Hybridising events and knowledge as a basis for building autonomic systems

Simon Dobson, Lorcan Coyle and Paddy Nixon

Abstract—**Event-based systems are a popular substrate for distributing information derived from sensors to be used in driving adaptive behaviour. This paper argues that using events directly provides a poor model of context, and that a hybrid approach that uses events to populate and maintain a distributed knowledge base offers a more stable solution. The inherent uncertainties in both sensor data and reasoning imply that traditional knowledge-based system techniques applied to contextual systems be extended to deal with more uncertain reasoning.**

Index Terms—**context, uncertain reasoning, event-based systems, knowledge-based systems**

I. INTRODUCTION

Autonomic systems are intended to adapt to their environment in a way that optimises performance, robustness and other features without requiring extensive human intervention. The challenges arise from the need to deal with complex and uncertain information about the environment, and to match this to appropriate changes in system behaviour.

Events have long been used as a central architecture for weakly-coupled distributed systems. Experience has shown that event infrastructures can be made extremely scalable [1], but this scalability may not translate into an attractive programming paradigm for many applications. Complex control structures, for example, can be difficult to build using event systems, especially in the presence of event loss. Moreover pervasive and autonomic systems suffer from problems of "false" events generated by sensor inaccuracies, which further complicate event-based programming.

This paper argues that the low-level attractions of event systems are not matched by an attractive high-level programming model in pervasive and autonomic systems. It argues instead for hybridising event-based information transport with a knowledge-based representation and reasoning infrastructure for context information. This use of knowledge as a "stabiliser" on the system can lead to a significantly simpler programming problem when creating complex self-managing behaviours.

Section II reviews the way events can be used to model and transfer information in sensorised systems, highlighting some of the semantic issues in using events in the presence of noise. This analysis leads to an architecture that hybridises events for data transport with a distributed knowledge base for reasoning (section III). This does not directly solve the

problems caused by erroneous events, and section IV describes some techniques for data fusion, which are then deployed in section V to address a simple example scenario. Section VI describes some measures that would improve the effectiveness of the knowledge-base by attempting to model and allow adaptation of the system's confidence in the context data it receives from sensors. Section VII concludes with some directions for further exploration and validation of these ideas.

II. CONTEXT, EVENTS AND KNOWLEDGE

Designs for autonomic systems draw their inspiration from a number of sources. Prominent among these are biologically-inspired systems built around stigmergy or swarm intelligence, where simple individual responses to stimuli are aggregated to produces a global result [2]. At the other extreme are attempts to model adaptive systems in a closed-form way that allows more precise characterisation of their behavioural envelopes [3]. The former relies on ideas from control theory, while the latter draws more on pervasive computing, continuous mathematics and AI.

The context of a system captures the environment in which it operates, including all "additional" or "non-functional" aspects that, while not being "core" to the system's behaviour, nevertheless affect the way in which that behaviour should be optimised. Pervasive computing systems are good representatives of the class of adaptive systems whose adaptations are constrained by their surrounding environment.

The Context Toolkit [4] is the canonical example of programming pervasive applications based on events. Such systems consist of a number of adapters or contextors [5], each capturing some aspect of the environment such as the temperature, or the reading from a location sensor. The advantage of such systems is that it is straightforward to construct both the infrastructure and the adapters; the disadvantage is that they place a large load on the developer to build a sufficiently flexible decision-support system to drive adaptive behaviour.

A. Why events and adaptation don't match

To understand the problem of using events directly, consider the following scenario. Suppose there are two people, P and Q, and a room A. Two events are defined, enter(i,j) and leave(i,j), indicating that entity i has entered (or left, respectively) place j. These events are to be used to drive a system that will adapt its behaviour when P and Q are together in A. Angular brackets are used to denote event traces: given events e1, e2, and e3, <e1,e2,e3> is used to denote the sequence of events occurring in the

order given and `<e1,...,e2>` to denote `e2` occurring after `e1` with zero or more events in between. In the simplest model there are two possible event traces that can bring the desired situation about: `<enter(P,A), ..., enter(Q,A)>` or `<enter(Q,A), ...,enter(P,A)>`. On observing either of these event traces the system may adapt.

The problem, however, is that this approach is only stable given three key assumptions. The first is that events cannot be "counteracted" by other events. Suppose the event trace `<enter(P,A), ..., enter(P,B), ...,enter(Q,A)>` is observed. Does `P` entering `B` mean that `P` is no longer in `A`? – in other words, are `A` and `B` disjoint spaces? This cannot be definitively answered without an understanding of the spatial relationships involved. Furthermore in a open system new events might be introduced which interact with existing events in unforeseen ways. Introducing an event `leave(i,j)` (with the obvious intention) means that an event trace such as `<enter(P,A), ..., leave(P,A), ..., enter(Q,A)>` is also not a valid trigger for adaptation.

