A design model for context-aware services based on primitive contexts

Andreas Pappas
Electronic & Electrical Engineering Department
University College London
and BT Exact
+44 (0) 1473 649511
apappas@ee.ucl.ac.uk

Stephen Hailes
Computer Science Department
University College London
+44 (0) 20 76793432
s.hailes@cs.ucl.ac.uk

Raffaele Giaffreda
BT Exact
+44 (0) 1473 644171
raffaele.giaffreda@bt.com

ABSTRACT
This paper describes a conceptual model for reasoning about context. The concepts introduced here may be applied in designing context-aware services and modelling context-sensitive interactions between context-aware systems. We introduce the concept of "primitive context" as the basic context abstraction. The concept of "primitive context" effectively captures the notion of "context" and provides a basis for formalising and reasoning about context in a consistent and conceptually simple way. A primitive context reflects an adaptation capability of a system. Each primitive context is associated with an ontology that describes the capabilities, relations and information valid for that particular context.

Keywords
Context, Context-awareness, Context modelling

INTRODUCTION
Today's networks are characterised by their highly dynamic nature most evidently exhibited through device mobility. Apart from mobility, many network parameters and resources such as QoS and bandwidth as well as other information sources vary through space and time. In order to cope with such variability, services provided on these networks should be able to adapt to changes in their operating environment. Context-aware computing aims to deliver this functionality. Context-aware services are services that exploit knowledge from diverse sources and adapt their operation to this knowledge.

In this paper we describe a novel design model for context-aware applications and services. The same concepts may be applied in modelling context-sensitive interactions between context-aware systems. We introduce the concept of "primitive context" as the basic context abstraction. The concept of "primitive context" effectively captures the notion of "context" and provides a basis for formalising and reasoning about context in a consistent and conceptually simple way. Each primitive context is associated with an ontology that describes the capabilities, relations and information valid for that particular context.

Our aim is to reduce the complexity and inconsistency associated with the design of context-aware services by introducing a useful and straightforward way of reasoning about context. The "primitive context" abstraction seems to provide such a facility. The model, being conceptual, is independent of any specific development environment and we believe that it will be able to form the basis for a well-structured development process that will effectively and efficiently capture the requirements and deliver the required functionality of a system. Additionally, this concept facilitates the modelling of context-sensitive interactions between different systems as it exposes, at any time, only the aspects of the system that are relevant to the interaction and hides any unnecessary complexity.

In order to achieve this, we will build on existing technologies that can provide the functionality desired in a context-aware system. These include ontology-based knowledge representation, service-oriented architectures, existing context-aware solutions and knowledge acquisition technologies.

The rest of this paper is organised as follows: Section two provides an overview of context-aware design and methodologies, section three presents a scenario that is used to illustrate our concepts while section four introduces the concept of "primitive context" and further definitions and conventions used in our model and describes the design and operation of systems based on our model. Section five concludes the paper and states future plans.

BACKGROUND
A major obstacle to widespread deployment of sophisticated context-aware services has been the lack of consistency in their design process. This has lead to the development of services that only work within their development environment and cannot interact with each other. The problem arises partly due to the complexity and the great diversity of context-sensitive services and the lack of consistency and standardisation in their design and operation. Furthermore issues relating to knowledge

acquisition, categorisation, processing, interpretation, aggregation, storage and dissemination (what we may collectively refer to as "knowledge management") are yet not very well understood. Although significant progress has been achieved in all of these fields, there still exists no consistent formal or informal approach for designing context-aware services and context-sensitive interactions are still treated in a proprietary way that suits the needs of the service/application developers.

The Context Toolkit [1] provides a useful environment for the development of context-aware applications by using widgets that encapsulate sensors and may be organised in a hierarchical architecture thereby inherently supporting basic features of context-aware applications such as information aggregation. It also hides the operation of sensors by providing an interface to which an application can subscribe. However, it does not go as far as to provide a formal treatment of context as we aim to achieve with our solution. It is nevertheless suitable for building a service-oriented architecture through its subscription model and will most likely provide the underlying platform on which our model will be implemented.

