An Approach to Dealing with Uncertainty in Context-Aware Pervasive Systems

Adrian K. Clear
Systems Research Group
UCD Dublin, Ireland

Simon Dobson
Systems Research Group
UCD Dublin, Ireland

***Abstract*— Pervasive computing introduces complex systems consisting of many users, sensors, and applications that should react to context data, provide services, and manipulate devices in a predictable and reliable manner. Context data sensed from the environment is largely uncertain due to lack of precision and imperfect or faulty sensors. Uncertainty is generally dealt with at the level of individual context data. Due to the difficulties associated with catering for data of such fine granularity in applications, the environment can be divided into larger chunks of context called *situations*. Situations, comprising of finer-grained events in the form of context data, warrant a different approach to dealing with uncertainty. Furthermore, we demonstrate that the uncertainty threshold of an event that triggers a behaviour is determined by the severity of the behaviour, making this task non-trivial. In this paper we detail an approach to dealing with uncertainty at the level of situations that takes into account the severity of the behaviour that it is triggering.**

I. Introduction

Uncertainty is a major issue in context-aware computing and is largely due to the lack of precision and accuracy of sensors interpreting the environment. It is a problem that cannot be avoided because even as sensors become more accurate and precise, full sensor coverage of an environment is not realistic. As a result, we need to accept that we are dealing with uncertain data and take measures to program with it or to mitigate its effect where possible. Uncertain context can have a major impact on the predictability of the overall system behaviour when low-level events, such as changes in context, are reacted to directly [2]. In particular this is a problem when those events have no notion of a certainty metric associated with them.

To deal with uncertainty in context data, Dobson *et al.* [1] propose to maintain a knowledge-base of context data that can attach a measure of its belief in the accuracy of the data. When applications query for data, the result reflects these beliefs. An inferencing engine returns a single value with an estimation of its correctness using an aggregation function over three pieces of data; precision, decay and confidence. Precision represents the physical accuracy of the data; decay represents how time affects the certainty of data; and confidence is a measure of the system's belief in the data.

This work is partially supported by Science Foundation Ireland under grant number 05/RFP/CMS0062, "Towards a Semantics of Pervasive Computing."

Adrian Clear is supported by an EMBARK Scholarship from the Irish Research Council in Science, Engineering and Technology.

The fine granularity of context data makes it difficult to use at the application level. Applications respond to events that are generally richer than a single sensor reading. They are sensitive to user activities as opposed to single observations. For example, the activity "Walking" is richer, and of more use than, the location coordinate "39 54 32". In the absence of higher-level structures, applications must query for each context that constitutes an event as opposed to a single structure—events become implicit in application logic which prevents them from being reused. The environment can be divided into larger chunks called situations [9] to cater for these events. Situations have two major advantages in the field of context-aware computing. Firstly, situations are high-level abstractions of collections of facts (context) about the system's environment. Their introduction relieves the burden of dealing with aggregation of context in application logic and provides a step towards the possibility of realistic user-programming of context-aware systems. Furthermore, situations can be used as stabilisers for the behaviour of the system whereby a small change in context may be irrelevant at the situation level so the behaviour of the system can remain the same [2]. With the introduction of situations to context-aware computing, we must review the measures in place for dealing with uncertainty and adjust them for use in these higher-level structures.

Loke *et al.* [5] recognise that context-aware behaviours are not uniform in the certainty levels that they require of events that trigger them. This non-uniformity correlates to the severity of the behaviours. We use the term severity to illustrate the measure of a behaviours ability to be harmful when exhibited in the wrong circumstance. It is trivial to take severity into account when dealing with individual contexts as events. Situations are composed of many events or individual contexts, however. In order to cater for this characteristic at the level of situations, we hypothesise that when higher-level situation abstractions are used as events, not only do behaviours have differing thresholds for their certainty, but the individual contexts do not always have uniform importance when arriving at the certainty measure.

