Integrating Multiple Contexts and Ontologies in a Pervasive Computing Framework [1]

Adrian K. Clear and Stephen Knox and Juan Ye and Lorcan Coyle and Simon Dobson and Paddy Nixon [2]

Abstract. There is a commonly accepted need for contexts and ontologies to describe the vast amounts of data that are available to pervasive computing applications. Existing contexts and ontologies are either much generalised, very application specific, or inflexible. An integrated approach is required in which new concepts can be added and related to existing ones transparently. This paper describes a novel approach to the design of a set of contexts and ontologies for context-aware pervasive computing systems. It describes a *Query Service*, that lies between applications and contextual information, which complemented by the contexts and ontologies, offers a more powerful query answering service to application developers than is currently available.

1 Introduction

Pervasive systems are interactive systems, whose behaviour must adapt to the user's changing tasks and environment using different interface modalities and devices [8]. In order to be able to adapt to its environment, the pervasive system's applications and the environmental sensors must have a common understanding of the contextual information. For this purpose, contexts and ontologies are vital. We view an ontology as an explicit modelling of the fundamental concepts of a domain that may be shared and reused. A context is an explicit model of the secondary concepts in a domain. It is more specialised than an ontology but can still be shared and reused.

To date, most ontologies for pervasive systems have been developed in a top-down manner in which the main focus is on application semantics. This leads to ad-hoc models which are neither extensible nor support interoperability [12]. On the contrary, they should be flexible in their design to support a wider range of applications and environments. Moreover, the current approach to modelling contextual data is to give it a single representation in contexts and ontologies. However, it is evident that sensors acquiring conceptually equivalent data provide different representations of such because of their nature; issues such as accuracy and heterogeneity necessitates that the data provided by these sensors are represented differently. At a common level of abstraction these representations are conceptually equivalent. The need to incorporate such relationships into the design of contexts and ontologies should be recognised and is thus the primary focus of this paper. Dealing with this issue at design time can be instrumental in the run-time inference of unknown facts from known contextual data.

[1] This work is partially sponsored by Science Foundation Ireland under grant numbers 05/RFP/CMS0062 and 04/RP1/I544. Adrian Clear is funded by a joint IBM/IRCSET EMBARK scholarship.

[2] The authors are with the Systems Research Group, School of Computer Science and Informatics, UCD Dublin IE (email adrian.clear@ucd.ie)

Contextual data can be viewed as being part of a spectrum where data modelled by ontologies lie at one extreme, data modelled by contexts lie somewhere in the middle, and data without an explicit model lie at the other extreme. In an effort to clearly illustrate this spectrum, we propose the concept of a *semantic sphere* of pervasive system data (see figure 1). In the semantic sphere we define a set of fundamental ontologies for pervasive systems. We call this set the *core* ontology. The core ontology describes the principle concepts in a pervasive computing environment – *who*, *where* and *when*. More precisely, these are: the entities that are in the environment (people, sensors, etc.), the locations of interest, and the times of interest, respectively. All remaining data are viewed as being somewhat less general and are modelled using *application contexts* (for example, weather and music), or not modelled explicitly. Within the semantic sphere, a class definition in a context or ontology can be viewed as a *hook*. By creating an instance of one of these classes, contextual information is effectively hooked onto the context or ontology. Our contexts and ontologies are designed in such a way that semantically equivalent contextual data can be found regardless of their syntax, and coarser levels of abstraction can be inferred from finer ones. Consequently, the scope of an information search is broadened using simple relations. In this paper we also present the Query Service (QS) that has been developed so that high-level application queries can be handled transparently, and results of the appropriate level of abstraction and representation are returned to the application.

Our design approach delivers contexts and ontologies that are well-defined and flexible. Sensor developers can hook contextual data onto, or extend, an existing context or ontology. They can be easily adapted to different applications and environments. Once hooked, the contextual data is put into a distributed store, and applications can access it independently of the sensors. The novelty of this approach is the organisation of the ontologies. This, along with a powerful QS, will be very useful for the building and supporting of a large number of context-aware applications.

