A Framework for Developing Mobile, Context-aware Applications

Gregory Biegel and Vinny Cahill
Distributed Systems Group
Department of Computer Science
Trinity College Dublin
{*Greg.Biegel, Vinny.Cahill*}*@cs.tcd.ie*

Abstract

The emergence of truly ubiquitous computing, enabled by the availability of mobile, heterogeneous devices that supply context information, is currently hampered by the lack of programming support for the design and development of context-aware applications.

We have developed a framework which significantly eases the development of mobile, context-aware applications. The framework allows developers to fuse data from disparate sensors, represent application context, and reason efficiently about context, without the need to write complex code. An event based communication paradigm designed specifically for ad-hoc wireless environments is incorporated, which supports loose coupling between sensors, actuators and application components. [1]

1. Introduction

Context awareness and mobility are core concepts in the vision of *ubiquitous computing* where networks of small computing devices are dispersed in the physical environment, operating autonomously and independently of centralised control. Context-aware applications are a large and important subset of the overall set of ubiquitous computing applications, and have already demonstrated the advantages gained from the ability to perceive the surrounding environment [1], [2], [3]. Such applications however remain difficult to develop and deploy, with no widely accepted programming model available. Programmers are often required to write large amounts of code and interact with sensor and actuator devices at a low level in order to develop relatively simple applications.

The mobility of devices in the ubiquitous computing environment also raises challenges in the areas of communication and interaction due to factors such as dynamically changing network addresses and system configurations, susceptibility to disconnection and low bandwidth [4].

The main components of a context-aware application are a set of sensors for capture of context data, a set of rules governing behaviour according to context and a set of actuators for generating responses. We have developed the *sentient object model* for the development of context-aware applications in an ad-hoc mobile environment, which defines software abstractions for sensors and actuators, and provides a framework for the specification of production rule driven behaviour. Sentient objects have a number of characteristics that are important in ubiquitous computing environments:

- *Sentience* - the ability to perceive the state of the environment via sensors
- *Autonomy* - the ability to operate independently of human control in a decentralised manner
- *Proactiveness* - the ability to act in anticipation of future goals or problems

The challenges introduced by mobile computing environments are addressed by an *event-based* communication mechanism, which does not rely on centralised control and provides loose coupling between objects, supporting object mobility and application evolution. The event-based communication model allows sentient objects to treat all input data, whether from sensors or other objects, in the same manner, providing a decoupled approach to interaction. The sentient object model incorporates the STEAM event service [5] to provide communication among components of the model.

Our model provides a systematic approach to the development of context-aware applications in mobile ad-hoc environments, supporting the important aspects of sensor fu-

1 The work described in this paper was partly supported by the Future and Emerging Technologies programme of the Commission of the European Union under research contract IST-2000-26031 (CORTEX). The authors are grateful to the anonymous reviewers and our shepherd, Roy Want, for their valuable contribution to improving this paper.

sion, context extraction and reasoning. We provide a framework that supports the application developer in the following key ways:

- It provides abstractions for sensors and actuators, thus relieving the developer of the burden of low level interaction with various hardware devices
- It provides a probabilistic mechanism for fusing multi-modal fragments of sensor data together in order to derive higher-level context information
- It provides an efficient approach to intelligent reasoning based on a hierarchy of contexts
- It provides an event-based communication mechanism for interaction between sensors, objects and actuators
- It provides an easily accessible visual programming tool for developing applications reducing the need to write code

The framework fulfills the two major goals identified by Dey and Sohn [6] which are necessary for the successful development of ubiquitous, context-aware applications, namely:

1. Applications are easier to design, prototype and test, supporting a faster iterative development process
2. Designers and end-users are empowered to build their own applications

A number of other middleware proposals address the challenges of effectively developing context-aware applications. Seminal work by Dey [7] provided a toolkit which enabled the integration of context data into applications, but did not provide mechanisms for performing sensor fusion, reasoning about context, or dealing with mobility. Context acquisition and use was often tightly integrated into a single application [2], and could not easily be incorporated into other applications. Other work provided mechanisms for reasoning about context [1], [8], but still did not provide a well defined programming model and did not address the challenges of mobility. Recent and ongoing work [6], [9] provides programmer support for the development of context-aware applications, but does not provide the ability to systematically specify and manage event filtering, sensor-fusion and rule-based inference in a mobile ad-hoc environment, as our framework does.