Egospaces [2] builds on a context abstraction defined as a "view". The "view" of an agent includes any environmental or operational data that may be of interest to the agent at any given time. An agent may have different views defined and may switch between views according to its operational requirements. A view is essentially a restriction placed upon the entire environment in which the agent resides. It is therefore similar to our approach in the sense that it aims to provide context abstractions that facilitate system design by imposing restrictions upon the environment.

Ontologies have increasingly appeared in context management due to their powerful descriptive capabilities and their re-usability [4] that make them ideal candidates for describing and communicating system properties and relations, i.e. knowledge. Several ontology description languages exist, the most advanced of them being the Web Ontology Language (OWL) [3] developed by the W3C. We aim to use ontologies as specifications for our primitive contexts.

A SCENARIO

In order to clearly illustrate the ideas presented here, we make use of a typical example scenario of the ubiquitous computing vision that could benefit from these ideas.

A visitor is being driven around a city centre while using his PDA to execute money transactions through his bank's online e-banking application. At the same time he is listening to a streaming newscast through his PDA. Network connectivity for his PDA is provided by a dense mesh of overlay networks including bluetooth, WLAN and 3G.

In terms of ubiquitous computing, there are three main considerations in this scenario: providing connectivity for the PDA, providing security for the e-banking application and providing enough bandwidth for the streaming audio application. If mobility, security or bandwidth parameters change, then action should be taken in order to adapt to the new environment. These actions should be automatically triggered and the new environment should automatically become known to parties interacting within this environment.

THE MODEL

In this section we first introduce some key concepts of our model along with their definitions and examples of their application to the scenario presented above.

A **primitive context** is a context specification that describes a discrete adaptation capability of a system within the operating environment. For example, in the scenario presented in section three, a primitive context is that of being a `"mobile_user"`, `"fixed_user"` or `"secure_user"`. Each primitive context is described by an ontology instance. The ontology describes the capabilities, relations and other information that are valid within that primitive context.

A primitive context is said to be an **active primitive context** when it describes the current situation that a system is in, i.e. it is a primitive context that is valid at some instance. A primitive context is activated when some condition becomes true. In our scenario, when the user is roaming while using the e-banking application and listening to streaming audio, the active contexts are: `"mobile_user"`, `"high_security"` and `"medium_bandwidth"` that reflect the basic capabilities that are required and that the operating environment should support.

The **current context** of an entity refers to some function of all active primitive contexts. The current context describes the state of a system at any time. However, conflicts may arise between two or more primitive contexts when they become active. For example the `"mobile_user"` and `"high_bandwidth"` primitive contexts may not gracefully coexist in certain conditions. Therefore, the function that generates the current context should be designed in a way that eliminates conflicts and complies with precedence rules that may be set either by the application developer or the user in the form of preferences.

Design

In order to build a context-aware system based on primitive contexts we should firstly decide upon the discrete adaptation capabilities or "degrees of freedom" of the system, i.e. whether we wish the system to adapt to changes in mobility, security, location, temperature etc. For each adaptation capability we decide on the granularity

of the adaptation depending on the degree of flexibility we desire. The primitive contexts reflect these discrete adaptation capabilities.

In our scenario we may define three independent lines of adaptation: mobility, security and bandwidth. In terms of mobility, the granularity of the adaptation we require may be "highly_mobile", "mobile" and "fixed" depending on the capabilities of the PDA. The e-banking application requires a very secure environment and therefore we could only define two primitive contexts for security: "high_security" and "low_security", while the streaming application may be able to adjust to various bandwidths through proprietary adaptation algorithms and therefore a number of primitive contexts may be associated with bandwidth, e.g. very-high, high, medium, low. The three distinct adaptation lines along with their primitive contexts are shown in Table 1.

Mobility	**Security**	**Bandwidth**
High Mobility	Strong Encryption	High
Mobile	No Encryption	Medium
Fixed		Low
		Offline

Table 1 Primitive contexts for example scenario

It is apparent that the more flexible the system is, the more primitive contexts are required in order to provide this flexibility but with a cost on complexity. The purpose of a primitive context is, however, to encapsulate a mode or state of the system that would otherwise be represented by a large number of information. It may be easier to think of it as an object-oriented approach to context-aware design and programming, where states or situations are represented, accessed and acted upon as if they were objects.