Our work is built within a framework called Construct, a fully-distributed and decentralised context aggregation infrastructure for pervasive computing environments [15]. Construct consists of a number of nodes that aggregate contextual data. Each node has its own data-store, and sensors register themselves with a node and inject data into it. Construct nodes gossip [6] amongst themselves to maintain a global model of the system as a whole. All information is represented using RDF as the underlying data model. Applications therefore see a soup of contextual data derived from sensors and can access it through the QS. High-level queries are passed to the QS using RDQL (RDF Data Query Language) [14]. The low level inferring is handled transparently and application-interpretable results are returned. A model of this process can be seen in figure 1.

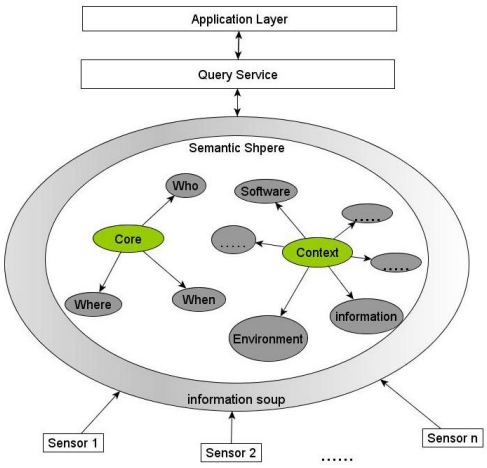

Figure 1. Contexts and Ontologies within Construct

The rest of this paper is organised as follows: Section 2 briefly illustrates the related work in the area along with the semantic web technologies that our approach depends on; in Section 3 we introduce the ontologies that are defined for the pervasive computing domain and describe some specific application contexts; Section 4 describes how the data represented in these ontologies and contexts are converted into information suitable for consumption by pervasive applications. This process is demonstrated with an example location-based application in Section 5. Finally, in Section 6 we conclude the paper and give some directions for future work.

2 Related Work

This paper addresses the issues of context modelling and context accessibility in context-aware pervasive computing systems. The area has attracted attention recently and some seminal approaches that focus on the same issues have emerged: Firstly, the work carried out by Heer et al on the liquid extension to the Context Fabric [10, 11] consists of a query service which supports distributed, continuous query processing for context data. They introduce the notion of an infospace which is a logical storage unit that may be centralised or decentralised. Once context is sensed, it is added to the appropriate infospace. Context is stored in infospaces using tuples consisting of types and values. The value can be a basic value or another infospace allowing queries to be structured as a concatenation of different types. Thus, to resolve a query involves the traversal of a string of tuples. There are drawbacks to this, however. The user is required to know the structure of the infospaces and the types of their tuples in order to make a query. Also, there is no mention of a common semantics for types that tuples may contain making interoperability difficult.

Another related concept is that of the Enactor extension to the Context Toolkit (CTK) [13, 9]. The CTK introduces Widget components which are structures that encapsulate a particular type of context acquiring sensor, for example, a location sensor. Each location sensor will have the same interface, be they an internal RF location system or GPS. This, however, allows only one level of abstraction per interface. The Enactor, which encapsulates some application logic, obviates the application developer from having to subscribe to each widget manually. It consists of a number of References which "support the declarative specification of interest in a set of CTK components through a general query package". References process queries to discoverers and automatically subscribe to any components that match. Like our approach, the low level queries are handled transparently.

Khedr et al [12] introduces context-level agreements into a multiagent pervasive computing environment. They allow user agents to specify context that is relevant to them so that the context management agent can subscribe to the appropriate context providing agents in order to have the appropriate context delivered.

All three systems support a high-level query language that decomposes requests into satisfiable responses and then returns a response to an application's request without the application needing to know the details of how the infrastructure is satisfying the response. However, there is no effort to structure the semantics of the context data to provide a more powerful query service.

Similar to the work from Chen et al on SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) [4], we use the Web Ontology Language (OWL) [3] to model our ontologies. The distinction that we make between the application contexts and the ontologies is closely based on the divide that exists in SOUPA between SOUPA Core and SOUPA Extension. Although the models that they define are quite extensive, we take the approach of organising our ontologies more effectively while keeping them simple. We also use Jena [1] which is a semantic web framework for java.

