On the Impact of the Temporal Features of Sensed Data on the Development of Pervasive Systems

Graeme Stevenson
School of Computer Science
University of St Andrews, UK
gs@cs.st-andrews.ac.uk

Juan Ye
School of Computer Science
University of St Andrews, UK
ye@cs.st-andrews.ac.uk

Simon Dobson
School of Computer Science
University of St Andrews, UK
sd@cs.st-andrews.ac.uk

ABSTRACT
In this position paper we argue that the temporal features of sensed data present a number of fundamental challenges to the development of systems software and programming approaches for the development of pervasive systems, that are yet to be addressed. After an overview of these features, we discuss their impact on the design of infrastructure to support data storage, data access, and reasoning on sensed information, and on approaches to programming with temporal data.

1. INTRODUCTION
It is commonly accepted that pervasive systems – or more generally systems that work with sensed data – present a number of fundamental challenges that are not found in the development of traditional distributed systems [1].

Over recent years, many successful projects have investigated some of these issues: from the need for a common language to represent heterogeneous data sources [2], to abstracting data from its raw form to higher-level concepts [3], and to capturing and categorising the uncertain nature of such data [4].

However, another aspect of sensed data that we believe impacts the development of pervasive systems as much as any of the above is its temporal features. This covers several aspects involving time: from when data is generated and how frequently, to how long the values produced can be expected to be an accurate reflection of the state of the real world.

By forming a more complete understanding of these features and the challenges they present to all levels of pervasive systems development, we believe the design of software and programming abstractions required to store, access, and work with sensed data can be better informed.

We begin in Section 2 by categorising the set of temporal features exhibited by sensed data, followed in Sections 3 and 4 by a discussion of the challenges and opportunities that these features present to the design of infrastructure and programming-level support for pervasive systems. Finally, we conclude with a summary and outline of future work in Section 5.

2. TEMPORAL FEATURES OF SENSED DATA
The role of *temporal information* in the modelling of sensed data is often simplified in the design of context models, systems software, and application APIs. A typical data model sees the value of a sensor reported along with a timestamp indicating the point in time at which it is generated. This timestamped value is then used to update some form of relational model, where a set of entities (people, places, objects, etc.) are associated with a set of timestamped attribute-value pairs.

In this section we explore the temporal features of sensed data to demonstrate why this approach is insufficient for the demands of pervasive systems development.

2.1 Moment of Assertion
Timestamping is a ubiquitous technique in software development for representing the point in time at which data is generated and added to a system. In pervasive systems development, we traditionally use a timestamp to indicate when an attribute of an entity is asserted [5]. For example, to represent the fact that Bob is located in the living room of a house, we might write a statement of the form ⟨`bob locatedIn livingRoom`⟩, where `bob` and `livingRoom` are entities in a system, and `locatedIn` is a unidirectional relationship connecting them. To indicate when this statement was asserted, we qualify it by using a timestamp. For example, the statement:

```
⟨⟨bob locatedIn livingRoom⟩
   at [2010-03-01T09:00:00Z]⟩
```

is interpreted as stating that Bob was located in the living room as 9am on the 1st of Match 2010.

2.2 Temporal Relevance
Despite its universality as a modelling concept, systems that only timestamp data are implicitly restricted to the assertion of current state. That is, in previous example of Bob's

location, the timestamp has an implicit dual meaning: it is both the time at which the statement was asserted in the model and the time at which the statement ⟨`bob locatedIn livingRoom`⟩ should be interpreted as being *true*.

Although in many systems the value of these two temporal properties never vary or the variation is not important, being able to differentiate between the two provides a facility for the modelling of both historic and predictive state. There are many instances where this is useful, but here we restrict ourselves to two examples: summarising a high volume of sensor readings over a period of time, and representing calendar data describing probable future events. Consider the situation where a positioning system in the University of St Andrews reports Bob's location with a frequency of ten seconds. While the precise location of Bob may be useful to applications operating in real time, *posteriori* it is less so. However, Bob's general whereabouts may still be useful to an application at a future date, so we may choose to abstract and summarise this data, rather than remove it entirely. We may do this by the *post hoc* assertion of Bob's location over a number of *time intervals*. For example:

```
⟨⟨⟨bob locatedIn StAndrews⟩ at
  [[2010-03-01T09:00:00Z] - [2010-03-01T12:00:00Z]
  [2010-03-01T13:00:00Z] - [2010-03-01T17:00:00Z]]⟩
    assertedAt [2010-03-02T00:00:00Z]⟩
```

is interpreted as stating that Bob was located in St Andrews between 9am and 12pm, and between 1pm and 5pm on the 1st of March 2010, and that this information was asserted at 12am on the 2nd of March 2010.