2. The sentient object model

Essentially, a *sentient object* is an encapsulated entity, with its interfaces being sensors and actuators. Actuation is controlled based on sensor input according to internal control logic, consisting of event filtering, sensor fusion and intelligent inference, as illustrated in Figure 1. In the sentient object model, sensors are defined as entities that *produce* software events in reaction to a real world stimulus, whilst actuators are defined as entities which *consume* software events and react by attempting to change the state of the real world in some way via some hardware device.

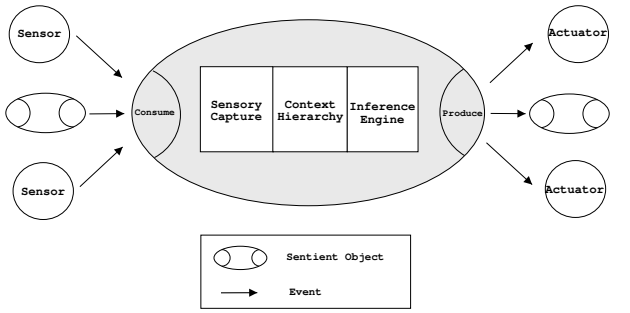

Figure 1. Sentient object model

2.1. Sentient object internals

The sentient object model is based on the notion of context-awareness, where *context* is defined as any information sensed from the environment which may be used to describe the situation of a sentient object. Context awareness depends on the accurate extraction, combination and interpretation of information from a variety of unreliable, multi-modal sensors and sentient objects are composed of 3 major internal components which provide these functions:

2.1.1. Sensory capture and context fusion The sensory capture component of a sentient object performs *sensor fusion* in order to manage uncertainty of sensor data and derive higher level context information from multi-modal data sources.

A probabilistic sensor fusion scheme is employed, based upon Bayesian networks [10], which provides a powerful mechanism for measuring the effectiveness of derivations of context from noisy sensor data. A Bayesian network graph allows us to manage the exponential increase in the size of a joint probability distribution over a set of random variables, by exploiting conditional independence. Using Bayesian networks to model the uncertainty of sensor data and the dependencies between a set of sensors, gives us the ability to efficiently reason about the truth of a hypothesis, given evidence from a set of sensors.

2.1.2. Context hierarchy The overall context of a sentient object is made up of a set of discrete environmental facts and data. These multi-modal context fragments are fused by the sensory capture component to determine higher level *contexts*. The set of contexts in which an object may

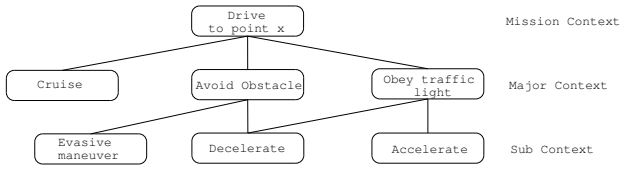

Figure 2. Subset of the context hierarchy for a simple sentient car

exist is represented as a hierarchy, based upon the Context-Based Reasoning (CxBR) paradigm [11].

This paradigm derives its name from the idea that the actions taken by an entity are highly dependent on the entity's current context, i.e., a recognized situation defines a set of appropriate actions and the identification of a future situation is simplified if all possible actions are limited by the current situation itself. CxBR is based on the following hypotheses [11]:

1. Small, but important portions of all available environmental inputs are used to recognise and treat the key features of a situation
2. There are a limited number of things that can realistically take place in any situation
3. The presence of a new situation will generally require alteration of the present course of action to some degree

The context hierarchy encapsulates knowledge about actions to be taken and possible future situations into contexts. By defining a hierarchy of contexts in which a sentient object may exist and by associating specific actions to be undertaken in each context, a sentient object's behaviour is influenced by its context.

An example context hierarchy for a simple sentient model car is illustrated in Figure 2. The mission context captures actions common to all contexts whilst a major context captures actions common to a set of sub contexts which in turn are usually actions with a short lifetime which change rapidly. Only one mission, major and sub context is active at any point in time which limits the actions possible and therefore the rules applicable and in need of evaluation. For example, in the sentient car when the major context **Avoid Obstacle**, and sub context **Decelerate** are active, rules and actions associated with other major and sub contexts do not need to be evaluated, which increases the efficiency of the inference process.

Treating context as a hierarchy helps to manage the complexity of development by reducing the problem to developing a set of 'contexts'. Within each context only a subset of the overall rule-base of the system needs to be specified which in turn reduces the complexity of rule development.

2.1.3. Inference engine Sentient objects are made context-aware by using conditional rules to specify application behaviour in different contexts, in other words the objects follow an *Event-Condition-Action* execution model [8].

