing their conjunction. The set of events that can be represented by a single Scene is greatly extended by Connectors. Single Scene events form the basis of all location-aware computing applications and are sufficient to account for 24% of events recorded in the survey.

**Consecutive Sequences** Consecutive Sequences contain Scenes separated by instantaneous Transitions. They are translated into linear FSM queries that have a single accept state preceded by a sequence of states and edges as shown in Figure 5. In the case of multiple objects or people, the Event Manager processes each Scene as though it were a Single Scene event, joins the results into a composite stream of independent events, and then runs a linear FSM query that corresponds to the Sequence. Consecutive Sequences are useful for detecting state transitions like entering or exiting a place, approaching an object, or beginning a trajectory; such events account for another 42% of surveyed events.

**Sequences with Gaps** Sequences may also contain Transitions that allow time (and potentially other events) to pass between one Scene and the next. Panoramic translates these Sequences into linear FSMs having a self-loop edge that is satisfied by the *negation* of the condition on the edge which leads to the next state. For example, the self-loop edge in Figure 6 ensures that the FSM remains in "pre-meeting" state until Alice, Nikhil, and Prof. Chen are all in Prof. Chen's office. Sequences with gaps represent a class of events not previously supported by Cascadia, thus increasing the flexibility and expressiveness of specifications. They also account for another 11% of the events in our survey.

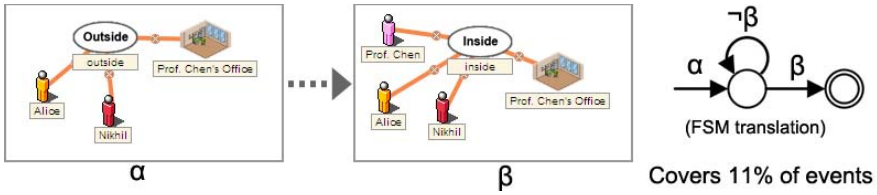


**Fig. 6. A Sequence event with a gap: "***Alice, Nikhil and Prof. Chen begin a meeting in Prof. Chen's office***". This event occurs as soon as the last participant enters the office. A FSM with a self-loop is used to wait for the last participant.**

**Disjunctions of Sequences** A disjunction of Sequences may represent different orderings of instantaneous events or alternate paths in a workflow. While such events comprise less than 5% of events in the surveyed literature, they are of growing importance in emerging domains like workflow monitoring in hospitals [27]. Panoramic does not directly support disjunctions, but users may specify multiple Sequences and compose their disjunction within an application. Here we stress that Cascadia has the capability to support disjunctions of Sequences and that we plan to extend Panoramic to directly specify them in future work.

Actor variables also create disjunctions of Sequences. Most surveyed events can be usefully modified with variables. For example, instead of "*Alice meets Nikhil in her office*", one could express "*Two researchers meet in any room*". These events are translated into a disjunction of Sequences where the variables in each Sequence are replaced with a different combination of possible values. This is effective for events with few variables that range over a limited domain.

**Unsupported Events** There are another 18% of surveyed events for which Panoramic provides limited or no support. This includes repeating sequences, which can be translated into FSMs with cycles. These queries are computationally difficult to answer over probabilistic data, but can be approximated by unrolling the cycle. Other common and unsupported location events involve precise 3D location and events involving speed of transition (i.e., velocity).

### 3.4 Formative Evaluation

In addition to the move from PEEX to Lahar, the design of Panoramic's event specification interface was driven by a formative user study. In this study, 12 non-programmers were given a brief tutorial on Panoramic and asked to complete a series of event specification tasks while talking aloud. Each task presented participants with an example application and usage scenario for which they were asked to specify an appropriate event. After participants declared a task complete, a researcher would then review the produced specification and explain exactly how Panoramic would interpret it. Participants could then revise their specification if the researcher's explanation did not match their intention.

As a result of the study, we identified a variety of usability and expressivity problems which we addressed with the design features presented above. However, we also observed several more fundamental problems regarding user ability to understand and verify the behavior of a created specification. We discuss these problems in the next section.

## 4 Problems in Specifying an Event

Even with the enhancements to the Scenic interface, users encountered a variety of difficulties while authoring an event specification with Panoramic. Here, we summarize and discuss those most frequently observed in our formative study.