As an example, consider the following applications in a hallway: a lighting application and a door authorisation application. The authorisation application will want to be certain that the user it is authorising is actually located outside the door. The certainty of location is very important as we do not want to allow access to others in the hallway. Once an intruder has entered, we cannot undo the behaviour. Alternatively, certainty in location is not as crucial to the lighting application. If people on room boundaries activate the light we can undo it by turning the light off and learning about the precision required. In this case it would perhaps be more desirable to turn the light on than to leave the hallway in darkness. Consequently, it becomes non-trivial to deal with uncertainty while moving from a programming model that uses low-level events to one that uses higher-level abstractions as events.

We wish to expand on the work by Dobson *et al.* by representing the uncertainty of context with the three values: precision, decay and confidence, and using this data at the level of situations to attain a measure of uncertainty. In order to satisfy our hypothesis, we wish to include a measure of influence that each context has in a situation structure, and make this highly customisable for applications. In this way, application developers get more flexibility in designing the triggers for adaptation. The system becomes more flexible when the effects of a behaviour are mild, thus providing a more useful service to the user.

We propose a more natural, fuzzy approach to dealing with uncertainty that builds on previous work. We are attempting to minimise the possibility of erratic behaviour in these complex systems, in an attempt to make them predictable and useful in their behavioural transformations, while minimising the constraints that we put on them. We are moving away from treating all context data with equal importance when triggering behaviours so that we enforce more natural, flexible constraints, while remaining cautious when it is necessary to do so.

Section II describes some background for the paper along with the related work in the field. Section III details the approach that we are taking to deal with uncertainty at the level of situations. We discuss the benefits and limitations of our work, along with some comparisons to existing research, in Section IV. Finally, in Section V, we give some conclusions and describe some possible future directions.

II. Background and Related Work

The Construct framework [8] is a context acquisition and distribution framework that we use to deploy pervasive computing applications. It consists of a number of distributed nodes that accumulate context data from local sensors. Each node has its own data store where context is represented as RDF triples. The contents of the local data stores are distributed throughout the network. Applications access context data by providing queries, written in SPARQL [1], to a query service component. We leverage the Construct framework to bring about our approach to dealing with uncertainty in situations.

Ontologies are used in the field of Pervasive Computing as one approach to model context data so that it is computer interpretable and can be communicated among heterogeneous devices. They form a semantics for sensor data. The GAIA project [7] is one example of the use of ontologies from the Semantic Web domain to model context. The Web Ontology Language (OWL) [6] is a language for defining and instantiating data models. An OWL ontology includes descriptions of classes and their related properties and instances. OWL ontologies can be used to process and reason about the content of information. They facilitate machine interpretability of data by providing a formal semantics for it.

Uncertainty is also an acknowledged problem in the field of pervasive computing. Hightower *et al.* [4] devised the Location Stack, which is a layered software engineering model for location in ubiquitous computing. One of the principles that their model is based on is that uncertainty must be preserved—uncertainty measurements at the sensor level should be preserved in order to provide correct uncertainty levels at higher abstraction levels, which may be more closely tied to adaptations. A given example is that an application routing telephone calls to a handset near the intended recipient may take a message if uncertainty about the user's location is too high. In their Location Stack, which is comprised of seven layers, each suggesting a division of functionality, there exists a fusion layer where location measurements are merged in an effort to reduce uncertainty. Differing capabilities, redundancies, and contradictions are exploited to achieve this. The paper also suggests a contextual fusion layer in which location data is merged with other types of context to produce situations. From their design principles, this layer assumedly contains measures to preserve uncertainty of sensor readings although no suggestions are made. This layer is equivalent to the space that we focus on in this paper.

Dobson *et al.* [1] suggest a means to dealing with uncertainty based on the association of meta-data with context data. Three values are used to determine its accuracy: precision, decay and confidence. Precision represents the physical accuracy of the data; decay represents how time affects the certainty of data, and confidence is a measure of the system's belief in the data. Context, along with this meta-data, is entered into a knowledge-base. For each reading, the uncertainty is calculated by multiplying the three values, which are between zero and one. When an application makes a query, if multiple results are returned, each one must be voted for. Every sensor reading returned gets a vote, the value of which is the product above. If multiple readings are returned from the same sensor, the votes are divided accordingly. The result returned to the application is given a percentage of certainty calculated from the total vote.

