CASS - Middleware for Mobile Context-Aware Applications

Patrick Fahy Siobhan Clarke

Distributed Systems Group,
Computer Science Department,
Trinity College Dublin,
Ireland
{Patrick.Fahy, Siobhan.Clarke}@cs.tcd.ie

Abstract
Among the difficulties faced by designers of mobile context-aware applications is the increased burden of having to deal with context and also the processing and memory constraints of small mobile computers. Although progress has been made in the area of frameworks and toolkits for context-awareness, there is still a need for middleware that supports higher-level context abstractions and is both flexible and extensible in its treatment of context.

CASS (Context-awareness sub-structure) is server-based middleware intended to support context-aware applications on hand-held and other small mobile computers. CASS enables developers to overcome the memory and processor constraints of small mobile computer platforms while supporting a large number of low-level sensor and other context inputs. A key feature of CASS is its support for high-level context data abstraction and the separation of context based inferences and behaviours from application code. This separation opens the way to making context-aware applications configurable by users.

1 Introduction
One difficulty for context-aware application designers has been the lack of generic infrastructure for developing context-aware applications (Salber & Abowd, 1998). In addition, existing applications have used a limited range of context types. They have also been limited in context-aware features. Dey & Abowd found that the most often used context parameter was location while the most common application feature was the presentation of context-aware information to users (Dey & Abowd 2000).

Schmidt et al. also find that location is the most often used context parameter and assert that mobile applications could benefit from a broader notion of context. This work investigates the use of multiple sensors to gather a wider range of context information (Schmidt et al. 1999).

It is likely that future context-aware applications will capture a richer variety of context information as well as exhibiting a richer variety of context-aware features than is currently the case. Providing support for application designers in the integration of a wider range of context into their applications will encourage the development of useful and compelling applications for hand-held computers. However, Chen & Kotz assert that the collection and management of context information from large numbers of diverse sensors is a fundamental challenge in ubiquitous computing (Chen & Kotz 2002).

In this paper we describe how mobile applications can benefit from the ability to use high-level context data. We examine some of the difficulties in deriving high-level context data and how application designers might be supported in this task. We then describe CASS, an extensible middleware for context-aware mobile applications. CASS enables developers to overcome the memory and processor constraints of small mobile computer platforms while supporting a large number of low-level sensor and other context inputs. A key feature of CASS is its support for high-level context data abstraction and the separation of context based inferences and behaviours from application code.

2 High-level Context Information
Using a wide range of context inputs from sensors or other sources increases complexity for application designers. However, it is the higher-level context abstractions that can be derived from these inputs that is of most interest. Dey et al. assert that using higher-level context abstractions that sit on top of individual component abstractions should make it easier for application designers to use context, they reason that the higher the level of the context, the easier it should be to use, at the expense of hiding some information (Dey et al. 2001).

Gellersen et al. describe the use of multiple, comparatively simple, diverse sensors to collect context information from which higher-level inferences can be made (Gellersen et al. 2002). However, some types of context information are difficult for application designers to deal with. As many as five or six low-level sensors, each of which output a wide range of values, may be required to take account of a single higher-level parameter. Consider for example, the weather. Humans perceive the weather as a whole and are able to make a useful estimate of current conditions quickly. Weather has though many parameters, including temperature, light-level, rain, wind, humidity and barometric pressure.

Here, difficulties arise not only in acquiring context but also in representing contextual information, deriving higher level contextual information and determining a response appropriate to the combination of parameters that represent the weather to a computer. A designer of a tour guide application for example, may simply want to display hyperlinks to beaches or indoor activities depending on the state of the weather, time and season. At present, there is very little support for a designer who wishes to incorporate such a behaviour into an application, other than to write the code for the relevant contexts and corresponding behaviours and compile this as a component of the application.

Handling context within application code increases the complexity of that code, making it more difficult to work with. To compound this problem, changes in context reasoning and context behaviours can only be achieved by re-writing and re-compiling, with changes necessarily made by programmers rather than by domain experts or indeed users.

A further difficulty facing context-aware application designers is the constrained nature of small mobile computers in terms of processing power, memory and persistent storage. Satyanarayanan asserts that for a given cost and level of technology, considerations of weight, power, size and ergonomics exact a penalty in terms of computational resources, so that while mobile elements may improve in absolute terms, they will always be resource-poor relative to static elements (Satyanaryanan 1996). However, the burden of deriving high level context abstractions can be expected to lead to increased processing, memory and persistent storage requirements.