3 Contexts and Ontologies for Pervasive Computing

Numerous ad hoc ontologies have been created for pervasive computing systems to date. They have been designed with the primary goal of providing a semantics for contextual data so that a common understanding can be given to data from heterogeneous sensors along with entities in the pervasive environment. The goal of this work is to not only develop such a semantics for contextual data, but also to develop our ontologies in a way in which they can be efficiently reasoned about. The hypothesis is that different applications may require the same contextual data, but in different representations or levels of abstraction. By adding a structure to our ontologies, using relations between their contents, this reasoning over data can be done at a lower level and will thus be transparent to the application developer.

The three core ontologies of *where*, *when* and *who* are described in this section along with their general properties and relationships. These ontologies form a base model that is general enough to be used in a range of pervasive computing applications. An overview of the application contexts along with a description of the relations used in the contexts and ontologies to achieve equality and levels of abstraction are also given.

3.1 The Where Ontology

The *where* ontology describes the concept of location in a pervasive computing environment. A location may be defined as a point (*Coordinate*) or as a region (*Space*). Figure 2 shows the hierarchy of location types that are possible. Locations may be either physical, e.g. a set of GPS coordinates; or symbolic, e.g. "RivadelGarda".

Locations may be related to each other in ways that declare equivalence, e.g. `RivadelGarda=GPS (45.88,10.82)` and containment, e.g. `RoomA007 isContainedIn CS-Building`.

Section 5 gives an example of how these mappings are used to transform location data from multiple contexts into a single result at

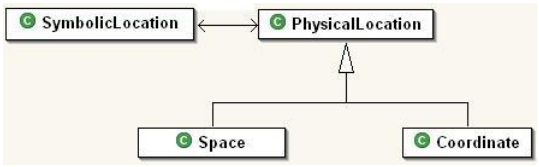

Figure 2. The *where* Ontology

the correct level of abstraction in response to a query from an application.

3.2 The When Ontology

The *when* ontology defines the concept of time in pervasive computing environment. Figure 3 illustrates this hierarchy. Time may be declared as an instant (*InstantTime*) or as a range of time (e.g. *TimeRegion*). Time may be declared as being either symbolic, e.g. `Yesterday`; or physical, e.g. `00:28, Friday 7th April 2006`. Again, there is an equivalence relationship. *TimeRegion* expresses a period of time in a tuple of `<from, to>`. We define three relationships for time: *equals*; *before*; and *after*.

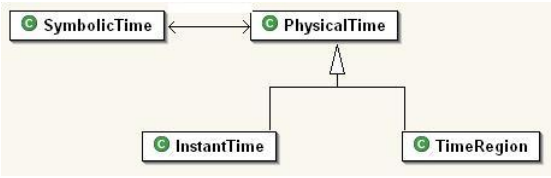

Figure 3. The *when* Ontology

3.3 The Who Ontology

The *who* ontology is different from the *when* and *where* ontologies. It describes an agent that inhabits a pervasive computing environment, e.g. a human user, intelligent agent or sensor. The *who* ontology has only one hook: an *Entity*. Every *Entity* in the system will be attached to this hook and will be uniquely identified. Each *Entity* must contain one or more *Identity* classes which are represented as attributes with values. Any piece of contextual data that identifies an agent declares itself to be representative of this token, e.g. in a tag-based location application the tag ID is mapped to an *Identity* attribute of the corresponding user *Entity*. Instances of the *who* ontology can be mapped to further, less general, contextual information such as a personal profile. Thus, by traversing the equivalence relations between *Identity* classes, any representation can gain access to contextual information regarding the agent in question.

In Section 5 we demonstrate how three representations (*Identities*) of a user from three different sensors are mapped together allowing an application to benefit from access to each of the contexts.

3.4 Application Contexts

Besides these core ontologies, there exist many other types of data that are reusable in a pervasive environment. However, they are too specific to be modelled in an ontology. We have defined application contexts for data from a number of diverse applications that we are working on. These include weather data, flight data and music data. These contexts are stored in a catalogue of data models and are available as hooks for application developers who wish to access the data of that type that are in the data store.

3.5 Transitivity and Equivalence

Contextual data can be modelled using set-theory. Referring to our core ontologies and application contexts, two relations in particular are critical to their structure; transitivity and equivalence. Consequently, there exists the notion of transitive and equivalence relations on elements of sets.