The inference engine component is responsible for changing application behaviour according to context and leverages the existing capabilities of the CLIPS (C Language Integrated Production System) production system language [12]. CLIPS contains an inference engine built into the language which given a set of *facts*, and predefined *rules*, is able to decide which rule to fire. Intelligence and the ability to react to changing contexts is added to the object through the specification of a set of production rules.

CLIPS' suitability for integration into reactive sentient applications is explained by [8] as: (1) the expressive power of CLIPS permits specification of complex relations of event patterns; (2) its built-in inference engine implements the RETE [13] algorithm, a very efficient mechanism to solve the many-to-many matching problem; (3) CLIPS is designed for full integration and extensibility with procedural languages such as C++ and Java.

The context hierarchy increases the efficiency of the inference engine by introducing the notion of an *active context*. Within the context hierarchy, only one context is active at any point in time and within this context, only a subset of the overall rules governing the objects behaviour are active. This derives from the hypothesis that there are only a limited number of things that can realistically take place in any situation, and by limiting the number of actions permitted in specific contexts, the efficiency of production-rule based inference is increased substantially. Rule based inference is typically $O(n)$, where n is the number of rules in the system but the introduction of the context hierarchy improves this to $O(k)$ where k is the average number of rules in an active context and $k < n$ [11]. Experiments [14] demonstrated that the introduction of a context hierarchy reduces the overall number of rules needed for inference, as well as speeding up the process.

2.1.4. Sensor fusion and the context hierarchy The fact that only a small portion of sensory input is relevant at any point in time is used to enhance the effectiveness of the probabilistic sensor fusion scheme by limiting the number of nodes in the sensor fusion network in each context.

Sensors in the sentient object model are highly distributed, with changing configurations due to the mobility of sentient objects. In addition, which of a set of sensors are consulted at a particular point in time is highly dependent on the active context at that time. We perform sensor

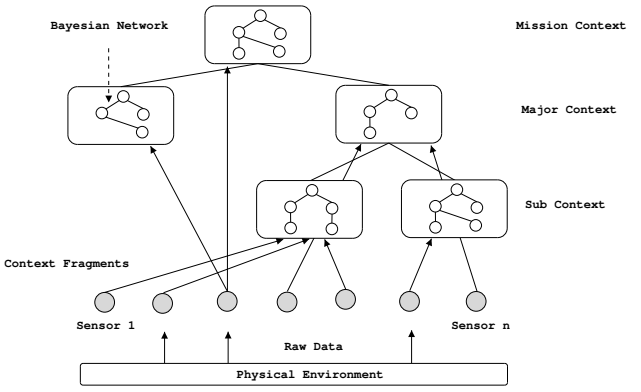

Figure 3. Sensor fusion in the context hierarchy

fusion at the level of a context within the context hierarchy. A context defines which sensors are relevant to that context as well as those which must be monitored in order to detect when transition to another context is indicated. A Bayesian network is constructed within each context in order to fuse the fragments of context information obtained from the sensors. The integration of Bayesian network fragments into the context hierarchy is illustrated in Figure 3. This figure shows how each context in the hierarchy is only interested in a subset of the sensor input, and Bayesian network fragments within each context act to fuse the context fragments obtained from the relevant sensors.

2.2. Event based communication

Event-based communication is well-suited for mobile applications where communication relationships amongst application components are established dynamically during the lifetime of the application [5]. STEAM (Scalable Timed Events And Mobility) is a location-aware event-based middleware service designed specifically for *ad-hoc* wireless networking environments. STEAM differs from other distributed event middleware in that it does not depend on any fixed communication infrastructure and event notifications may be bounded based on geographical location.

STEAM contains no centralized components and provides a number of types of filter to control propagation of events through the system. In addition to subject and content filters, STEAM provides **proximity** filters which define the geographical validity of an event.

3. The sentient object programming model

We provide a programming model, based on the sentient object model and incorporating the STEAM event service, which provides abstractions for the development of mobile, context-aware applications.

3.1. Programming sensors and actuators

Sensors are developed as software abstractions that produce STEAM events, whilst actuators are developed as software abstractions that consume STEAM events. These software components encapsulate and act as wrappers for hardware and software sensors and provide mappings between specific sensor protocols and proprietary data formats, and STEAM events. Sensors provide a uniform interface to sensory information through STEAM events, and hide details of the underlying sensing technologies.

Actuators extend from STEAM *consumer* entities which export an API for subscription management. Actuators consume STEAM events according to active subscriptions and filters, and transform the information contained within these events, to specific hardware or software commands. Actuators *subscribe* to event types of interest based on a set of event filters.

In addition to the class file, each sensor and actuator also contains an XML description file, which contains information about the events produced or consumed, as well as probabilistic information regarding the uncertainty of each event.