The second problem concerns triggers that rely on a correspondence between events. Suppose the event trace `<enter(P,A), ..., enter(Q,A), ..., enter(Q,A)>` is seen – what should be concluded? Should the second `enter(Q,A)` event be considered a duplicate, an error, or the start of another trigger for which a corresponding `enter(P,A)` event should be anticipated? This leads directly to the third problem. Event systems were developed from process algebra which in turn describes processes that might be termed exact: the events that occur are assumed actually to have occurred. The problem with many pervasive (and other) systems that have a close connection to the real world, for example by way of sensors, is that the processes they are engaged in are inexact: the events may be noise.

It seems intuitively likely, in the absence of any intervening `enter()` or `leave()` events, that the second `enter(Q,A)` event is a duplicate. However, knowing this implies an enormous amount of knowledge about the structure of the real world and the external semantics of events. Moreover, encoding this knowledge in a way that will be suitable for triggering an adaptation seems likely to be inordinately complicated for any realistic case.

Although simple, these cases would defeat most event-algebra systems (for example the one described by Hayton *et alia* [6]). This work hypothesises – without any formal justification – that the twin problems of openness and noise may render such algebraic systems intractable.

Perhaps the clearest way to view this problem is as follows. Event systems are by their nature "crisp", in the sense that an event is a binary occurrence. Autonomic systems require a significantly more fluid style of control to allow them to adapt to a range of signals and to handle the uncertainties in sensed and inferred information. This suggests that modeling such systems using event traces will be unsatisfactory.

The conclusion that may be drawn is that, while event systems may be scalable from a systems perspective, they are decidedly not scalable from a programmer's perspective.

The problem is that events are being used to two disjoint purposes. On the one hand, events are used to indicate that "something happened" (albeit with some uncertainty); on the other hand, event traces are being used as the system's model of the outside world. The former is a system-level issue that is handled well by events; the latter is a semantic-level issue that is not. If the two are decoupled, a hybrid system may be developed having the disadvantages of neither.

B. A more knowledge-driven approach

It may be observed that many adaptive systems decisions are phrased in logical terms: "when P and Q are in the room then ...". Techniques from knowledge-based systems might therefore be imported to drive adaptations when particular conditions are true. This may be performed on a per-consumer basis [7], but this passes much of the complexity of handling noise onto component programmers. A more global approach seems more attractive, for example Wang *et alia*'s use of RDF to represent knowledge [8]. Several programming techniques are then possible, including the use of truth-maintenance techniques to execute adaptation code when a predicate changes truth-state.

Such techniques face twin problems of uncertainty and noise. Most information derived from sensors is inherently error-prone, and sensors give rise to incorrect observations. To take one example, several authors have used RFID sensors to observe tags attached to people or artifacts. However, RFID sensors will fail to spot some tags, perhaps because it is moving too slowly to activate. They will also sometimes mis-identify tags because of interference. This means that a sensor-derived event may be incorrect or may be missed. It is easy to see why event traces are such an inadequate source of modeling.

However, it is possible to use a knowledge base as a stabiliser on the context model. Events must be treated as evidence for a fact, rather than as true Boolean values. Techniques such as Bayesian probability, fuzzy logic or Dempster-Schaffer evidence theory may then be used to combine individual pieces of evidence into a more confidently-held view of the environment, which can then in turn be used to drive adaptation. This helps combat the danger of a system changing state dramatically as a result of a single, erroneous event, since other already-accepted evidence can act as a counterweight. Adding more knowledge about the system (such as the behaviour of people in space) further increases this stabilisation effect.

RDF is attractive as an information model for a number of reasons. It provides for immediate, standard-based exchange of information between systems. As an open and extensible system it reduces the level of ontological commitment required of a developer, allowing new semantic elements to be introduced as required. However, this flexibility comes at a cost. Adding a new element is not unconstrained: the new element's relationship with existing elements must be clearly captured, and new reasoning pathways may be required to handle the new information provided. These issues may be mitigated to some extent if new information (from new sensors, for example) may be aligned with a known upper-level ontology.

The issue of reasoning in open semantic domains remains however an open area of research.

C. Programming hybrids

The above discussion leads to a hybrid model in which an event infrastructure is used to collect and distribute evidence for the state of the system's environment, with the evidence being used to populate a knowledge base that maintains levels of confidence (or uncertainty) about that environment. This in turn leads to three observations about programming in the presence of such a knowledge base.

The first observation is that all decisions are necessarily tentative. Uncertain reasoning approaches may allow a system to maintain an on-going level of confidence about its environment. Having a confidence interval makes such systems sensitive to small changes: a small change in evidence may cause the decision-making process to "tip". It remains the case, however, that many adaptation decisions are "crisp", so that the uncertain reasoning collapses to Boolean logic when the decision is made. This uncertainty means that there is a need to maintain one or more recovery strategies for any adaptation or decision the system makes, since each may need to be undone for at least two reasons: because circumstances change to cause a new adaptation, or because the additional evidence shows the initial adaptation to have been mistaken.

A second observation is that adaptations are not arbitrary: systems do not change from one behaviour to another, completely unrelated behaviour, but rather change within an envelope according to environmental changes. A core task for engineering autonomic systems is to ensure that all adaptations do indeed remain within the design envelope and do not take the system to unacceptable parts of the behavioural space.