In mathematics, a binary relation R over a set X is *transitive* if it holds for all a, b and c in X, that if a is related to b and b is related to c, then a is related to c. Transitive relations strengthen the reasoning capabilities and are invaluable for certain ontology structures. For example, the *where* ontology is naturally modelled using transitive relations between different levels of abstraction of contextual data. Rooms are contained in floors which are contained in buildings, so that a result for an application query for a high level of abstraction such as "What building..." can be inferred from lower level representations of the same content.

The equivalence relation is a little simpler. An equivalence relation on a set X is a binary relation on X that is reflexive, symmetric, and transitive and it is used to group objects that are similar in some sense. Taking the *where* ontology as an example, symbolic names for locations are equivalent to their corresponding physical representations. Furthermore, in the *who* ontology, the *Identity* instances of an Entity are equivalent representations of the Entity.

To demonstrate the usefulness of these relations alone, a general query for a building name can be derived from $\langle x, y, z \rangle$ coordinates sensed by a tag-based location system by inferring the physical location that contains the coordinate (at a building level of abstraction) and finding the equivalent symbolic name.

4 Query Service

The Query Service (QS) is a layer that sits between the application layer and the data-store. It provides an interface to the application to make high-level queries on the store, and returns the results to the application in the correct level of abstraction. In order for sensor developers to take advantage of the QS functionality, they can make use of the existing ontologies and contexts so that their contextual data can be represented with the appropriate semantics and relations between levels of abstraction. The ontologies and contexts mentioned in the previous section make such a tool possible.

The QS consists of three main components; the Query Handler, the Query Executer and the Query Service Reasoner:

The Query Handler is the query interface that the QS provides to the applications of the system. Any application can use the QS by sending a high-level query to the Query Handler. When an application makes a query, the Query Handler must first determine the *known* and *unknown* facts of the query. The unknown facts are those that the application is requesting and the known facts are those that the unknown facts are requested in relation to. For example, take the query "What room is Bob in now?". In this situation, the unknown fact is the room and the known facts are Bob (the subject) and Tuesday, 11th April, 10:03am (the time that the query is made at). The next step is to find the different representations of the known and

unknown facts, and query for each representation of the unknown's using the known ones as filters on the results. The Query Executer is handed all of the derived queries and the results are returned to the QS Reasoner.

The Query Executer The purpose of the Query Executer is to execute all of the low-level queries that are passed to it from the Query Handler. The Executer queries the data-store and passes the results on to the Query Service Reasoner so that it can then infer further information that the application requires. Virtual sensors may be used to derive properties that are not explicit in the contexts and ontologies. For example, a *hasLocation* property can be derived from a higher level notion of a sighting. A sighting might introduce three predicate triples to the data-store; one stating the person, one stating the time, and another stating the location.

The Query Service Reasoner The basic results returned by the Query Executer may not be of the level of abstraction required by the application. The job of the Reasoner is to reason about the results so that, if possible, they can be moulded into the representation required by the application. Currently, only bottom-up inferencing is supported as top-down inferencing would produce ambiguous or superfluous results (e.g. a building reasoning about what is contained in it could return numerous rooms). Finer levels of abstraction can be generalized to coarser ones using the relations from Section 3.5. From the query, the Reasoner knows the type, level of abstraction and representation that it must match. Using the ontologies and contexts as a reference, this match can be inferred from different representations and finer levels of abstraction. In the above example, the tag-based location system may return a coordinate which, using the *where* ontology as a reference, can infer that the coordinates are in a particular room which, in turn, is in a building as these physical spaces are defined by a set of coordinates.

Currently, a simple custom-inferencer has been implemented to reason over equivalence and transitive relations in order to seek out the required levels of abstraction. Referring to the *where* ontology, one of the transitive relations is *isContainedIn*. Equivalence relations can be defined over multiple types also. For example, a symbolic name of a location might be equal to a physical representation of a location. They are semantically equal but they are syntactically different. Consequently, a query returned for one representation can be converted to another to be of use to the application. Using these relations, the Reasoner references the appropriate ontology to locate the level of abstraction and representation that the application is looking for. If the values returned by the Query Handler are not syntactically correct the Reasoner searches for an equivalence relationship between the syntactic form that is required and the form that is returned by the Query Handler. If one exists, the Reasoner then abstracts the value to the correct level of abstraction using the transitivity relation. Once the level of abstraction is met, the representation can be mapped to the required syntactic form using the equivalence relation.