### 4.1 Syntax Errors

Nearly all participants produced one or more syntactically illegal specification. Most syntax errors were the result of forgotten connections or targeting errors while rapidly creating a Scene. In a few cases, participants made mistakes because they did not clearly understand how Actors and Primitives could be connected. We addressed these problems with three new features (see Figure 3). First, we constrained the interface to allow only legal connections between Actors and Primitives, thereby eliminating the possibility of syntax errors. We also added an *information panel* below each Scene that displays English explanations for fully connected Primitives in that Scene. Finally, we included a *status flag* in the information panel that shows a green checkmark when a Scene is legally specified or an under construction symbol when more work needs to be done. If more work is needed, the information panel provides a hint as to what elements (e.g., Actors, Primitives, Connectors) are needed to complete the Scene.

### 4.2 Design Problems

A problem encountered by 3 of the first 5 participants was in deciding on a specification design that would meet the task's requirements. While 1 participant had difficulty reasoning about what needed to be specified, others knew what they wanted to specify but weren't sure how the available widgets could be composed to do it. To address both of these problems, we introduced *event templates*, stock specifications that provide examples of common events with their usage scenarios - at least one for every type of event in the taxonomy. The last 7 participants in our formative study were provided with a library of templates during the study session; those that encountered design challenges were able to use the template library to decide on a design.

### 4.3 Problems with Timing

Half of the participants chose to revise a completed specification due to problems with timing. These problems consisted in use of the wrong Transition type (e.g., *instantly* instead of *over time*) or in using one Scene when two Scenes were needed. For example, a participant intended to specify "the custodian leaves the lab and goes to the closet" as two Scenes separated by an instantaneous Transition. This specification was flawed because a custodian cannot move instantly between rooms. Another participant used a single Scene to tell a context-aware notifier to send her an email when a meeting occurs. While the single Scene would effectively detect a meeting and send an email, it would also occur repeatedly throughout the meeting, causing many emails to be sent. Event templates helped to reduce problems with timing, but some more subtle problems remained, forcing participants to revise their specifications. We therefore developed additional solutions to timing problems which we present in Section 5.

### 4.4 Tuning the Level of Specificity

A critical challenge faced by all participants while designing and revising their specifications was to determine the appropriate level of specificity. Some participants routinely *under-specified* events by leaving out Actors or Primitives. One explained her under-specification by saying "I wanted it to cover every case so I only need one event," another simply explained that it took two revisions before he realized that another object was needed. *Over-specification* was also common. In such cases, participants often explained that they added non-essential Actors, Primitives, or Scenes because they weren't confident that Panoramic would detect the intended event without them. For example, one participant constrained a "group meeting" event to occur only when all group members were together in a room with laptops and coffee mugs. He explained that "they could be in the room for some other reason, but if they all have laptops and coffee then they're probably in a meeting." Whether an event is over or under specified, the result is a specification that does not fit the user's intention.

While mismatches in specificity may be less of a problem when users are reasoning about events in their own life with which they are intimately familiar, some tuning is likely to be required whenever a new event is specified. Many participants adopted a trial and error methodology when specifying an event, using the researcher's explanation to "test" an event's behavior over multiple design iterations. This motivated the design of additional interface components that support users in understanding and verifying the behavior of a specification. We discuss these components in the next section.

## 5 Understanding and Verifying Events

In response to the problems discussed in the previous section, we extended Panoramic to support end-users in understanding and verifying their event specifications. Here we face the hard problem of generating test data for specifications. Our approach is to allow users to assess the correctness of a specification by running it on *historical data*. We use Cascadia to detect the event together with the timeline-based and map-based widgets presented below to visualize the results. The advantage of this approach is that it does not require complex simulations or synthetic data which may not be truly representative. Instead, it reveals the behavior of the event on real, readily available sensor data. The obvious constraint is that the tested event must have already been recorded by user's RFID deployment. This is a reasonable sacrifice for scenarios like hospitals and office environments where both simple events and complex workflows occur repeatedly.