Finally, while autonomic systems of this type can make use of significant bodies of existing AI research, the levels of noise and uncertainty, coupled with the degree of unsupervised operation required, do seem to pose genuinely novel challenges. There are several foundational innovations to be made in the logics and reasoning approaches used to describe autonomic adaptation, as well as in the way this reasoning is used to select adaptive behaviour. In particular, approaches that account for the entire system behaviour at once may have advantages over those which try to coalesce a number of individual independent adaptations. In a sense this is the difference between set theory and topology: that topological approaches may prove useful for both the analysing and programming adaptive systems.

III. A KNOWLEDGE-BASED HYBRID ARCHITECTURE

The above argument would tend to suggest an architecture in which events are used to *transport* data (to obtain scalability and flexibility) while knowledge-based techniques are used to *process* or *reason about* that data to generate the information needed to drive self-adaptation.

A. Architecture

This work is built within a framework called Construct, a fully-distributed and -decentralised context aggregation infrastructure [9]. Construct consists of a number of nodes that

Concept	Description
PC-ID	The identity of the terminal where the user is active.
User	The user that is currently logged in.
Time	The time of the reading.

TABLE I

CONTEXT DATA FROM AN ACTIVITY SENSOR

aggregate contextual data. Each node has its own data-store, and sensors register themselves with a node and inject data into it. There are three types of sensor: pure sensors that produce and insert data (usually from the real world); applications that query and consume context data; and aggregators that query data, perform some translation on it, and insert some new derived data. An event is modelled as anything which causes a sensor to insert a piece of context data into the data-store. Construct nodes gossip amongst themselves to maintain a global view of system context. All information is represented as *(subject, predicate, object)* triples using RDF [10] as the underlying data model, which allows easy interfacing to various technologies arising from the semantic web initiative. Applications therefore see a "soup" of knowledge derived both directly from sensors and indirectly from sensor-fusion algorithms.

All contextual data comes with at least two pieces of meta-data relating to their provenance: a reference to the sensor that inserted it; and the time it was inserted. The pieces of context data created by events are called *context instances*. Context instances are made up of a number of features or *concepts*, which can be viewed as simple attribute-value pairs. Context instances may contain special concepts that relate to the accuracy of the event. These concepts are loaded into a knowledge base, which is used to provide a more stable solution than the pure event-trace solution. This scheme will be demonstrated using an example scenario from a tag-based location detection application using Ubisense [11].

The knowledge base calculates a belief or certainty for each piece of context data in the knowledge base. When an application queries the knowledge base for a piece of data, it examines every source of relevant context data and returns the result that it believes is most accurate. It does this using Bayesian inferencing, by aggregating the confidences it has in the individual sources of data and selecting the piece of data with the highest level of belief. The success of this technique is dependant on having a good knowledge of the accuracies of sensors. It is also advantageous to have as many sources of context data as possible.

B. Knowledge Representation

A location-sensor is used to demonstrate the representation of context data. This is a sensor that has been developed in the Systems Research Group (SRG) in UCD Dublin. It is a daemon that runs on each computer in the SRG lab. It generates an event every time it detects user activity from either the mouse or the keyboard. Table I shows the concepts of a typical event and describes their meaning.

In event-trace terms, this event would look like: `activity(PC-1, waldo, ''23:28 22/03/2006'')`. This event says that the user `waldo` was seen accessing `PC-1` at time `''23:28 22/03/2006''`. Figure 1 shows a graph representation of the RDF that describes the "output" of this event. The "verbs" `hasID`, `hasUser` and `hasTime` are described by ontologies developed both internally and externally (such as those used in CoBrA [12], Gaia [13], and ASC [14]. Ontological descriptions are important because subsets of these data may be useful beyond the scope of activity sensation, and it should be possible to extract them from this event. Furthermore they must retain their semantics so that they are not misused. Techniques from the Semantic Web domain are used to "label" data with meaning that is retained even as data from different sources are recombined into other forms [15], [16].
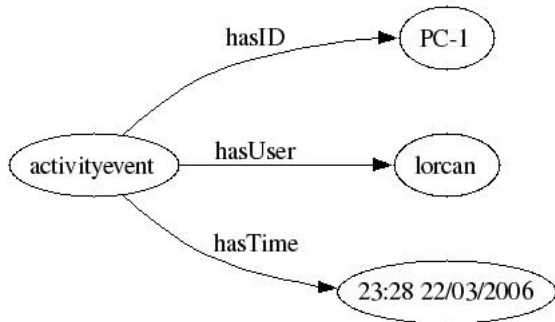

Fig. 1. Example output from the activity sensor

Another issue is the issue whereby different sensors will provide contextual data that refer to the same concept but require some translation to allow them to be compared directly. An example of this would be the location feature [17]. Here are three examples of location that come from sensors being used in the SRG: "waldo is at location $\{x, y, z\}$"; "waldo is giving a presentation in the boardroom at 2pm"; and "waldo is not logged into this PC". Each of these locations are correct, but there is no way of mapping or comparing them directly without defining an ontology for location that includes coordinate data, symbolic data and more complicated constructs (such as "waldo is not here"). Some semantic web technologies exist (such as OWL [18]) that make it possible to define relationships between pieces of data. By defining mappings between these data it is possible to translate between the values inserted by different sensors.