Loke *et al.* [5] recognise that taking actions only when there is a high degree of certainty in context may be unnecessarily cautious. Actions may be reversible and/or there may be reasonable mitigation strategies for them. In these cases the potential benefits of the actions may outweigh any potential costs if the context is incorrect. The paper models a system with rules that take into account action-specific thresholds for uncertainty and severity in making a decision about exhibiting an action. They introduce argumentation structures to represent context as justifications and explanations for recognised situ-

[1]http://www.w3.org/TR/rdf-sparql-query

ations and for taking actions. Using these structures they can gain a quantitative measure of certainty of a perceived context. They use arguments to support or oppose beliefs in context and by summing the supporting and opposing arguments, they gain a measure of certainty.

Henricksen *et al.* [3] developed their own context modelling approach based on Object Role Modeling. In order to deal with imperfect information, they added extensions to this context modelling approach based on four different classes of context information: sensed, static, profiled and derived. Each of these may be prone to different types of imperfection. Context can be unknown, when no information about it is available; ambiguous, when several different reports are available; imprecise, when a reported state is an inexact approximation of the true state; or erroneous, when there is a mismatch between actual and reported states. By incorporating different classes into their context model, they can reason about issues like uncertainty or conflict detection in a more fine-tuned manner.

III. Uncertainty in Situations

In preparation for dealing with uncertainty at the level of situations, we first describe our approach to modelling the system's environment and contextual facts. For the purpose of modelling, we make a clear distinction between a model of the environment and a model of context. The environment is a data model used to represent concepts that are relevant to the system, and the relationships between those concepts. A context is a fact, made machine interpretable, through the use of this data model. We view the environment as any concepts that may need to be processed by applications in order to make the system context-aware. Thus the environment consists of concepts such as location, time, and person. We use ontologies then, written in the Web Ontology Language, in order to express the meaning and semantics of these concepts so that they may be processed and reasoned about.

A. Modelling Context

We view a context as a fact that is constructed using the concepts in our model of the environment, data types such as strings and integers, and relationships between concepts and between concepts and data type values. These relationships are also modelled in the above-mentioned ontologies as properties. A context is structured as a graph represented in the form of a triple, $\langle subject, predicate, object\rangle$, where the predicate is a relationship between the subject and object—a directed, labeled edge in the graph from the subject to the object.

We use the Resource Description Framework (RDF) [2] in order to model contextual facts as triples. In RDF, these triples are known as statements. For the purpose of modelling context, statements are an insufficient model—often extra information is required to annotate context (e.g., the time that they were added or the sensor that added them). In our case, to deal with uncertainty, we need to annotate statements with three values: precision, decay and confidence.

[2]http://www.w3.org/RDF

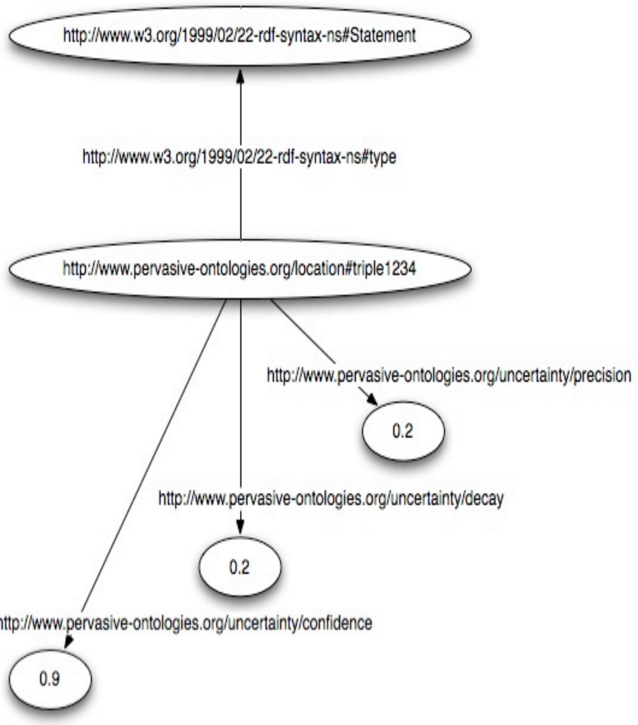
http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.pervasive-ontologies.org/location#triple1234
http://www.pervasive-ontologies.org/uncertainty/precision
0.2
http://www.pervasive-ontologies.org/uncertainty/decay
0.2
http://www.pervasive-ontologies.org/uncertainty/confidence
0.9