### 5.1 Timeline Overview

We developed a timeline widget (see Figure 7) that provides a rapid overview of detected event results as horizontal *bars* in a timeline where a bar's start and end points correspond to a detected event. Events are organized in groups of sub-events (e.g., sub-sequences, Scenes, Primitives) in order to provide an in-depth explanation as to why an event was or was not detected. Each group

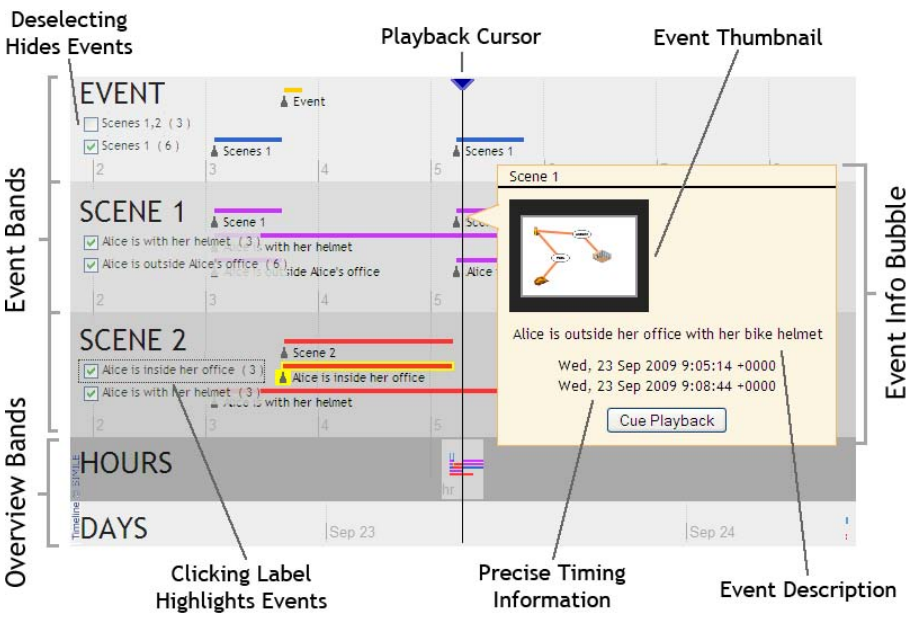


**Fig. 7. The timeline reviews the event "***Alice enters her office with her helmet***".**

of events is displayed in its own *band*, with one band for the event itself along with its sub-sequences, and one band for each Scene with its Primitives. By correlating the sub-event bars with the presence or absence of an event bar, users can gain an understanding of how each sub-event contributes to or detracts from the detection of the event. For under-specified events, this process can reveal that frequently occurring sub-events result in a larger than intended number of detections. In the case of over-specification, it can show a user that absent sub-events are preventing the event from being detected. Timing problems are also visible as unexpectedly long or short event durations.

The timeline further facilitates the verification process with exploratory browsing functions. The timeline can be dragged left or right to move through time, and two zoomed-out bands (one for hours and one for days) provide additional context for large datasets. By default, event bands are rendered with minute-level granularity but may be zoomed in or out using the mouse scroll wheel. Checkbox labels on the left side of each band describe the contents of each sub-event group in that band. The bars representing a sub-event can be shown or hidden from the timeline view by checking or unchecking that sub-event's checkbox. Selecting a checkbox label will highlight all occurrences of an event and its sub-events in the timeline. Finally, clicking a bar in the timeline brings up a bubble containing a thumbnail image of the corresponding event or sub-event along with an English description and precise timing information.

The timeline-based design is particularly well-suited for display of sequential data and allows users to leverage their natural ability for reasoning about temporal events [16]. With a suitably large dataset, we anticipate that the timeline view can help users to quickly gather evidence of a specification's behavior.

**Filtering Event Signals** The timeline must present a simple, discrete view of the complex probabilistic data it displays. As such, we take several steps to transform raw event signals from Lahar into discrete event streams that are amenable to visualization. First, because the probability of a true event occurrence may vary widely, we identify and flag all local maxima in a signal as potential event occurrences. We then filter out all spikes (i.e., peaks lasting less than 2 seconds) from this set. Spikes are unlikely to correspond to a true event because they are faster than any action humans commonly perform. They may occur in an entered-room event signal, for example, when a user passes but does not enter the room. After removing spikes, we transform the remaining "humps" into discrete events having the same duration and which can be displayed on the timeline.