It should be noted that both `PC-1` and `waldo` refer to resources that may also have concepts attached to them. `PC-1` is a computer that has a physical location, and `waldo` is a human user that has attributes, such as a name, contact details, identity tag information (used for SRG's Ubisense sensors).

By defining these relationships using ontologies, and by defining statements containing verbs such as `PC-1 within RoomA` relationships are defined between two separate pieces of data, e.g., `RoomA is-a Symbolic-Space` and `Building is-a Symbolic-Space`. Using statements such as these an ontological tree can be defined that

contains the relationships between physical locations. Sensor developers must ensure that the data outputted by these sensors (i.e. the sensor readings) fits into this ontology. A definition of the relative positions of location data might look like Figure 2. Arrows in this tree refer to containment – so room `A` is contained within `Building`, and door `y` is contained by both room `A` and room `B`. This allows the context aggregator to amass pieces of location data at a specific granularity (say room or building granularity) that was originally produced by sensors at a higher level of granularity.
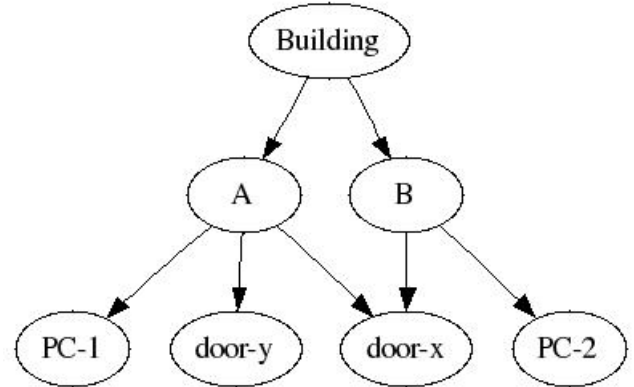

Fig. 2. A Location Tree

Semantic web techniques, strong ontological descriptions, and requiring that all sensor developers use the provided hooks to ensure that the context data they provide maps to part of the ontology, ensures that all contextual data should be in a standard format. This goes some way to solving the "babel problem", whereby every application uses its own independent ontology. In this way application developers need only to query against the ontology, and not have any awareness of specific sensors. The mappings that relate different granularities context data will be used to ensure that data can be converted into the correct granularity for consumption. All of this decouples the producers and consumers of context data and lowers the cost of entry for application developers.

Of course the next problem is that when an application developer makes a simple request such as "what room is waldo in now?" they will get multiple answers which may not all be equal (or correct). The first problem is that answers might be at different granularities; Clear et al. [16] describes how Construct uses ontologies and a query service to allow applications to query the data-store to get results at particular granularities. The second problem, that of accuracy, is esentially an unresolvable problem in pure event-based systems. The next section describes how a knowledge based approach may be used to resolve this problem, as well as the stability issues inherent in event-based systems.

IV. AGGREGATING CONTEXT

Most computing systems expect their inputs to be correct, at least at the level of not including erroneous information (although information may be missing). In section II it was noted that sensors do not have these properties, and moreover

generate data at different rates and with different precisions. A further characteristic is that events may not reported accurately by sensors – in fact it can safely be assumed that sensors will occasionally misbehave. This makes it difficult to reason about the system from its event traces, and introduces instabilities into any reasoning process which need to be damped.

A. Modeling inaccuracy

Overcome these stability issues is attempted by modelling inaccuracy and incorporating it into a knowledge-base. Construct stores all event messages in a distributed data-store. The knowledge-base is built upon this store. Events from different sensors will be able to insert data into the knowledge-base, which will attach a measure of its belief in the accuracy of that data. When applications seek to query the data, the result will reflect these beliefs, and the knowledge-base will have a certain *confidence* in its answer.

Construct allows the application developer to pose a query for a piece of contextual data. If that data is defined by one of the ontologies, the tools outlined in Section III-B are used to ensure that the resulting data is at the correct granularity. These concepts are then passed to the inferencing engine, which will return a single value with an estimation of its correctness. This inferencing is performed using an aggregation function. This function takes into account three pieces of data; *precision*, *decay* and *confidence*.

1) Precision: Precision relates to the physical accuracy of a sensor reading. With outdoor location systems, satellite based systems such as GPS should have a higher precision (approximately 2m) than network-based triangulation methods (up to 20m [19]). The burden of calculating precision should fall on the sensor developer as they should be capable of providing a more accurate estimation of their own precision than the application developer who consumes the data. Sensors should provide an estimation of their precision as a consequence of reporting an event. This may be a difficult estimation for a developer to make, and Section VI shows how it might be possible to overcome bad estimations automatically.