Fig. 1. Context Model: Reification of contextual facts

We use what the World Wide Web Consortium (W3C) refers to as *reification* [3] in order to annotate context. RDF provides an in-built vocabulary for describing RDF statements. A description of a statement using this vocabulary is called a reification of the statement. Reification allows us to treat an RDF statement as a resource and describe its subject, predicate and object using statements about this resource. It is also possible to introduce further statements about the resource to annotate the statements. For example, the triple $\langle Adrian\ hasLocation\ Room00\ .\rangle$ could be described as $triple1$ and we could then state facts about $triple1$ using further statements such as $\langle triple1\ wasEnteredAtTime\ 13:32\ .\rangle$. We model context in RDF as a reification of a contextual fact, with the addition of precision, decay and confidence attributes. We can also associate time, sensor, etc. in this way. In Figure 1, we associate a URI with a context (triple1234 in the diagram), and as a result treat it as a resource. We can then associate uncertainty values (precision, decay and confidence) with it using statements about the triple.

Facts about the environment are interpreted from raw data through sensors. Sensors construct RDF triples from this data and use the ontological description of context to associate the appropriate meta-data with them. The resulting triples are then entered into a shared, distributed data store which is part of the Construct framework. All of this information is then available to applications through a query service interface.

[3]For more information regarding the RDF reification vocabulary, see http://www.w3.org/TR/rdf-mt

B. Modelling situations

A situation is a set of constraints on contextual facts so that when these constraints are satisfied in the sensed environment, the situation is said to occur. We model constraints on context as separate ontology classes. We model two types of constraints using ontologies: intervals and ranges. Intervals contain a *max* and *min* value so that we can query for anything in between them. Ranges are a list of possible values for contexts that cannot be represented numerically, such as location. We recognise that these constraints alone are not rich enough to design a context-aware system—they, along with the use of ontologies to model them, are simply a proof of concept. In future we will need more advanced relationships for constraints such as "beside" or "close to".

The severity of behaviours or actions in a context-aware system is an important factor to take into account when dealing with uncertainty. Behaviours can be thought of as a spectrum with critical at one end and safe at the other. As a result, whether a behaviour should be exhibited or not depends on both the position of the behaviour in the spectrum, and the confidence that the system has in the event that triggered the behaviour.

Because each contextual fact is an event, there is a need for something different when dealing with uncertainty in situations. Our hypothesis is that context information, of which a situation is comprised, may be of different importance [4] to the situation, dependent on the behaviour. Our confidence in the location of a person may be more important in a security conscious application than one with few potential bad effects, for example. We need to incorporate this into our model of situations.

In our model of situations, each context constraint has associated with it a *relevance* value to signify the certainty requirement of that particular type of context to a situation. A relevance value is a figure between 0 and 1, where 0 indicates that the context must have a high certainty, and 1 indicates that its certainty is unimportant. A context constraint also has a subject and predicate used for querying purposes. The constraints represent values that should be returned when querying with a given subject and object. In Figure 2, we can see that an interval constraint has a min and max value along with the subject and predicate used for querying. It also has a relevance value to express how certain the system must be in the context for a particular situation to occur.