Similarly, if we wish to represent information *a priori*, for example about future events in a calendar application, we can write:

```
⟨⟨⟨bob locatedIn Dublin⟩ at
  [2010-03-23T00:00:00Z] - [2010-03-27T23:59:59Z]⟩
    assertedAt [2010-03-02T00:00:00Z]⟩
```

which is interpreted as stating that on the 2nd of March 2010, we expect that Bob will be in Dublin from the 23rd to the 27th of March 2010.

Note that the distinction between the temporal relevance of the data, and the timestamp denoting the instant at which it was asserted is what allows us to determine if data is historic or predictive, irrespective of the point in time at which the data is later used. The assertion time can also be used to determine the most recent prediction of a future state if multiple predications are present in the model.

2.3 Sample Rate

The rate at which sensors sample the real world is another temporal property of sensed data. This information, which is available from the technical documentation of a sensor or set by the engineer responsible for its installation, is typically represented as a fixed duration. However some sensors may require more complex descriptions of their operation. For example, the sampling period of a water quality sensor may increase if a trend towards some critical condition is detected.

Such information usually forms part of a meta-model, that is as a property of a sensor description rather than a property of the data it produces. For example:

```
⟨ubisenseSensor frequency 200ms⟩
```

states that the sampling period of a Ubisense positioning sensor is 200ms.

2.4 Predicted Rate of Change

Although the sample rate of a sensor tells us when a sensor will (or may) report new data, it provides no indication of the likelihood of a change in the reported state. For example, the coordinate location of a person who is walking is likely to change at the rate of under a second, while a representation of their location at a higher granularity, such as a region of a city or country will change less frequently: perhaps over hours, days, or months. Indeed, little information is truly static; even a person's name, a building's structure, and a country's geography is susceptible to change [6].

Information about the *dynamics of data* comes from common sense or expert knowledge of a domain, or may also be obtained through the application of machine learning techniques. A more sophisticated approach involves inferring this information from the relations between data in the model. For example, if a meeting is scheduled to last 30 minutes the predicted rate of change for the location of the meeting participants could be implemented as a fuzzy function on the meeting time. There may also be types of data for which this knowledge is impractical or impossible to acquire. For example, if its dynamics are dependent on human or environmental features that cannot be modelled or predicted with any degree of accuracy.

There are several options for representing expected rate of change of data. A simple approach may be to tag a statement with a duration over which it is expected to remain stable, while a more complex approach may represent this information as a set of duration-probability pairs, indicating the likelihood that the value will change over several durations. For example:

```
⟨⟨⟨bob locatedIn livingRoom⟩ at
  [2010-03-01T09:00:00Z]⟩
    assertedAt [2010-03-01T09:00:00Z] ;
    predictedRateOfChange [[1s, 0.1%]
                           [1m, 1%]
                           ...
                           [1hr, 82%]]⟩
```

allows us to interpret the likelihood of Bob's location changing within the next second, minute, . . . , and hour as 0.1%, 1%, . . . , and 82% respectively. The drawback of this approach is that the set of granularities (e.g., seconds, minutes, hours, or years) are an arbitrary choice. However, we observe that these duration-probability pairs are essentially samples of the probability distribution over the rate of change of the variable. Therefore, if the cumulative distribution function (CDF) of the rate of change of the variable is known, a set of samples could be taken at a number of predefined percentages (e.g., 5%, . . . , 50%, . . . , 95%) to simplify inspection of the data. Better still, the CDF could itself be modelled (e.g., using MathML [7]). Although this approach increases modelling complexity (and the cost to evaluate a CDF at runtime may be undesirable), it supports the evaluation of the probability of a value remaining the same as when it was observed at any future time point.

In the context of predictive data, an orthogonal extension to this model is to also represent the dynamic nature of a *prediction*: i.e., both the expected rate of change of the predication and of the predicted data can be asserted as part of the model.