2) Decay: Decay attempts to capture the reduction in belief in a sensor reading as time passes since an event is reported. If readings are generated regularly then the most up-to-date report is likely to be the most accurate. The decay rate will be different for different sensors. Decay will be directly related to the frequency of readings. Evaluations will be needed to investigate this premise.

3) Confidence: Confidence reflects the belief of the knowledge base in a particular sensor reading. This is a more fluid concept than either precision and decay, and can change based on the knowledge-base's trust in the readings provided by a sensor. If one sensor is continuously different from all the others its importance in the aggregation function should be reduced. This altering by the knowledge base of the confidence parameter is discussed in more detail in Section VI. As a rule, confidence will be constant for all readings from an individual sensor, although it may change over time.

Context data are aggregated as follows: context data are first mapped to the granularity that will answer the query posed by an application. In the case of the location example the query is at the granularity of a room, so a number of data will be returned at this granularity. Each value gets a vote towards the determining of the correct value. However, these votes are not all equal. Votes are attenuated by the precision, decay and confidence values of the source data. The values for precision, decay and confidence will each lie between zero and one. This ensures that each piece of data, c gets to vote towards the returned value using the equation:

$$\text{Vote}(c) = \pi_c \times \delta_c \times \gamma_c$$

where π_c, δ_c, and γ_c refer to the precision, decay and confidence of value c respectively. If many votes come from an individual sensor they are averaged to prevent that sensor from overwealming the alternatives in the aggregation process. When all votes are counted, the datum with the highest vote is that which is returned. The accuracy of that reading will be related to the proportion of votes for that datum using the equation:

$$\text{Confidence}(c) = \frac{\text{Vote}(c)}{\sum_{i \in C} \text{Vote}(i)}$$

where c is the highest-ranked concept, and C is the set of all concepts that recorded votes.

Smoothing of context data (especially fast changing data) is performed implicitly by the decay parameter's part in the aggregation process. Data in Construct have expiry dates that are different for each datum. This expiry date is part of the meta-data that is provided by the sensor and managed by Construct. If multiple data exist with the same source, they are given votes proportional to the number of instances, for example if there are five pieces of data from a single sensor they are each given a fifth of a vote. Thus, smoothing is performed when aggregation is performed due to the taking into account of previous readings; decaying of readings over time places a bias in favour of more recent readings.

V. EXAMPLE SCENARIO

In order to demonstrate the aggregation of contextual data from multiple sources, an application scenario is described that, although simple, is sensitive to possible sensor errors and therefore benefits from a fusion-based approach. The activity sensor was introduced in Section III-B. Table II describes the activity sensor in more detail, as well as two other location sensors. These sensors provide location data at different granularities with different accuracies and decay coefficients. An example is used to demonstrate how location data provided by these sensors are aggregated into a single location result with an estimation of the error. This example uses the query "what room is waldo in now?".

These sensors are located in a physical space described in the map in figure 3. There are door sensors in door `x` and door `y`, Ubisense sensors throughout the building and activity sensors in both `PC-1` and `PC-2`. There is also an aggregation sensor that queries for (`x`, `y`, `z`) coordinate data (such as that generated by the Ubisense sensors) and converts it

Sensor Type	Description, Accuracy and Decay
UbiSense	Determines the (x, y, z) coordinates of an individual with a peak granularity of 30cm in 3D space. As these sensors report events frequently (as often as 5Hz), the decay parameter is very high
Door Sensors	Triggers an event whenever an individual passes through a door frame (it may be implemented using RFID). The granularity of this sensor is of the order of two rooms, since it unable to determine the direction the user is travelling in. These sensors have a very low rate of decay since these events are reported infrequently
Activity Sensor	Determines whether an individual is located at a computer by checking if they are logged-in and active at that terminal. This sensor acts at a sub-room granularity. These readings have a variable rate of decay, since events are triggered either when a user strikes a key or moves the mouse, or when there is a long period of inactivity

TABLE II

SOME AVAILABLE LOCATION SENSORS

using a defined mapping into room granularity. The semantic web tools described in Section III-B allow location data to be translated easily between different levels of granularity; in this example, location data are produced at room-level granularity.

Figure 3 shows an example set of location events that were sensed by the Ubisense, door sensors and activity sensors superimposed onto a physical map. The dotted line shows the exact trace of where user waldo travelled with letters marking the estimated location when each event was triggered.

These pieces of data are now entered into the knowledge-base's context aggregation algorithm. The data are shown in Table III. The values for confidence have been omitted; for now it is assumed that these values are constant for all sensors. This this assumption will be challenged later in section VI.