### 5.2 Detailed Playback

Though it provides a useful overview and a direct look at the cause for Sequence and Scene events, the timeline does not explain why a given *Primitive event* does or does not occur. This is a crucial question when working with real historical sensor traces because missed RFID readings can lead to false positives or negatives that are indistinguishable from unexpected behavior in the timeline. For example, a "group meeting" event may match a user's intention but fail to work in practice because one group member's RFID tag is routinely missed. Using only the timeline, it would be impossible to distinguish an absent group member from a tag that needs to be replaced. To help users identify problems that are rooted in sensor errors rather than in a specification, we developed a map-based trace playback widget. The widget allows a user to semantically drill-down into any point in the timeline and review both the sensor trace and the MStream starting at that time.

The playback widget renders at the right side of the timeline (see Figures 1 and 8) whenever a user clicks the "Cue Playback" button in the pop-up bubble for a timeline bar. At the same time, a playback cursor appears over the timeline to designate the current time in the trace to be replayed. Users may also be interested in portions of the timeline that contain no detected events, as such, they can drag the timeline to any location to cue playback there. Standard video controls (e.g., pause, play, stop, rewind and fast-forward) provide a familiar interface for reviewing segments of the trace. Playback occurs in a collection of *map panels* that show RFID readings and MStream data (i.e., particles from the particle filter) overlaid on a map of the RFID deployment. The timeline scrolls synchronously with the map-based playback. Each such panel follows a particular person or object from the trace, automatically panning and switching floors as needed. The user may also choose to show multiple traces in a single map panel by selecting checkboxes above the map that correspond to additional traces. RFID readings and MStream data may be switched on and off in a given map panel using the "Show Data" and "Show Model" checkboxes. Unneeded map panels can be collapsed to facilitate side-by-side comparison of traces. Labels at the top of the playback widget show the current time in the trace and summarize the collection of people and objects being viewed in the trace.

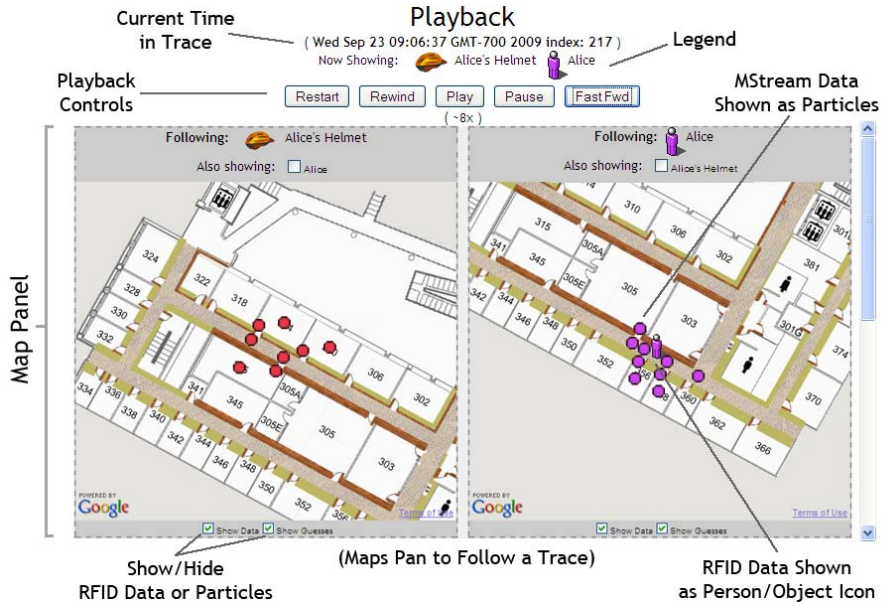


**Fig. 8. The playback widget reviews traces for Alice and her helmet. Person and helmet icons represent raw location data. Particle icons display a probability distribution over a tracked entity's possible locations.**

## 6 Implementation

Panoramic's event specification and verification interfaces are entirely web-based and built using the Google Web Toolkit [12]. The playback widget was built using Google Maps [11], and a customized version of MIT's Simile Timeline [31] was used to implement the timeline widget. In total, Panoramic contains 152 Java classes that control the interface and orchestrate AJAX communication with a server. An additional 42 classes were added to Cascadia to support translation and execution of Panoramic events with Lahar.