5 Application

To demonstrate the exchange of context data and ontology data in a pervasive system, we introduce a location-tracking application that queries the data-store for the location of a user. It has a semantic map defining the locations in its realm, and a list of the users of the system. It is capable of making queries for the location of a user at the level of abstraction of a room, floor or building. We use the following sensors to provide location data at different levels of abstraction.

- Ubisense [2] sensors generate coordinate location data for each tagged user with a peak level of abstraction of 30cm in 3D space.
- Bluetooth location sensors which can track location to approximately ten metres. This provides a room-level abstraction to the data-store.
- Activity sensors determine whether an individual is located at a computer by checking whether they are logged in and active at the terminal. This sensor also provides a room-level abstraction.

Each of these sensors insert data into the data-store which have the properties: *hasLocation*, *hasTime* and *hasIdentity*. The Ubisense sensor generates raw data that looks as follows: (`tagID=tag184, time=30/03/2006 13:22:13, x=13.28, y=11.82, z=0.35`). These data are hooked to the core *who*, *when*, and *where* ontologies as follows: `tagID` is hooked onto the *Identity* class of the *who* ontology; *time* is hooked onto the *InstantTime* class of the *when* ontology; and *x=13.28, y=11.82, z=0.35* are collectively hooked onto the *Coordinate* class of the *where* ontology.

When an application asks a question relating to a person's location, e.g. "What room is Bob in now?", the Query Handler takes the known and unknown terms and generates a suitable query in RDQL:

```
SELECT ?location WHERE
?person alsoKnownAs Bob
?time after (currentTime − 5)
?x hasTime ?time
?x hasIdentity ?person
?x hasLocation ?location
```

The Query Executer executes this broad query. At least three results will be found (one for each active sensor). The data that came from the Ubisense sensor might come out in the following format: (`Bob, 30/03/2006 13:22:13, (13.28, 11.82, 0.35)`). These results are passed to the Query Service Reasoner.

The required level of abstraction for the location data response is at the room level. In this example, two different levels of abstraction are returned; data at the room granularity (the data that originated at the activity sensor and Bluetooth sensor); and at the coordinate level (from the Ubisense sensor). The former results are at the correct level, and can be returned unaltered. However, the coordinate data must be raised from the coordinate level to the room level.

Figure 4 illustrates a series of steps that the Query Service Reasoner goes through to process this inconsistent data in order to return the correct level of abstraction to the application. The Query Service Reasoner starts at the level of abstraction of the available data; in this case coordinate data, and follows the transitive *isContainedIn* relation, defined in the *where* ontology, to discover whether it is contained within a defined *space*. The *equals* relation is also used to move between physical and symbolic locations. These relationships are followed until the resulting location maps to a level of abstraction that matches the original query (or it fails if there is no valid mapping — i.e. the coordinate does not match a known room). In this example, the Query Service Reasoner returns the room called "RoomA007". This is done for all available data and the results are returned to the application that made the original query.

6 Conclusions and Future Work

This paper describes a novel design approach to a set of core pervasive computing ontologies describing the concepts of *who*, *where* and *when*. These ontologies are used to ensure interoperability between data from different application contexts. Data is accessed us-

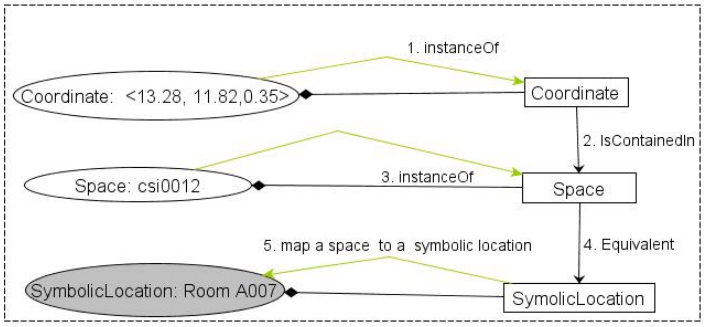

Figure 4. The process by which location data is abstracted to a higher level.

ing a specialised query service that searches for and translates appropriate data to the required level of abstraction for the query. We demonstrate this interoperability with an application that queries for location data. This data has been collected from a variety of sensors at different levels of abstraction. By using the tools in this paper the application developer does not need to be concerned with translating this data.