A situation is composed of one or more context constraints, and these are used for reasoning about whether the situation is occurring or not. Each context-aware application, deployed using the Construct framework, has an associated properties file, which points to the situation instances that are relevant to it. We have also designed a generic *situation query service* that

[4]We require a single measure of certainty for a situation. In order to arrive at this value we must take the certainty of each individual context into account. By importance of context to a situation, we mean that the nature of a behaviour may enforce different requirements on the certainty level of contexts, thus giving them different influences in arriving at a certainty measure for the situation.

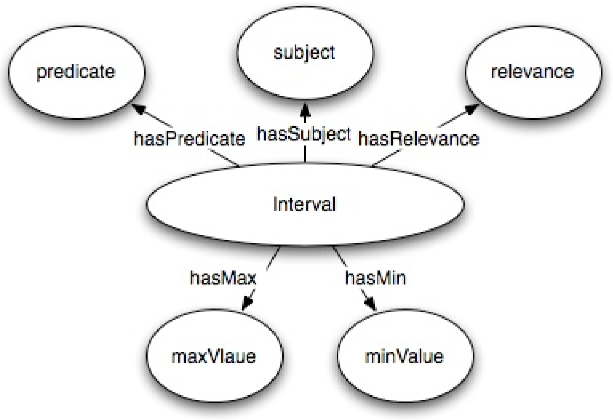

Fig. 2. Context constraint model of an interval

takes a set situation instances and polls the local query service to inspect whether the situation has been realised or not. In order to reduce complexity of the middleware's query service, each application has its own situation query service. Once a situation has been realised, it is noted in a *situation cache*, along with its certainty measure. The application monitors the cache and adapts its behaviour accordingly. The architecture of our application can be seen in Figure 3.

In order to arrive at a measure for the situation's certainty, we have the following process: for each context constraint in the situation, the contexts that support it are returned from the local query service. In order to come up with a percentage for the certainty of this context constraint, each context gets a vote as in Dobson *et al.* [1]. Once we have a percentage certainty for each context we apply a weighting scheme based on the relevance values of the constraints. We multiply the percentage by the relevance value and add the result to the original percentage. For example, if we were 50% sure that a context was correct, and it has an influence of 0.5 (semi-important), we can boost its value up to 75%. We then take the lowest value of all the percentages and assign this as our certainty in the situation. An example can be seen in Figure 4 where even though the relevance value for location boosts its value slightly, it still does not meet the strict threshold required by the application.

Situations in this approach are closely tied to applications, and it is the task of the application developer to set constraints on context information that situations are comprised of. The relevance values act as a weight on the certainty of a context which may, as a result, increase or unalter its actual value, thus changing the influence it has on the certainty of the situation. We allow certainty of context to be fuzzy where appropriate to increase the flexibility of the system. A threshold is defined for the application, which the overall situation certainty must equal or exceed, for a behaviour to be triggered.