## 7 Evaluation

We ran a qualitative study of Panoramic's event verification capabilities with 10 non-programmers. Participants were offered $20 to perform 60 minutes of event verification tasks. In addition to a 10 minute tutorial, participants were prepared with a background story that described a week in the life of Alice, a fictional student. The story included precise information on events that occurred (e.g., "group meeting on Monday at 11") as well as information on events that often occur (e.g., "Nikhil often stops by Alice's office to talk"). Each task included a specification, a description of the application and usage scenario for which Alice created the specification, and the corresponding detected events from the week of Alice's data. Participants were asked to talk aloud as they used Panoramic in combination with what they knew about Alice's week to decide how each specification worked, whether it met her needs, and how it could be fixed.

The week of historical data was spliced together from traces collected in the RFID Ecosystem and for which we have ground truth information on when and where events occurred. We combined a set of high fidelity traces with a small number of highly ambiguous traces (e.g., traces with a large number of missed tag reads). The first four tasks were presented in random order with one task having a specification that clearly succeeded over the week of data and three that failed because they were over-specified, under-specified, or contained a timing error. A fifth task contained a specification that should have met Alice's needs but was not detected over the week's data as a result of ambiguous sensor traces.

### 7.1 Observations and Enhancements

Overall, participants were able to complete the tasks, averaging 15-20 minutes for the task involving ambiguous traces and 10-15 minutes for all other tasks. All participants understood the behavior of specifications and could distinguish sensor errors from specification errors. Participants were also able to grasp the intended behavior of a specification both from the usage scenario and from the specification itself. As such, they used the timeline and playback widgets as a means for verifying intuitions about a specification's behavior rather than for exploring its overall behavior. Moreover, though overconfident participants initially declared a flawed specification to be correct in 6 of 50 tasks, they quickly changed their minds after comparing the timeline to their knowledge and intuitions about events. All participants were comfortable using Panoramic and several remarked that it was fun or "like a game". However, while they did not encounter any critical barriers to task completion, many participants faced recurring difficulties for which we have developed preliminary solutions (see Figure 9).

First, participants often checked the timeline for consistency with the events they knew occurred during Alice's week. In many cases the first question they tried to answer was "how many times was the event detected?". The timeline does not directly answer this question, so participants had to scroll through the week to count event occurrences. We addressed this problem by adding a count for each event beside that event's label at the left side of the timeline.

The task involving an under-specified event required participants to further constrain the specification by adding an object. This was difficult because the set of available objects was buried in the specification interface, leaving participants unsure of what objects were available. Moreover, without the ability to review traces for other Actors alongside the currently loaded trace, it was difficult to decide whether or not another Actor was relevant to an event. While this problem may be less critical when users reason about people and objects they know, we did introduce a new section to the legend at the top of the playback panel that shows other Actors which may be relevant to the event. By clicking the checkbox next to an Actor, users can load a new map panel that plays the trace for that Actor. The set of displayed Actors is currently chosen as those that are proximate to, or move in the same time window as the currently loaded trace. This is a reasonable compromise to the unscalable alternative of displaying every possible Actor because proximate or moving Actors are likely to be more relevant.