We describe the core ontologies. However, developers may extend from the core by implementing their own contextual models and adding them to the semantic sphere. When creating new sensors, the developer should use the preexisting contexts and ontologies but this is not required. If they enter data without an explicit model, it is available to application queries but only if they query directly against the data.

When a query is made, multiple sensors may have sensed a context which matches the query constraints. Each of these results will be returned to the application level. It is up to the application developer to process this data. One characteristic of pervasive computing environments is that sensors cannot be relied upon to always give accurate readings. Work is being done to associate a quotient of accuracy with each piece of contextual data provided by a sensor [7]. This will be available to applications and will improve the overall accuracy of an application by allowing sensor data to be fused based on the individual accuracies of the available data. We will annotate data with notions of trust [5] in the same way.

As part of our ongoing development, we intend to explore semantic translation (e.g. *adjacent* in the *where* ontology) with richer relationships in basic structures. Such semantic translation will help to support more reasoning capabilities. We intend to further develop our location-tracking algorithm to query against other types of data. We also intend to develop another application for making generic queries for pieces of data in the data-store that will assist in self-diagnosis, e.g. "tell me everything you know about *x*", where *x* is a single piece of data.

Additionally, as a consequence of Construct's use of gossiping to spread information between its distributed nodes, latency is a concern which must be more fully investigated. Some safeguards have to be put in place so that the occurrence of redundant data is minimised. Due to our use of backward chaining in our virtual sensor for inferencing, every query is independently dealt with by the QS. The disadvantages to this are latency and the computational cost of the same query being inferred multiple times for different applications. We will therefore be investigating the use of forward chaining, where all inferencing is done on all data when it is inserted into the data-store. In this way, the result to every satisfiable query is explicit in the data-store. This has its own problems, but would improve latency, which is paramount in pervasive systems. Truth maintenance is also a factor in pervasive systems, whereby information is inferred from lower level data. If this data is deleted or changes, it is important that this inference is still valid.

REFERENCES

[1] Jena semantic web framework. http://jena.sourceforge.net/.

[2] Ubisense. http://www.ubisense.net/.

[3] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Harrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein, 'Owl web ontology language guide', (2004).

[4] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi, 'SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications', in *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, MA, (August 2004).

[5] Michael Collins, Simon Dobson, and Paddy Nixon, 'Security issues with pervasive computing frameworks', in *Workshop on Privacy, Trust and Identity Issues for Ambient Intelligence at Pervasive 2006*, pp. 1–7, (2006).

[6] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry, 'Epidemic algorithms for replicated database maintenance', *SIGOPS Oper. Syst. Rev.*, **22**(1), 8–32, (January 1988).

[7] Simon Dobson, Lorcan Coyle, and Paddy Nixon, 'Hybridising events and knowledge as a basis for building autonomic systems', *Journal of Trusted and Autonomic Computing*, (2006). To Appear.

[8] Simon Dobson and Paddy Nixon, 'More principled design of pervasive computing systems.', in *EHCI/DS-VIS*, eds., Rémi Bastide, Philippe A. Palanque, and Jörg Roth, volume 3425 of *Lecture Notes in Computer Science*, pp. 292–305. Springer, (2004).

[9] A. K. Dey et al, 'A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications', in *Human Computer Interaction*, pp. 97–166, (2001).

[10] J. Heer, A. Newberger, C. Beckmann, and J. Hong, 'liquid: Context-aware distributed queries', in *Fifth International Conference on Ubiquitous Computing*, pp. 140–148, (2003).

[11] Jason I. Hong and James A. Landay, 'An infrastructure approach to context-aware computing', in *Human-Computer Interaction*, volume 16, (2001).

[12] M. Khedr and A. Karmouch, 'Negotiating context information in context aware systems', *IEEE Intelligent Systems magazine*, **19**(6), 21–29, (November/December 2004).

[13] A. Newberger and A. Dey, 'Designer Support for Context Monitoring and Control', (2003).

[14] Andy Seaborne. RDQL - a query language for RDF. W3C member submission, Hewlett Packard, January 2004.

[15] Graeme Stevenson, Lorcan Coyle, Steve Neely, Simon Dobson, and Paddy Nixon, 'Construct — a decentralised context infrastructure for ubiquitous computing environments', in *IT&T Annual Conference, Cork Institute of Technology, Ireland*, (2005).