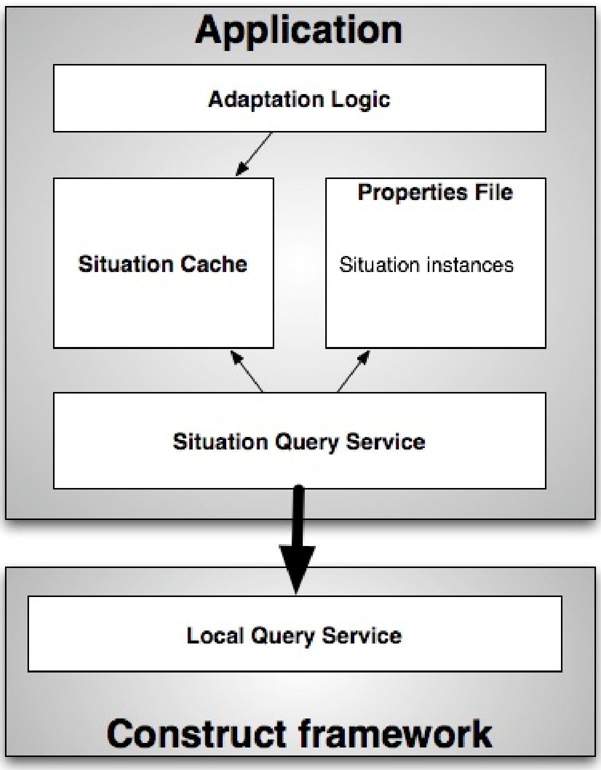

Fig. 3. System Architecture

IV. Discussion

In this section, we will discuss the limitations and benefits of our approach, and compare it to existing work where possible.

A major concern for context modelling is reuse. Context models are designed so that common concepts can be reused by many applications, allowing them to communicate in the same language. Models are decoupled from applications. With the advent of situations into the field, an obvious approach is to model them in the same manner. Situations then can be shared among applications in the system and they can all react to the same events. Although it is possible to set a threshold on their uncertainty, we argue that treating all contexts with the same importance in coming up with this value leads to an inflexible system.

Moreover, as Hightower *et al.* [4] suggest, uncertainty should be preserved and not abstracted so that measurements at higher levels should be as correct as at the sensor level. Using our approach, situations such as "meeting" can be modelled in a generic manner. However, instances of these, such as "in-meeting", are more closely tied to applications. The reason for this is that the influence of individual contexts on a situation is not uniform. Therefore, we cannot model instances in a system-wide manner—they are application-specific. "in-meeting" may have different requirements in triggering a severe behaviour as opposed to a less severe one. Due to these differences in severity, a single weighting scheme for the context constraints of a situation is insufficient.

This places a larger burden on the developer, than with the use of decoupled, reusable situations, although the benefits in terms of flexibility are much greater. The burden is not as great as that of reacting to context directly in applications, however. Loke *et al.* [5], in arriving at a certainty measure for a "belief", treat each fact supporting or opposing the belief as equal when arriving at a single measure. We take a step further and acknowledge that not only do applications require different uncertainty thresholds for chunks of context, but individual contexts are important from a flexibility standpoint.

Our approach still works on the basis of thresholds. The lowest value of uncertainty of an individual context represents the certainty of the situation. The difference, however, is that we associate relevancies with these contexts in order to weight the effect that they have in determining this measure. This has two notable effects: if any single context is of critical importance, and has low certainty, our model will have little effect; if a single context is of critical importance, and has high certainty, while a less important context has a low certainty, the latter will get boosted up making the action more likely to occur. This has the effect of making the system safely flexible where possible but robust and reliable when necessary. Our model provides the flexibility to implement applications such as switching the lights on, for example, when the importance of the service outweighs that of the certainty of a persons presence. Other systems do not provide the flexibility required to make this possible. An application can be made more uncertainty-conscious by simply raising the influence of an important context.

Programming with fine-grained data such as individual contexts makes application logic very cumbersome and difficult to maintain. Using our approach, an application developer can instantiate a generic situation and customise it with *relevance* values. Situations are more natural abstractions to use when programming context-aware systems, and are kept separate from the application logic. As a result, any changes that the application designer wishes to make to tune relevance values will not affect the program code. A limitation of this approach is that more processing is required in the middleware. With the customisation of situations to applications, we increase the amount of situations that must be reasoned about.

Our approach to handling uncertainty in this way also paves the way for an aspect of user-programming of these systems. Situations provide a more natural means for users to understand the merge between the physical and virtual world. The model that we use would allow them to easily tweak these relevancies based on their own experience of the system.

Another issue with our approach is that of composition of situations. In theory it is possible for a situation to be composed of other situations. As the hierarchy increases, the complexity and burden placed on the developer to associate relevance values with each context constraint could potentially become unreasonable. Perhaps there are approaches that could be taken to deal with this such as the application of our model only at the highest abstraction. This is something that we must look into in the future.