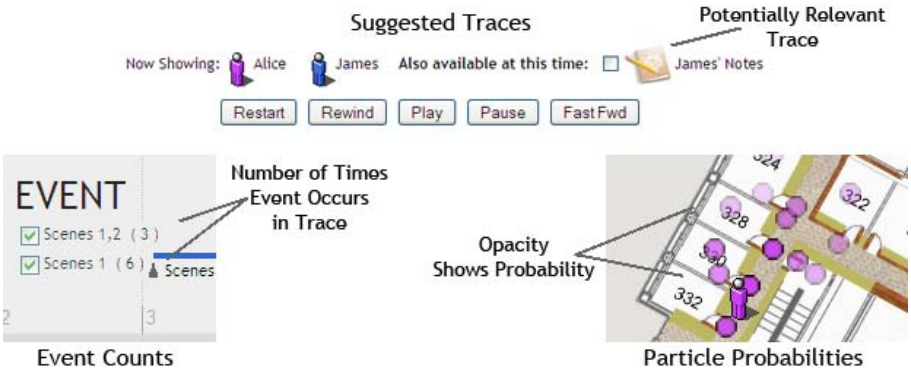


**Fig. 9. Enhancements to the timeline and playback widgets.**

Two additional recurring frustrations were voiced by participants when using the playback widget. First, they had difficulty understanding the visualization of the MStream as a set of particles. After explaining the particles as "Panoramic's guess at where a person or object is," participants were better able to understand but still had difficulty reasoning about sensor errors and missed detections as a result of ambiguity. As such, we changed our rendering of particles to include an opacity level that corresponds to a particle's probability. This helped the last 8 of 10 participants to identify sensor errors in the ambiguous trace 3-5 minutes faster than the 2 who did not have this feature available. A second difficulty was that participants felt it was difficult to correlate the trace playback with the events in the timeline. Although the timeline was animated to correspond to the trace, participants were uncomfortable looking back and forth between the trace and the timeline. One participant explained her difficulty with the playback widget by saying "it shows where Panoramic thinks the people are, but not what it thinks about the events, you have to keep looking back at the timeline to see the events, that's hard". This problem could be addressed in future work by arranging the timeline below the playback widget, or by embedding pop-ups in the playback maps that mark when and where events occur.

### 7.2 Limitations

Panoramic currently has several key limitations. First, while it supports specifications with Actor variables, verification of such events is tedious because users may need to review multiple sets of historical traces - one for each possible parameterization of the variables. Panoramic is also limited by its minimal support for debugging suggestions. While a list of potentially relevant Actors is useful, more intelligent suggestions could be made by assessing the how sensitive the event detection results are to slight changes in a specification. Finally, Panoramic's reliance on historical data may be problematic for events that seldom occur. Here it may be possible to automatically generate synthetic test data for a particular specification using techniques similar in spirit to recent work by Olston *et al.* [25]. Future work will address these and other limitations.

## 8 Related Work

Here we review and discuss a variety of end-user software engineering techniques for sensor systems. We focus on specification and verification of events, omitting discussion of techniques that specify event-triggered behaviors.

**Specification Languages** Event specification systems for end-users are difficult to design because they must lower the barrier to entry without compromising expressive power. One approach is to create a specification language with abstractions that represent high-level concepts in the target application domain. For example, early systems like PARCTAB [35], Stick-e Notes [26], and SPECs [17] used scripts to describe primitive location events. More recent work with Semantic Streams [40] and probabilistic context-free grammars [22] can detect some complex and even uncertain events in sensor networks. While these systems use data processing techniques similar to those in Cascadia, they expose languages that are not well-suited to end-users.

**Specification Interfaces** Several systems have made specification more accessible with graphical interfaces that declaratively specify events. EventManager [24] used four drop-down boxes to specify a small set of primitive location events. CAMP [33] specifies non-sequence (i.e., instantaneous) events with a magnetic poetry interface that answers the questions: who, what, where and when. The Topiary [20] design tool also specifies instantaneous events, but uses an interface with an active map and a storyboard. Panoramic is quite similar to iCAP [32], a visual interface for specifying spatio-temporal sequence events. However, CAMP, iCAP, and Topiary are all less expressive than Panoramic because they rely on custom-coded event detection modules instead of a flexible event detection engine like Lahar. Moreover, they do not explicitly support event detection over uncertain sensor data like Panoramic.

**Programming by Demonstration** Another approach is programming by demonstration (PBD), in which users supply example sensor traces to train an event detector. a CAPpella [8] is a PBD system that allows users to train a Dynamic Bayesian Network with labeled sensor traces. Apart from low detection rates, a CAPpella is difficult to use in our motivating scenarios because it requires 5 or more traces for training. Other systems focus on detecting simpler events with fewer examples and rapid feedback. For example, Crayons [9] and Eyepatch [23] enabled users to rapidly train visual classifiers using a demonstration-feedback loop. The Exemplar system [13] employs a similar loop featuring an algorithm that requires only one demonstration. Exemplar focuses on exposing an intelligible and editable visualization of its model, addressing the fact that automatically learned models are often inscrutable [5, 18]. These prior PBD methods become impractical for complex events that involve the simultaneous movement of people and objects.

**Verifying Specifications** Past work has shown that end-users must be able to verify that a specification works as intended [5, 18]. Verification identifies three broad categories of error: (1) syntactic errors that make specifications illegal or ambiguous, (2) semantic errors that make valid specifications behave in unexpected ways, and (3) sensor errors that cause event detectors to erroneously

detect or miss events. Most languages and declarative interfaces use interactive visual feedback (e.g., error flags, prompts for disambiguation) to cope with syntactic errors [20, 32]. Semantic errors are often identified by testing with sensor traces (as discussed below). However, both CAMP and Panoramic can reveal semantic errors in non-sequence events by generating high-level English descriptions - in Panoramic these are descriptions of Primitives. Most systems provide no support for identifying sensor problems beyond what may be inferred from detection errors. In contrast, Panoramic directly supports discovery of sensor errors by providing a visualization that correlates sensor data with detected (and missed) events.