2.5 Summary

This section has outlined four key temporal features of sensed data: its *moment of assertion*, *temporal relevance*, *sample rate*, and *predicted rate of change*. In the following sections, we examine the impact of these features on the design of infrastructure-level support and programming approaches for the development of pervasive applications.

3. THE IMPACT ON THE DESIGN OF INFRASTRUCTURE SUPPORT

In this section we explore how the temporal features of sensed data can be used to inform the design of infrastructure-level support for pervasive applications. For the purposes of this paper, we restrict discussion to the scenario of a distributed middleware in a university campus environment; however, the examples we present are generally applicable to other environments.

3.1 Data Storage and Distribution

The efficient storage of data is a primary concern of middleware in a data-rich environment. In our university campus example, we can conceive of many heterogeneous data sources providing information about people, places, objects, the environment, and events taking place.

As the amount of generated data increases, it becomes impossible to hold it all in memory simultaneously. A typical solution to this problem is to discard the oldest data from memory as the maximum capacity is approached. However, it does not necessary follow that the oldest data stored is that which is least useful. As a result, the situation arrises where we need to periodically re-assert relatively static data, such as a building layout, in order that it remains in the model.

One solution to this problem is to incorporate rules in the middleware design to treat different types of data differently (e.g., to state that building layout information should never be deleted). However the major drawback of this approach is that *a priori* knowledge is required about the importance of all types of data that may be added to the system, which is unrealistic. Another possibility is to distinguish between profiled data and sensor data at the point it is added: perhaps by storing them in separate repositories and preferring the removal of sensor data over profiled data.

A better approach, and one that can be handled in a generic fashion, is to use information about the sample rate and predicted rate of change of data to select candidate data to be discarded. For example, prioritising the discard of data that is asserted frequently, but has a slow rate of change (e.g., ambient temperature) over data that is frequently asserted but changes rapidly (e.g., a user's coordinate location). At the other end of the spectrum, and therefore unlikely that it should ever need to be deleted, is relatively static data that is infrequently asserted (e.g., building layouts)[1].

An altogether different approach is to treat the data store of the middleware as a repository only for highly dynamic data. In this scenario, data that changes infrequently is written to disk and is only fetched and integrated with the highly dynamic data as and when it is required by applications. Pointers to the external sources of data (e.g., URLs) are maintained by the middleware in order that this integration process can be done transparently from the perspective of the application developer. As only relatively static data is stored externally, the load on the servers in which it is contained will be relatively light compared to that of the main repository (as we can presume that application requests for such data need only be sent infrequently). The middleware developer can therefore focus on optimising storage of and access to the highly-dynamic data generated by sensors.

Additionally, in distributed middleware (e.g. Construct [8]), adopting such a strategy reduces the amount of data to be transmitted between nodes to maintain consistent state.

3.2 Reasoning Strategies

The process of reasoning on a large data model, or any size of data model using a large number of axioms can be both time consuming and a performance bottleneck. However, knowledge about the dynamics of data can play a role in determining an appropriate reasoning strategy to adopt. For example, we may choose to reason on relatively static data as it is generated, adding the inferred knowledge directly to the model. Whereas with highly dynamic data we may decide to perform reasoning only at the point when an application query is executed, and to restrict the reasoning to the smallest amount of volatile data that will produce a correct answer for a given query.

When incorporating reasoning capabilities into a middleware design, a generic scheme to optimise performance can be devised to partition, reason on, and integrate data over several stages, where the stage at which any given data is reasoned on depends solely on its temporal properties.

3.3 Data Summarisation

As the temporal model is orthogonal to the data model, it is possible to design a generic approach (or at least a partially generic approach) to the problem of summarising and archiving sensor data. For example, a rule to trigger

[1]Note that if any data falls into the high rate of change, infrequently asserted category then this probably indicates a gap in sensing infrastructure that needs to be addressed.

the summarisation of location data may look like:

```
summarise ⟨?u locatedIn ?l⟩ after 2 hours:
  max: 10hr intervals
```

which states that all statements about the location of entities in a model should be summarised using intervals of up to 10 hours in length, two hours after each statement is asserted. Other summarisation strategies, including selecting a maximum, minimum, or average value (e.g., for summarising temperature readings), or invoking a custom operator, may be applied similarly. After data has been summarised, it may be discarded or archived on disk for future use.