Each primitive context is associated with some actions, services, preferences and devices that must or can be used within that context. These properties are fully described in an ontology associated with each primitive context. In our scenario, for example, the primitive context "high_security" may be associated with strong authentication services and may only be used in networks that provide high security such as 3G as specified in the ontology that describes the "high_security" primitive context. Once this primitive context is activated, the behaviour of the system is dictated by the ontology instance that describes it. Therefore it will be restricted to the capabilities that emerge from the ontology, such as connectivity to only secure, 3G networks.

Primitive contexts that refer to a single adaptation capability of the system (e.g. mobility) may be associated with the same ontology specification. I.e. "highly mobile", "mobile" and "fixed" refer to the ability of the system to adapt to different degrees of mobility and therefore can be described by a single ontology specification while the ontology instances may vary in each case.

Context Activation

Each "primitive context" is associated with a number of context-information items that are monitored through sensors. A primitive context is activated when a set of conditions on the monitored parameters are met. The term "sensors" here refers to both hardware sensors that capture environmental data as well as logical sensors that capture digital information from logical sources such as databases. A primitive context is triggered (activated) by events captured through one or more sensors. In the scenario we have been using, the primitive context "high_security" will be triggered once the e-banking application is started. This behaviour is automated by pre-associating the e-banking application with "high_security" primitive context. Once this primitive context is activated, the PDA will try to connect to a secure network and the user may be required to enter additional authentication information. We emphasise that all these actions are specified in the ontology, therefore the ontology should be quite detailed. Additionally, since an ontology imposes a set of restrictions on the operation of the system many system capabilities may have to be restricted. For example, if the PDA was connected to a WLAN before the e-banking application was started and the "high_bandwidth" primitive context was active, the new context dictates that only 3G networks are accepted therefore the WLAN connection may be dropped and the "high_bandwidth" primitive context should be de-activated. This may then cause a significant degradation on the streaming session's quality. It is in events as this that the conflict resolution mechanism mentioned above should take over in order to resolve conflicts, precedence and preferences.

Interaction

When two systems interact, their interaction is determined by the current context of each system. And since the current context of a system is described by a dynamic or transient ontology, the interaction between two different systems is also determined through these ontologies. As in context activation, interaction between systems should also be subject to conflict resolution and/or a negotiation phase in which details of the interaction (such as protocols, services, resource allocation) are determined. To illustrate this, consider the case in which the PDA in our scenario has a variety of different networks on which to connect. These networks are also associated with ontologies reflecting their specification, services, protocols, state, rules etc. Therefore the PDA will select which network to

connect on depending on its current context (e.g. secure) and the availability of options. To do this it will query all available networks on their security provisions (described in an ontology) and if one can match the set requirements it will be selected or a new round of querying will start if there are many candidates. The point here is to match as many requirements as possible for both systems. At the same time, however, the network may deny connectivity to any node that does not comply with its specification. E.g. PDAs may not be allowed to connect to a private WLAN because they may be considered insecure.

Conclusion and Future Work

We believe that the concept of "primitive context", as presented in this paper, provides a suitable and effective context abstraction for reasoning about context-sensitive systems. Based on this concept we presented a design methodology that is based on standardised tools (e.g. ontologies) and is therefore independent of any specific development environment. These properties of our model have the potential to significantly reduce the complexity evident in the design of context-aware services and applications and to improve the interoperability of independently developed context-aware services.

We aim to implement a prototype of this model based the Context Toolkit [1] as the underlying context acquisition platform and define ontologies using OWL [3]. Additionally we aim to deliver a theoretical analysis of the model in order to determine its benefits as a context-modelling approach.

REFERENCES

1. A. Dey and G. Abowd, The Context Toolkit: Aiding the Development of Context-Aware Applications, *Proceedings of the Workshop on Software Engineering for Wearable and Pervasive Computing*, Ireland, 2000
2. C. Julien and G.C. Roman, Egocentric context-aware programming in ad hoc mobile environments, *Proceedings of the Tenth {ACM} {SIGSOFT} Symposium on the Foundations of Software Engineering*, New York, 2002
3. S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider and L.A. Stein, OWL Web Ontology Language Reference, W3C Recommendation, http://www.w3.org/TR/owl-ref/, 2004
4. X.H. Wang, D.Q. Zhang, T. Gu and H.K. Pung, Ontology Based Context Modeling and Reasoning using OWL, PerCom Workshops 2004, 18:22.