3 CASS Overview

In this section we describe how CASS provides mobile context-aware application designers with support for high-level context derivation. CASS addresses the issue of separation of context-aware application code from high-level context reasoning and behaviours. More precisely, this would allow an application's context reasoning and resulting behaviours to be changed without re-compilation. This opens the way to changes and indeed extensions being undertaken by the owners of an application rather than programmers.

3.1 Requirements

(Dey et al. 2001) as well as (Schmidt & Gellersen 2001) emphasise the importance of the separation of concerns between sources and users of context. A middleware must also be flexible enough to provide for the presentation of information and services, automatic execution and the attaching of context to information. These categories of context-aware application features were identified in (Dey & Abowd 2000). In addition to these, we have gathered the following set of requirements for CASS. The sources range from work presented in (Dey et al. 2001), (Schmidt & Gellersen 2001) and our own perspective on the contribution CASS can make:

- ❑ Support a large number of context sources, including multiple sources of the same context.
- ❑ Provision for context history.
- ❑ Support for context interpretation. For example raw temperature reading to reading in degrees Celsius.
- ❑ Support higher-level abstraction of contexts while taking into account the processing and storage constraints on small mobile devices.
- ❑ The middleware must be event-based, so that there is no requirement on applications to poll for changes in context.
- ❑ The middleware must be extensible -For example to accommodate further contexts and sensors but also to accommodate tools that would allow easier configuration of context rules and behaviours.
- ❑ Support transparent use of distributed sources of context.
- ❑ Support the separation of application procedure from context-awareness rules and behaviours.

3.2 Architecture

The CASS middleware (Fig. 1) is server-based with context-aware applications on mobile computers connecting over wireless networks. Sensor nodes are computers with sensors attached and may be mobile or static. One or more sensors may be attached locally at a sensor node. Because the middleware is server-based, it does not suffer from the processor and memory constraints that would apply on a mobile computer. This allows use of a database and artificial intelligence components as required, as well as the facility to store large amounts of data. However there is a reliance on communications between the mobile platform and the server hosting the middleware. Because of this reliance, CASS supports applications in the use of local caching of information to reduce the effect of intermittent connections.

CASS applications need not communicate with each individual source of context directly but only with the middleware and therefore do not need to store low-level details of context sources.

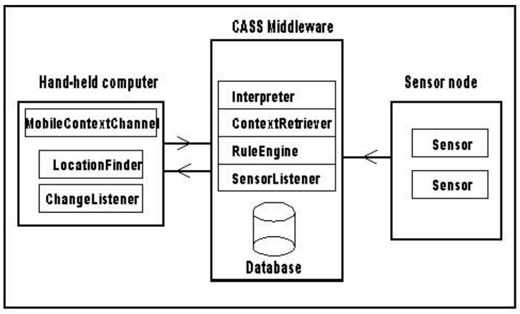

Fig. 1. CASS architecture

3.3 Design

The classes that belong to the CASS middleware are shown in Fig. 2. Central to the design is the RuleEngine class. Three subclasses of RuleEngine correspond to the categories of context-awareness application features identified in (Dey & Abowd 2000).

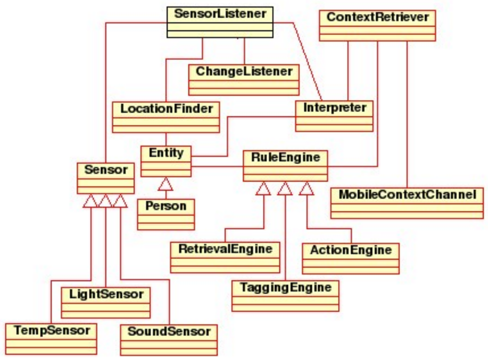

Fig. 2. CASS design

The SensorListener class listens for updates from Sensors and stores context information while the ContextRetriever class retrieves stored context. Both of these classes may use the services of an interpreter. The ChangeListener is a component with communications capabilities, that allows a mobile computer to listen for notification of context change events. Sensor and LocationFinder classes also have built-in communications capabilities.