Inspecting the temporal properties of application queries can also be a prompting to automatically adjust data summarisation rules. For example, if applications only query for current state, the middleware may adapt to increase the rate at which data is archived (improving the performance of queries over the memory-held state). Conversely, if an application regularly issues queries for historic data, then the rate of summarisation can be slowed down, or previously archived data may be reintroduced into the model.

4. THE IMPACT ON PROGRAMMING APPROACHES

We have discussed the temporal features of sensed data and the ways in which these features provide opportunities and challenges to infrastructure-level support for managing the lifecycle of data. In this section we turn our attention to application programming, and the typical queries an application might ask involving temporal semantics. For example:

- Where was Bob at 3pm on the 3rd of January 2010?
- What was Laura's state at 5pm yesterday?
- Who was present at the most recent meeting attended by Ric?
- When did Alice first travel to Italy?
- What was the duration of George's stay in St Andrews today?
- Where will Sarah be next Tuesday?
- How often does Claire travel to Glasgow?
- What usually happens before Nick cooks dinner?
- What is the average duration between Tom waking up and leaving the house?

We briefly examine some issues relating to the design of APIs and implementation considerations regarding these queries, and propose an approach for working with predicted state.

4.1 API Design

Temporal design patterns [9] are one approach to designing APIs that can work with time. To support the query *where was Bob at 3pm on the 3rd of January 2010?* the *Temporal Property* pattern can be used to provide two implementations of the necessary method: one to query for current state as normal (e.g., `bob.getLocation()`) and another that takes an instant of time as a parameter to support querying for historic state (e.g., `bob.getLocation(Time t)`).

When we are interested in several temporal properties of an object, as with the query *what was Laura's state at 5pm yesterday?*, the *Snapshot* pattern can be used to provide a copy of an object's state at a particular instant in time. This design pattern is realised as a method of the object that takes as a parameter the time instant of interest (e.g., `laura.getSnapshot(Time t)`).

As the temporal semantics of queries increase in complexity, a correspondingly complex API is required to work with temporal concepts like *most recent*, *first*, *duration*, *before*, *yesterday*, and *usually* to name but a few. For example, in evaluating the query *who was present at the most recent meeting attended by Ric?*, the first step is to find the most recent meeting that Ric attended. One possible design might adapt the *Temporal Property* pattern to accommodate this (e.g., `ric.getMeeting(MOST_RECENT)`), while another conceivable route to this information is via a utility class responsible for handling queries about meetings (e.g., `Meetings.byParticipant(ric, MOST_RECENT)`). Overuse of the former approach results in excessively overloaded methods and bloated APIs, while the latter approach violates good object-oriented design practice and requires the introduction of multiple utility classes to handle similar queries.

We expect that the above problems only represent the tip of the iceberg. A detailed exploration of the issues involved in programming with temporal data is required.

4.2 Implementation Considerations

Programming languages are designed to model and manipulate a single state. That is, the values of variables may be updated over time, but there is no provision to access the history of a variable. As a result, we require specialised data structures to store data with temporal semantics, and algorithms to query these data structures efficiently.

Tappolet et al. [10] have demonstrated the use of a tree-based temporal index to perform efficient time-point querying of temporally-qualified data in an RDF data model [11]; that is, selecting data whose validity is asserted to be within a given period of time. However, the approach described performs poorly where the temporal period associated with each assertion in the data model is distinct, as is the general case with sensed data. The development of similar algorithms optimised to the temporal characteristics of sensed data is one area requiring further research.

More complex queries require different strategies if they are to be answered efficiently. For example, consider the evaluation of the query *who was present at the most recent meeting attended by Ric?*. One approach to evaluating this is to generate time-point indexes only over certain "key" types of data (e.g., "meeting events"), thereby decreasing the query's search space. A more targeted approach is to index all meetings by person identifier and sort them by time. However, the more optimised an index is to a particular query, the heavier the reliance on guessing the likelihood of specific queries to be executed. The tradeoffs between more generic and targeted optimisations, and between performance and the resources required to store and maintain multiple indices in memory needs further exploration.