**Trace-Driven Debugging** Test traces are commonly generated using either Wizard of Oz [20, 32], in which the user simulates sensor traces with a special interface, or demonstration [8, 13], in which the user enacts an event while recording it with sensors. Both of these techniques become prohibitively demanding for complex events. The Wizard of Oz approach also fails to capture the impact of uncertainty in real sensor data. Panoramic avoids these problems by using pre-recorded traces. Moreover, though other systems could adopt Panoramic's approach, they do not provide the support for archiving, exploration, and visualization of uncertain events and sensor data that Panoramic does. Debugging also requires that users correct erroneous specifications. This is a simple matter of modifying the specification in declarative interfaces like iCAP and Panoramic. It is much less straightforward in PBD systems [5], and may require that entirely new sets of demonstrations be recorded.

**Intelligible Context Models** Several papers have established and articulated the need for intelligible models of context [4, 6, 21]. A few systems have also explored support for intelligible context. Cheverst *et al.* [6] supported users with scrutable decision tree rules and context histories. The PersonisAD [2] framework allowed developers to access supporting evidence for context items. Dey and Newberger [7] support intelligibility in the Context Toolkit using Situation components that expose application logic to developers and designers. Panoramic adds to this body of work by providing an intelligible, scrutable context model for complex location events. Moreover, Panoramic contributes a context management system that directly copes with uncertainty using probabilities while not requiring users to explicitly specify probability thresholds.

## 9 Conclusion

In this paper we presented the design and evaluation of Panoramic, an end-user tool for specifying and verifying RFID events. Our design leverages and extends the Cascadia system and the Scenic tool in significant ways, and is informed by feedback from a formative study with 12 non-programmers. We also contributed the design of an interface for verifying complex location events that was motivated by problems observed in the formative study. We evaluated our verification interface with 10 non-programmer study participants and found that in spite of minor difficulties with the interface, all users were able to complete five representative verification tasks. Overall, we have presented a tool that satisfies

a growing need in a way that is accessible to end-users and which works in spite of inevitable sensor errors. Moreover, we demonstrated techniques that support intelligible context for applications that use complex location events.

## 10 Acknowledgements

The authors would like to thank Kayla Gould and Emad Soroush for their efforts in support of the design and evaluation of Panoramic. This work was supported in part by the National Science Foundation under grants CNS-0454394, CNS-0454425, IIS-0713123, and IIS-0812590; and by gifts from Intel Research and the University of Washington's College of Engineering.

## References

1. Amelior ORTracker: Orchestrate Patient Flow. `http://www.pcts.com/unified/ortracker.php`, 2009.
2. Assad, M. et al. PersonisAD: Distributed, Active, Scrutable Model Framework for Context-Aware Services. In *Pervasive 2007*, pages 55–72, 2007.
3. Bardram, J. E. The Java Context Awareness Framework (JCAF) – A service infrastructure and programming framework for context-aware applications. In *Pervasive 2005*, pages 98–115, 2005.
4. Bellotti, V. and Edwards, K. Intelligibility and Accountability: Human Considerations in Context Aware Systems. *HCI*, 16:193–212, 2001.
5. J. Chen and D. S. Weld. Recovering from Errors During Programming by Demonstration. In *IUI 2008*, pages 159–168, 2008.
6. Cheverst, K. et al. Exploring Issues of User Model Transparency and Proactive Behaviour in an Office Environment Control System. *User Modeling and User-Adapted Interaction*, 15(3-4):235–273, 2005.
7. Dey, A. and Newberger, A. Support for Context-Aware Intelligibility and Control. In *CHI 2009*, pages 859–868, 2009.
8. Dey A. K., et al. a CAPpella: Programming by Demonstration of Context-Aware Applications. In *CHI 2004*, volume 1, pages 33–40, 2004.
9. Fails, J. and Olsen, D. A Design Tool for Camera-Based Interaction. In *CHI 2003*, pages 449–456, 2003.
10. Garofalakis, M. N. et. al. Probabilistic Data Management for Pervasive Computing: The Data Furnace Project. *IEEE Data Eng. Bull*, 29(1):57–63, 2006.
11. Google Maps API - Google Code. `http://code.google.com/apis/maps/`, 2009.
12. Google Web Toolkit - Google Code. `http://code.google.com/webtoolkit/`, 2009.
13. Hartmann, B. et al. Authoring Sensor-Based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. In *CHI 2007*, pages 145–154, 2007.
14. Heer, J. et al. liquid: Context-Aware Distributed Queries. In *Ubicomp 2003*, volume 2864, pages 140–148, 2003.
15. Real-time locating systems 2009-2019. `http://www.idtechex.com/research/reports/`, 2009.
16. Knoll, S. et al. Viewing Personal Data Over Time. In *CHI 2009 Workshop on Interacting with Temporal Data*, Apr. 2009.