3.4 Data Management

Given the requirement to support context history, as outlined in Section 3.1, as well as the need to manage data relating to multiple sensors, applications and users, the CASS middleware uses a database for persistent data storage. The database, is server-based and therefore does not suffer from the storage space and performance constraints that apply on small mobile platforms.

Although databases make it easier to handle large volumes of data and contribute to the enforcement of standards (McFadden & Hoffer 1994), they also have other advantages to offer as a means of persistent storage for context-awareness:

❑ Data can be read and manipulated at a high level of abstraction using Structured Query Language (SQL).

❑ The database can be used to store, not only context, application and user data but also domain knowledge in the form of context-awareness rules and behaviours relevant to specific applications.

With respect to this last point, the storage of domain knowledge, independent of context-aware applications supports the context reasoning and behaviours of specific applications to be changed in a dynamic way. Further, by providing a suitable interface to the database containing the context inference rules, these rules could then be updated and changed by users rather than programmers.

CASS applications use database tables to store rules about high-level context together with related information to be displayed to users, actions to be executed or attached information tags. The database also stores data on entities, applications and their context interests, context values, context sources, groups and profiles.

4 Context Inference Mechanism

In this section we describe how high-level context is inferred using an inference engine and knowledge base. We also explain the technique of abstraction and search used by CASS to infer high-level context information.

4.1 Inference Engine

An inference engine works in conjunction with a knowledge base and uses the rules contained in the knowledge base to solve problems. It is essentially an interpreter for the knowledge base (Luger & Stubblefield, 1998a). For example, given a certain set of conditions or parameters, the inference engine consults its knowledge base to find a matching set or parameters and a corresponding goal or solution to the problem.

The separation of the knowledge base from the inference engine is desirable from at least two standpoints (Luger & Stubblefield, 1998a):

❑ Separation makes it possible to represent knowledge in a more natural fashion, particularly for domain experts without any knowledge of software.

- The knowledge base can be updated and changed without changing the inference engine implementation.

The CASS inference engine is used to find a matching goal or goals when a change in context is detected. Having a separate knowledge base means that changes can be made to context inferences and goals relevant to an application without necessitating changes in application code.

The CASS inference engine uses a technique called forward chaining, where known facts are used to infer other facts and these facts can be used in turn to infer further facts. Forward chaining is a search technique useful for situations where the search space is wide with many potential goals (Luger & Stubblefield, 1994a), which is the case with context-aware systems. For example, many context inputs lead to fewer higher-level contexts and eventually to a goal represented by some application behaviour. This behaviour might be to display some information, execute some service or attach context to information.

4.2 Knowledge Base

The knowledge base contains the rules used by the inference engine described above. CASS stores application knowledge bases as database tables where the records contain high-level context states and a corresponding context behaviour, the goal.
The usefulness of CASS knowledge base tables is partly due to the close alignment with high-level context abstractions like, for example, the weather. An example of a rule from a table containing weather states is shown in Fig. 3. A tour-guide application might use such a rule to allow it to display hyperlinks to indoor activities. In this case the goal 'indoor', could be used as a key into a table containing the hyperlinks. Alternatively, the goal could be used in a further rule that also takes account of a 'location and time' goal obtained from another CASS table. This could then be used to show only nearby activities that were currently open to visitors.

Rain	Brightness	Temp	Goal
wet	dull	cold	indoor

Fig 3. An example of a weather context rule

4.3 Abstraction and Search

Abstraction allows the expression of only the information needed for a given purpose (Luger & Stubblefield, 1998b). Higher levels of abstraction lose some information but make a problem manageable. Abstraction is used by the CASS inference engine to reduce the state-space of individual context components. For example, an ambient temperature range of 30 degrees Celsius could be abstracted to as few as three states and these represented in corresponding semantic variables of Cold, Normal and Hot as suggested in the context ontology vocabulary described in (Korpipää et al 2003). Such abstractions can be based on crisp limits or Fuzzy sets. Our current implementation uses context abstractions with crisp limits. Context abstractions together with their limits are stored in a database table and are adjustable without re-compiling or even re-starting an application.

Having abstracted each context component to give a manageable state space that is appropriate to the needs of the context aware-application, the knowledge base for that application can then be searched for a goal or application behaviour corresponding to the state of these context components. This context-aware behaviour is then activated. Since the knowledge base is contained in a database table, SQL provides a convenient means of searching.