Source	Event	δ	π	Proportion	Vote
Door Sensor	a	0.05	0.8	0.5	0.02 (A)
Activity Sensor	b	0.9	1.0	1.0	0.9 (A)
UbiSense	c	1	0.8	0.2	0.16 (B)
UbiSense	d	0.8	0.8	0.2	0.13 (B)
UbiSense	e	0.6	0.7	0.2	0.08 (A)
Door Sensor	f	0.97	0.8	0.5	0.39 (A&B)
UbiSense	g	0.4	0.9	0.2	0.07 (A)
UbiSense	h	0.2	0.8	0.2	0.03 (A)

TABLE III

LOCATION DATA AT ROOM GRANULARITY

Figure 4 shows how votes are aggregated in this example. The door sensor had two half votes, one each for events a and f. The vote for event f (the right most arrow in the figure) has a decay of 0.97 and a precision of 0.8 therefore its vote is 0.39. All votes are aggregated and found to be 1.50 in favour of room A and 0.68 in favour of room B. Therefore the knowledge base will report that user waldo is currently located in room A with a confidence of 69% (69% = 1.50 ÷ (1.50 + 0.68)).

This prediction is actually incorrect, since user waldo is really in room B (he just crossed the boundary between room A and B). However, as the readings from the activity sensor fade due to decay, and further Ubisense data is inserted, the votes in favour of Room A will reduce and those in favour of B will increase until the knowledge base predicts the correct result. This lag in accuracy can be expected whenever boundaries in context are crossed and is a side-effect of smoothing.

VI. LEARNING CONFIDENCE

In the example no mention was made of certainty – the third measure of accuracy – other than to say that it was equal for all sensor readings and so did not take any part in the overall accuracy calculation. Confidence comes into play where the knowledge base *believes* a sensor is behaving in a way that suggests it is unreliable. This could occur due to a number of factors: the sensor is broken; the quoted precision value for the sensor readings is incorrect; or the sensor is misbehaving. On the other hand, the sensor may be behaving correctly, in which case the knowledge-base is not justified in suspecting the sensor of misbehaving (a false positive). This section outlines some ways that confidence might be *learned*.

It is important at this point to stress the difference between confidence and precision. A source of context data might be very accurate and yet generate spurious data. A common example of this that is when a user carries an incorrect Ubitag, or when they leave their Ubitag on their desk and proceed to walk around. Since this is not the sensor network's fault it should not have its precision value reduced, instead context data referring to this user should be discounted until the problem is resolved. It would be advantageous if this be done automatically and if confidence is restored once the problem is resolved.

Confidence can be learned in a supervised or unsupervised manner. One could use a single context sensor (that the system has confidence in) as a baseline or ground-truth by which to compare other readings (the supervised approach). Or one might compare subsequent readings and seeing if they are reasonable (an unsupervised approach). An example of this could be in a location-based sensor where readings vary wildly over short periods of time. This could indicate that a person is either moving very erratically or (much more likely) the system is behaving erratically (i.e. inaccurately). The next step in future work will favour the supervised approach and an evaluation will be performed of a location sensor network using camera tracking as the baseline.

When the system enters a new context (e.g., if user waldo enters a new room), but the knowledge base predicts a different context the system will attempt to correct it by reordering the confidence weights. This technique is similar to that used by Stahl and Gabel [20] and Coyle and Cunningham [21]. The correction or learning of confidence parameters should ensure that one sensor cannot bully the system into providing consistently incorrect readings. Obviously, where there is only one source of context data, it is pointless to learn confidence since the system is bound to accept the sensor's value as it is the only value available.