17. Lamming, M. and Bohm, D. SPECs: Another Approach to Human Context and Activity Sensing Research, Using Tiny Peer-to-Peer Wireless Computers. In *Ubicomp 2003*, pages 192–199, 2003.
18. Lau, T. et al. Why PBD Systems Fail: Lessons Learned for Usable AI. In *CHI 2008*, 2008.
19. Letchner, J. et al. Challenges for Event Queries over Markovian Streams. *IEEE Internet Computing*, 12(6):30–36, 2008.
20. Li, Y. et al. Topiary: A Tool for Prototyping Location-Enhanced Applications. In *UIST 2004*, pages 217–226, 2004.
21. Lim, B. and Dey, A. Assessing Demand for Intelligibility in Context-Aware Applications. In *Ubicomp 09*, pages 195–204, 2009.
22. Lymberopoulos, D., et al. A Sensory Grammar for Inferring Behaviors in Sensor Networks. In *IPSN 2006*, pages 251–259, 2006.
23. Maynes-Aminzade, D. et al. Eyepatch: Prototyping Camera-Based Interaction Through Examples. In *UIST 2007*, pages 33–42, 2007.
24. McCarthy, J. F. and Anagnost, T. D. EVENTMANAGER: Support for the Peripheral Awareness of Events. In *HUC*, volume 1927, pages 227–235, 2000.
25. Olston, C., et al. Generating Example Data for Dataflow Programs. In *SIGMOD 2009*, pages 245–256, 2009.
26. Pascoe, J. The Stick-e Note Architecture: Extending the Interface Beyond the User. In *IUI 1997*, pages 261–264, 1997.
27. Philly Hospital Uses RTLS to Track Patient Flow, Care and Training. `http://www.rfidjournal.com/article/view/4934/1`, May 2009.
28. Ré, C. et al. Event Queries on Correlated Probabilistic Streams. In *SIGMOD 2008*, pages 715–728, June 2008.
29. RTLS Providers Cite Strong Demand From Hospitals. `http://www.rfidjournal.com/article/print/4981`, June 2009.
30. Salber, D. et al. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *CHI 1999*, pages 434–441, 1999.
31. SIMILE Timeline. `https://simile.mit.edu/timeline/`, 2009.
32. Sohn, T. and Dey, A. iCAP: An Informal Tool for Interactive Prototyping of Context-Aware Applications. In *CHI 2003*, pages 974–975, 2003.
33. Truong, K. N. et al. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In *Ubicomp 2004*, Oct. 2004.
34. Vilamovska, A. et al. Study on the requirements and options for RFID application in healthcare. Technical report, RAND Corporation, July 2009.
35. Want, R. et. al. An Overview of the PARCTAB Ubiquitous Computing Experiment. *IEEE Personal Communications*, 2(6):28–33, Dec 1995.
36. Welbourne, E. et. al. Challenges for Pervasive RFID-based Infrastructures. In *PERTEC 2007*, pages 388–394, Mar. 2007.
37. Welbourne, E. et. al. Cascadia: A System for Specifying, Detecting, and Managing RFID Events. In *MobiSys 2008*, pages 281–294, June 2008.
38. Welbourne, E. et. al. Building the Internet of Things Using RFID: The RFID Ecosystem Experience. *IEEE Internet Computing*, May 2009.
39. Welbourne, E. et. al. Longitudinal Study of a Building-Scale RFID Ecosystem. In *MobiSys 2009*, June 2009.
40. Whitehouse, K., et al. Automatic Programming with Semantic Streams. In *SenSys 2005*, pages 290–291, Nov. 2005.