5 Evaluation

Mark Weiser describes the construction of working prototypes, as an important part of the research effort at Xerox PARC. Indeed a key part of the design philosophy was to put devices into everyday use (Weiser, 1993). The emphasis in the CASS project too, is on gaining real experience with context-awareness.

5.1 Applications

As part of our evaluation of the CASS framework, we are currently developing two applications. MALLET (Maintenance Assignment Listing Lightweight Electronic Tool) is an application that allows domestic and building maintenance tasks to be organised in a context-aware way. Tasks can be associated with context and displayed when the context conditions are satisfied. For example, a pending maintenance task like exterior painting would be displayed by the application in dry, bright, warm conditions. Tasks can also be attached to location and time so that a quick tour of a building for example, will show what tasks need to be carried out at different locations. Override functions are provided for user convenience. For example, a user can easily force the display of all tasks regardless of context.

A second application called STONE (STart ON Entrance) is designed to avoid the awkward start to presentations and lectures involving slideshows. Typically, the presenter must switch on the beam projector, start the presentation application and find the correct presentation file as the audience waits. With STONE, a presenter can enable these preparations to happen automatically as soon as the presenter enters the presentation area and if required, present an introductory slide. We are also investigating ways of automating the advancement of slides based on context information such as pauses in speech, elapsed display time and presenter orientation.

5.2 Sensors

We are currently investigating aspects of sensor-derived context including sensor failure, sensor fusion and error handling and have built several sensor prototypes to assist in this work. The two prototypes shown in Fig.4 are designed to connect to a static sensor node which sends readings over a network to the CASS middleware. The sensor hardware is based on inexpensive and widely available analogue to digital converter chips. Software abstracts the low-level details of the hardware, such as how readings are taken, for example, and provides communications support. The sensor software is extensible to allow new types of sensor to be added.

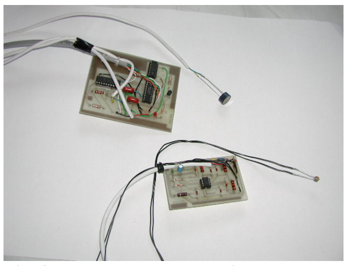
Fig 4. Temperature and light sensors.

To support indoor location awareness, we have built a system based on infrared beacons (Fig. 5). A key feature of the system is that the receiver (Fig. 7) uses the infrared port on the PDA and so leaves other ports free for use with a wireless communications device and a GPS receiver for example. Hardware support can thus be provided for both indoor and outdoor location awareness as well as communications with the CASS middleware. The indoor location system has a range of 5m and is not highly directional in confined spaces, as in a small office for example due to reflection from walls and other surfaces.

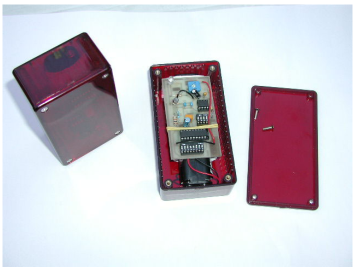
Fig. 5. Infrared location beacons.

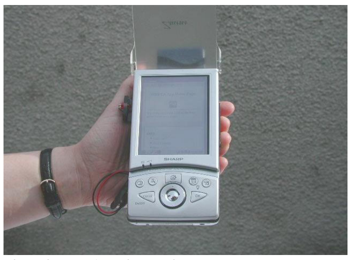
Fig. 6. PDA with wireless card and location receiver.

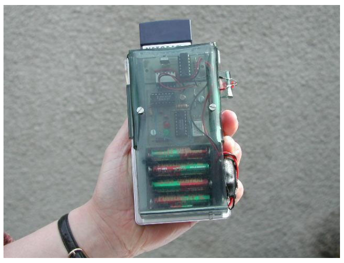
Fig. 7. PDA view from back showing location receiver hardware.

6 Related Work

The concepts of context and context-awareness have been analysed in (Dey and Abowd 2000). This work improves on previous definitions of context and context-awareness and also identifies categories of context and categories of features for context-aware applications.