VII. CONCLUSION

This paper has presented a significant step toward a more usable form on context gathering service based on precision,
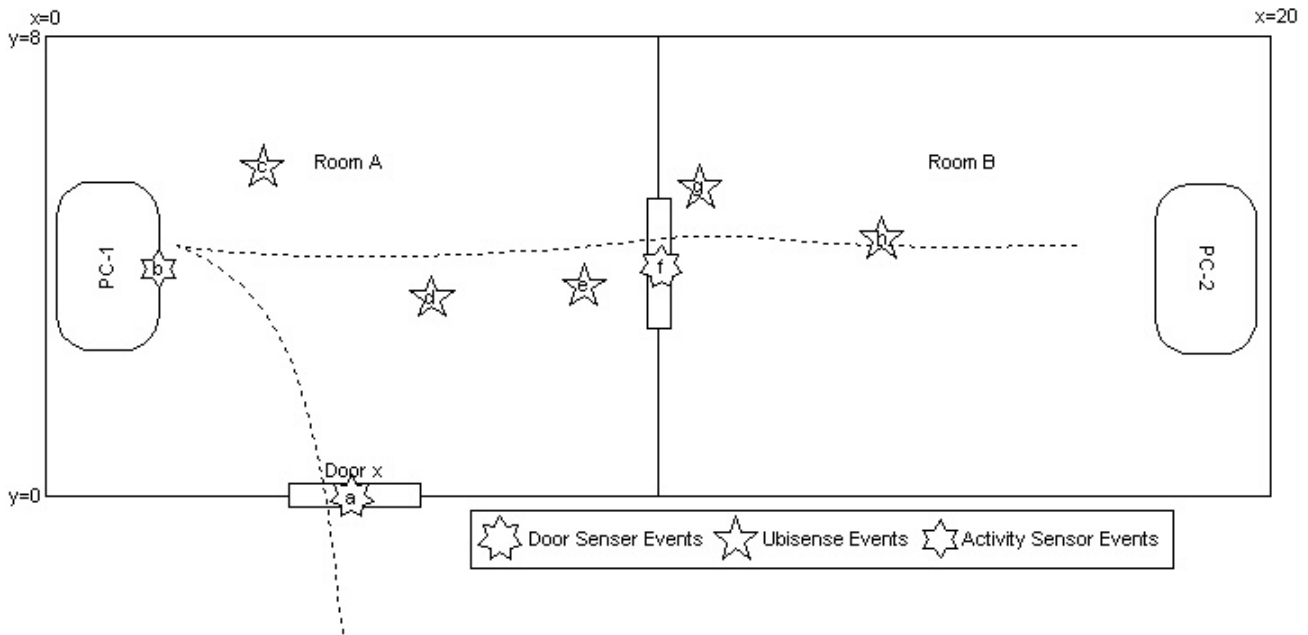
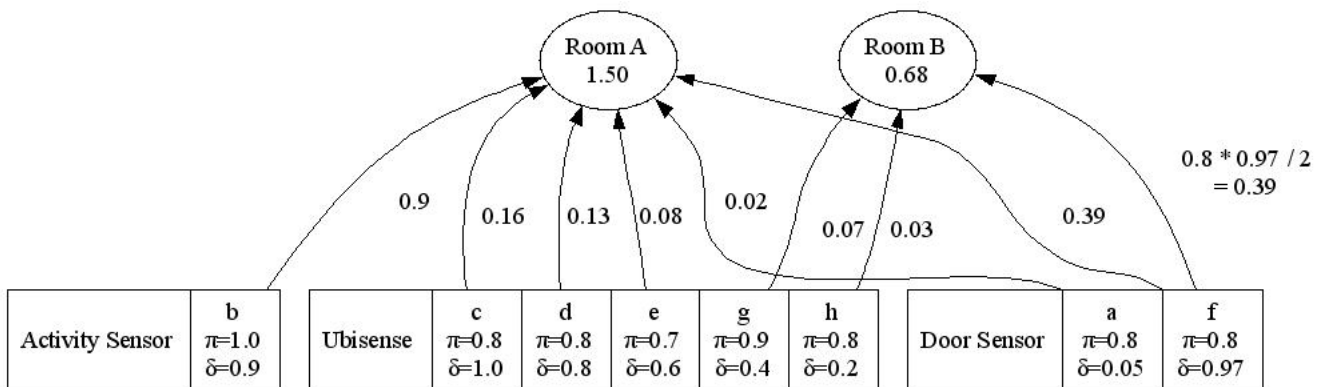Fig. 3.   Events superimposed on the physical map

Fig. 4.   Calculating votes for rooms A and B

decay and confidence processed by a probabilistic processing system. It proposes the hybridisation of two techniques for managing contextual information in adaptive systems. Events by themselves seem too fragile in systems with significant noise, but can be used effectively as a transport mechanism underlying a system of sensor fusion and uncertain reasoning. The reasoning approach needs to deal with the three factors of sensor accuracy, observation timeliness and system confidence in the results. Taken together these techniques can lead to very scalable context-adaptive systems expressed at a reasonable semantic level.

A combination of event-handling and knowledge management – distributed systems combined with AI techniques – offers a useful hybrid approach to modelling the context of adaptive systems. The knowledge base provides an important gain in the expressive power of the system in the face of erroneous events. The partial and tentative nature of all such knowledge means that programming techniques must make

extensive use of uncertain reasoning and other AI-derived techniques. It is important to move away from one-adaptation-at-a-time engineering to adopt a more holistic, closed-form approach to describing, analysing and programming adaptive behaviours.

The chief advantage of using a hybrid system is that it provides a cleaner and more stable representation of the state of the environment at a given time. However, as with all smoothing systems there will be times where temporal accuracy is sacrificed to this stability, which will become apparent where behaviour causes context to cross boundaries. This sacrifice may be necessary when dealing with real-world sensors, since sensor accuracy is not as certain as with theoretical models.

The way that adaptation alters the confidence parameters for a particular sensor is analogous to a top-down reasoning approach. By contrast, the way sensors alter the accuracy parameter is a bottom-up approach. By combining the two,

it may be possible to achieve a happy medium. This is a sensitive balance, and it will be instructive to produce some formal results that compare the abilities of both approaches individually and in concert.

As part of the further work in this area, a number of evaluations will be performed to investigate the ability of precision, decay and confidence to effectively capture and model the accuracy of event reports. This will be done by calculating the precision of a set of the location sensor network, by altering the decay parameter and by performing an evaluation of the aggregation function. By manipulating the data, and by entering spurious data, it will be possible to investigate the context aggregation process. The adaptation of the confidence parameter will be evaluated in the same way.

An interesting application of confidence learning is that it could be used for fault detection in sensor networks. In such a system the knowledge base could automatically issue an alert to sensor developers (or users) about problems in the system if it loses confidence in the outputs of individual sensors (or the network as a whole). This premise will be investigated further with evaluations in the future.