4.3 A Language-level Construct for Working with Predictive Data

In Section 2 we described how it is possible for sensors to assert predictive data; for example, calendar data that provides information about the expected future locations of a person. A fundamental question to be addressed is how we deal programatically with the situation where an action is taken based on a particular assumption about the state of the world that later turns out to be incorrect[2].

One possible approach, influenced by *design by contract* programming is to declare the assumptions upon which a block of code is to be executed. A second block of code can then be defined to mitigate the action of the first if we later find out that an incorrect assumption was made. For example:

```
⟨bob locatedIn Dublin⟩@??/??/2010:
   hotel = bookHotel(...) ;
   rsvr = makeRestaurantReservation(...) ;
mitigate:
  hotel.cancelBooking() ;
  phoneAlert('cancel restaurant booking',
             rsvr.details()) ;
```

The above pseudo-code describes a generic action to book a hotel and restaurant for each day Bob is in Dublin during 2010 (data that may be asserted by his calendar). If Bob's schedule changes, the *mitigation block* associated with operation takes steps to cancel the hotel booking and to inform Bob to call the restaurant to cancel his reservation.

Although this is a simplified example, it indicates a new form of semantics we require be supported within a programming language where specifying actions based on potentially unsound data is a normal rather than exceptional case.

5. CONCLUSION AND FUTURE WORK

In this paper we argued that the temporal features of sensed data present a number of fundamental challenges to the development of architecture- and programming-level support for pervasive systems that are yet to be addressed. After outlining these features, we discussed their impact in the areas of data storage, reasoning, and approaches to programming pervasive applications.

We are currently in the process of extending our earlier work, *Ontonym*, which developed a set of ontologies describing high-level concepts in pervasive environments [12], with these temporal semantics. Based on this we are designing a middleware for working with sensed data in which we will investigate further and apply some of the approaches we have outlined within this paper.

[2]We note that the inherent uncertainty associated with sensed data makes it likely that code will often be executed based on incorrect assumptions about the state of the world.

6. REFERENCES

[1] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowski. Challenges: An Application Model for Pervasive Computing. In *Proceedings on the 6th International Conference on Mobile Computing and Networking*, pages 266–274. ACM Press, 2000.

[2] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *First Annual International Conference on Mobile and Ubiquitous Systems*, Boston, MA, USA, August 2004.

[3] Guanling Chen, Ming Li, and David Kotz. Design and Implementation of a Large-Scale Context Fusion Network. In *The First Annual International Conference on Mobile and Ubiquitous Systems*, Boston, MA, USA, August 2004.

[4] Juan Ye, Susan McKeever, Lorcan Coyle, Steve Neely, and Simon Dobson. Resolving Uncertainty in Context Integration and Abstraction. In *Proceedings of the 5th international conference on Pervasive services*, pages 131–140, Sorrento, Italy, July 2008. ACM Press.

[5] Phillip B. Gibbons, Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan. IrisNet: An Architecture for a Worldwide Sensor Web. *IEEE Pervasive Computing*, 2:22–33, 2003.

[6] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In *International Conference on Pervasive Computing, Zurich, Switzerland*, 2002.

[7] W3C. Mathematical Markup Language (MathML) Version 2.0 (Second Edition). http://www.w3.org/TR/MathML2/, Last accessed, March 2nd 2010.

[8] Lorcan Coyle, Steve Neely, Graeme Stevenson, Mark Sullivan, Simon Dobson, and Paddy Nixon. Sensor Fusion-Based Middleware for Smart Homes. *International Journal of Assistive Robotics and Mechatronics (IJARM)*, 8(2):53–60, June 2007.

[9] Martin Fowler. Patterns for things that change with time. http://martinfowler.com/ap2/timeNarrative.htm, Last accessed, March 2nd 2010.

[10] Jonas Tappolet and Abraham Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web*, pages 308–322, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] W3C. Website for the RDF primer. http://www.w3.org/TR/rdf-primer/, Last accessed, March 2nd 2010.

[12] Graeme Stevenson, Stephen Knox, Simon Dobson, and Paddy Nixon. Ontonym: A Collection of Upper Ontologies for Developing Pervasive Systems. In *Proceedings of the Workshop on Context, Information And Ontologies*, Heraklion, Greece, June 2009.