The Context Toolkit (Dey et al. 2001) supports the separation of concerns between the acquisition of context data from sensors and its use in applications. It also provides developers with re-usable components and an accompanying conceptual framework. The Context Toolkit differs from our middleware approach, in that our work also supports the separation of context-awareness rules from application code.

Situated Computing (Hull et al. 1997) is concerned with the detection, interpretation and response to aspects of the user's local environment. It is essentially a layer of middleware that relieves the applications of the task of collecting and dealing with raw sensor data. A prototype SitComp service was built using one type of sensor, namely active tag detectors. Situated Computing confined context sensing to the immediate environment of the user.

The Owl Context Service (Ebling et al. 2001) is based on an architecture that provides context services as part of the infrastructure. Design considerations for the Owl architecture include scalability, extensibility and privacy. Owl accommodates heterogeneous context sources. Owl differs from our work in that our work concentrates on providing context information at a higher level of abstraction and emphasises the separation of context abstractions from application code.

7 Conclusion

We have presented a server-based middleware that supports context-aware applications on hand-held and other small computers. The middleware supports the use of large numbers of context inputs and provides for high-level context abstractions while placing no additional processing or memory load on constrained mobile platforms. The middleware allows context inferences to be separated from application code in a manner that opens the way to configuration by users rather than necessitating re-programming. Our immediate goal is to develop context-awareness configuration tools for use with CASS and applications that take advantage of CASS features.

8 References

(Chen & Kotz 2002) Chen G. & Kotz D. Context Aggregation and Dissemination in Ubiquitous Computing Systems. In Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, 2002 pp 105 -114

(Dey and Abowd 2000) Dey, A.K., Abowd, G.D. Towards a Better Understanding of Context and Context-Awareness, presented at the CHI 2000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness, April 1-6.

(Dey et al 2001) Dey A.K., Abowd G.D. & Salber D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications, Human Computer Interaction, 2001, Volume 16, pp 97-166

(Ebling et al. 2001) Ebling Maria R., Hunt Guerney D. H. and Lei, Hui. Issues for Context Services for Pervasive Computing. Workshop on Middleware for Mobile Computing, November 2001

(Gellersen et al. 2002) Gellersen H.-W., Schmidt A., Beigl M. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts, *Mobile Networks and Applications (MONET)*, Kluwer, Oct 2002.

(Hull et al. 1997) Hull R., Neaves P. and Bedford-Roberts J. Towards Situated Computing, in Proceedings of IEEE ISWC'97 First International Symposium on Wearable Computers, Cambridge, MA, USA 1997.

(Korpipää et al. 2003) Korpipää P., Mäntyjärvi J., Kela J., Keränen H. & Malm E.J. (2003). Managing Context Information in Mobile Devices, IEEE Pervasive Computing 1536-1268-03.

(Luger & Stubblefield 1998a) Luger G.F. & Stubblefield W.A. Artificial Intelligence, Structures and Strategies for Complex Problem Solving. Third Edition, Addison Wesley Longman, MA. U.S.A., 1998 pp210.

(Luger & Stubblefield 1998b) Luger G.F. & Stubblefield W.A. Artificial Intelligence, Structures and Strategies for Complex Problem Solving. Third Edition, Addison Wesley Longman, MA. U.S.A., 1998 pp34.

(McFadden & Hoffer 1994) McFadden F.R. & Hoffer J.A. Modern Database Management, Fourth Edition, Benjamin Cummings Publishing Company, CA. 1994 pp 18-20

(Salber & Abowd 1998) Salber D. & Abowd G. The Design and Use of a Generic Context Server, Georgia Tech GVU Technical Report GIT-GVU-98-32

(Satyanarayanan 1996) Satyanarayanan M. Fundamental Challenges in Mobile Computing. In Proceedings of the ACM Symposium on Principles of Distributed Computing.

(Schmidt et al. 1999) Schmidt A., Beigl M. & Gellersen H.-W. There is More to Context than Location, Computer & Graphics 23(6) December 1999 pp. 893-901

(Schmidt et al 2001) Schmidt A. & Gellersen H.-W. Modell, Architektur und Plattform fuer Informationssysteme mit Kontextbezug,. Informatik Forschung und Entwicklung 2001 16:213-214.

(Weiser 1993) Weiser M. Some Computer Science Issues in Ubiquitous Computing , CACM July 1993.