An important strand of future work is the integration of uncertain reasoning more closely into adaptive applications. Current programming languages do not lend themselves to truly pervasive uncertainty: we hypothesise that new language constructions and types will be required properly to leverage the flexibility introduced into systems by context fusion.

The Construct framework has been released under an open-source licence[1] and is being used to actively develop the themes outlined in this paper.

REFERENCES

[1] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, August 2001.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. Oxford University Press, 1999.

[3] S. Dobson and P. Nixon, "More principled design of pervasive computing systems," in *Human computer interaction and interactive systems*, ser. Lecture Notes in Computer Science, R. Bastide and J. Roth, Eds., vol. 3425. Springer Verlag, 2004.

[4] D. Salber, A. Dey, and G. Abowd, "The Context Toolkit: aiding the development of context-enabled applications," in *Proceedings of the ACM Conference on Computer-Human Interaction, CHI'99*, 1999, pp. 434–441.

[5] J. Coutaz and G. Rey, "Foundations for a theory of contextors," in *Computer-aided design of user interfaces*, C. Kolski and J. Vanderdonckt, Eds. Kluwer, 2002, vol. 3, pp. 13–34.

[6] R. Hayton, J. Bacon, J. Bates, and K. Moody, "Using events to build large-scale distributed applications," in *Proceedings of the 7th ACM SIGOPS European workshop on systems support for worldwide applications*. ACM Press, 1996, pp. 9–16.

[7] G. Biegel and V. Cahill, "A framework for developing mobile, context-aware applications," in *Proceedings of 2nd IEEE Conference on Pervasive Computing and Communications*, 2004.

[8] X. Wang, J. S. Dong, C. Y. Chin, S. R. Hettiarachchi, and D. Zhang, "Semantic Space: an infrastructure for smart spaces," *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 32–39, July–September 2004.

[9] G. Stevenson, L. Coyle, S. Neely, S. Dobson, and P. Nixon, "Construct — a decentralised context infrastructure for ubiquitous computing environments," in *IT&T Annual Conference, Cork Institute of Technology, Ireland*, 2005.

[10] O. Lassila and R. Swick, "Resource Description Framework model and syntax specification," World Wide Web Consortium, Tech. Rep., 1999. [Online]. Available: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[11] "Ubisense," http://www.ubisense.net/.

[12] H. Chen, T. Finin, and A. Joshi, "An Ontology for Context-Aware Pervasive Computing Environments," *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, vol. 18, no. 3, pp. 197–207, May 2004.

[13] A. Ranganathan, R. E. Mcgrath, R. H. Campbell, and M. D. Mickunas, "Use of Ontologies in a Pervasive Computing Environment," *Knowledge Engineering Review*, vol. 18, no. 3, pp. 209–220, 2004.

[14] T. Strang, C. Linnhoff-Popien, and K. Frank, "Applications of a Context Ontology Language," in *Proceedings of International Conference on Software, Telecommunications and Computer Networks (SoftCom2003)*, D. Begusic and N. Rozic, Eds. Split/Croatia, Venice/Italy, Ancona/Italy, Dubrovnik/Croatia: Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, Croatia, October 2003, pp. 14–18. [Online]. Available: http://www.kn.op.dlr.de/~strang/paper/softcom2003

[15] L. Coyle, S. Neely, G. Rey, G. Stevenson, M. Sullivan, S. Dobson, and P. Nixon, "Sensor fusion-based middleware for assisted living," in *Proc. of 1st International Conference On Smart homes & heath Telematics (ICOST'2006) "Smart Homes and Beyond"*. IOS Press, 2006, pp. 281–288.

[16] A. K. Clear, S. Knox, J. Ye, L. Coyle, S. Dobson, and P. Nixon, "Integrating multiple contexts and ontologies in a pervasive computing framework," in *Contexts and Ontologies: Theory, Practice and Applications*, Riva Del Garda, Italy, August 2006, pp. 20–25.

[17] S. Dobson, "Leveraging the subtleties of location," in *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*. New York, NY, USA: ACM Press, 2005, pp. 189–193.

[18] D. L. McGuinness and F. van Harmelen, "Owl web ontology language overview," World Wide Web Consortium," W3C Recommendation, February 2004.

[19] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit, "Place lab: Device positioning using radio beacons in the wild," in *Proceedings of PERVASIVE 2005, Third International Conference on Pervasive Computing*, Munich, Germany, 2005.

[20] A. Stahl and T. Gabel, "Using evolution programs to learn local similarity measures." in *Case-Based Reasoning Research and Development, 5th International Conference on Case-Based Reasoning, ICCBR 2003, Trondheim, Norway, June 23-26, 2003, Proceedings*, K. D. Ashley and D. G. Bridge, Eds. Springer, 2003, pp. 537–551.

[21] L. Coyle and P. Cunningham, "Improving recommendation ranking by learning personal feature weights," in *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004 Madrid, Spain, August 30th through Sep 2nd, 2004, Proceedings*, P. A. G. Calero and P. Funk, Eds. Springer, 2004, pp. 560–572.

[1]Available online at: http://www.construct-infrastructure.org