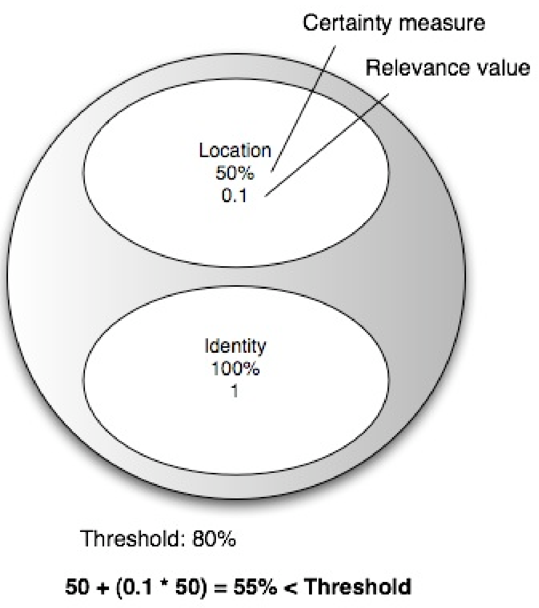

Fig. 4. An example situation with certainty measures and relevance values

V. Conclusions and Future Work

Uncertainty is a major issue in pervasive computing due to the inaccuracies and imprecisions inherent in raw sensor data. Although research has looked at dealing with uncertainties at the level of context, there is a need for a different approach when dealing with uncertainty at the level of situations. Research has shown that uncertainty measures should be preserved from the sensor level up to higher abstractions. Furthermore, the differences in severity of context-aware behaviours places non-uniform requirements on the level of certainty required by events that trigger them. We have shown that as higher-level abstractions of context are composed of finer-grained events, there is a need at the application level to correlate the uncertainty requirement of events encapsulated in these abstractions with the severity of behaviours that they trigger.

In this paper we have illustrated a first-cut approach to dealing with uncertainty at the level of situations. By introducing context constraints and application-specific instances of situations, we can arrive at a measure of uncertainty for situations that takes into account a weighting scheme on individual contexts, provided by the application. Context-aware adaptations become more flexible in their requirements of certainty, while remaining safe when the consequences of an action are severe.

While our approach has substantial benefits, it also makes apparent some drawbacks and issues, which we wish to look further into in the future. Also, as it is a first-cut approach, we wish revisit some aspects of its design and implementation in the future. In particular, we wish to further analyse and evaluate the relevance metric that we use. Moreover, we wish to do an evaluation based on a comparison between situations from the physical world and perceived situations and certainty values in the virtual world. We also wish to evaluate the correlation between severity levels of context-aware behaviours, and the realisation of situations that trigger those behaviours.

References

[1] S. Dobson, L. Coyle, and P. Nixon, "Hybridising events and knowledge as a basis for building autonomic systems," *IEEE TCAAS Letters*, 2007, to appear.

[2] S. Dobson and P. Nixon, "Whole-system programming of adaptive ambient intelligence," in *Proceedings of HCI International 2007*, ser. LNCS. Springer-Verlag, 2007.

[3] K. Henricksen and J. Indulska, "Modelling and using imperfect context information," in *1st Workshop on Context Modelling and Reasoning (CoMoRea)*, ser. PerCom'04 Workshop Proceedings. IEEE Computer Society, March 2004, pp. 33–37.

[4] J. Hightower, B. Brumitt, and G. Borriello, "The location stack: a layered model for location in ubiquitous computing," in *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, 2002, pp. 22–28.

[5] S. W. Loke, "Facing uncertainty and consequence in context-aware systems: towards an argumentation approach." [Online]. Available: citeseer.ist.psu.edu/loke04facing.html

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

[7] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces," *IEEE Pervasive Computing*, pp. 74–83, Oct–Dec 2002.

[8] 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, Ireland*, 2005.

[9] J. Ye, A. K. Clear, and S. Dobson, "Towards a formal semantics for pervasive adaptive systems," *Computer Journal*, 2007. To appear.