3.2. Programming sentient objects

Sentient objects are developed using a graphical development tool which allows developers to specify relevant sensors and actuators, define fusion networks, specify context hierarchies and production rules, without the need to write any code.

3.2.1. Specifying inputs and outputs Specification of inputs to the object is done simply through 'drag and drop' from libraries which contain XML descriptions of sensors and sentient objects. Each descriptor describes the name and type of each STEAM event produced by the sensor or object. Outputs are specified in the same way, with descriptors available describing the events consumed by the actuator or object. Much of the programming effort is concentrated in the specification of the logic of the object.

3.2.2. Specifying contexts Specification of the context hierarchy is the next step in the development of an object. The hierarchy is built up by specifying the attributes of each individual context. The most important attributes of a context are: (1) The set of events which are of interest in this context; (2) The set of rules which are active in the context; (3) The set of other contexts to which this context is related (i.e. child contexts) and may transition to; (4) The conditions under which transition to another active context occurs.

In effect, a context acts as a composite event filter - whilst it is active, only those events which are defined as being of interest in that context are delivered to the object.

A context defines the behaviours which are appropriate within the context by specifying the set of production rules which are active, and consequently evaluated, in each context. The fact that only a subset of sensor inputs and production rules needs to be considered for each individual context aids in reducing the complexity of developing context-aware applications.

3.2.3. Specifying fusion services Fusion services within sentient objects are defined at the level of a context, using Bayesian networks. The tool allows the user to specify a Bayesian network for fusing fragments of context data on a per-context basis, via an intuitive graphical network builder.

The network is constructed by defining the events of interest and their relationships by adding nodes and arcs in an interactive manner. The probabilities of all root nodes are then specified and a conditional probability table (CPT) is constructed for each non root node. These tables capture prior probabilities of events and rely on the a priori availability of probability data. This may be determined through experimental evidence as was done for ultrasonic sensors in the development of our sentient model car, where the probability distribution that an arbitrary sensor reading was within a specified threshold was calculated based on experimental observation.

3.2.4. Specifying rules The complexity of specifying effective rules and knowledge bases is one of the greatest challenges in context-aware application development. Sentient objects make use of an embedded CLIPS inference engine, but CLIPS syntax is complex and not easily assimilated by the majority of developers. We attempt to make knowledge capture more accessible to domain experts by providing a high level, graphical rule builder, allowing the definition of application behaviour at the level of a context, and hiding the complexities of CLIPS syntax from the user.

4. Conclusion

In this paper we have presented a framework, based on the sentient object model, for developing context-aware applications in a mobile, ad-hoc environment. Our framework provides a systematic approach to context-aware application development, including the ability to fuse context fragments and deal with uncertainty in a probabilistic manner, the ability to represent context within the application, and the ability to easily compose rules to reason efficiently about context. This functionality is offered in a tool which is easily accessible to a wide range of developers, permitting the rapid design and development of applications, based on sentient objects, for ubiquitous computing environments. We have successfully applied our framework to the development of a number of proof-of-concept applications, including a simple sentient model car application which drives and obeys traffic signals autonomously.

References

[1] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. *Advanced Interaction in Context* in Handheld and Ubiquitous Computing Springer-Verlag, pp. 89-101, 1999.

[2] Mike Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, Andy Hopper *Implementing a Sentient Computing System* IEEE Computer Magazine, Vol. 34, No. 8, pp. 50-56, August 2001.

[3] Diego López de Ipiña, Paulo Mendona and Andy Hopper *TRIP: a Low-Cost Vision-Based Location System for Ubiquitous Computing* Personal and Ubiquitous Computing journal, Springer, vol. 6, no. 3, pp. 206-219, May 2002.

[4] George H. Forman, John Zahorjan *The Challenges of Mobile Computing* IEEE Computer, 27(6), April 1994.

[5] René Meier and Vinny Cahill *Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications* in Proceedings of the 4^{th} IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03). Paris, France: Springer-Verlag Heidelberg, Germany, 2003.

[6] Anind K. Dey and Tim Sohn *Supporting End User Programming of Context-Aware Applications* Conference on Human Factors in Computing Systems (CHI) Workshop on Perspectives in End User Development, Fort Lauderdale, FL, April 5-10, 2003.

[7] Anind K. Dey, Daniel Salber and Gregory D. Abowd *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications* Human-Computer Interaction (HCI) Journal, Volume 16 (2-4), pp. 97-166, 2001.

[8] Diego López de Ipiña *An ECA Rule-Matching Service for Simpler Development of Reactive Applications* in Proceedings, Middleware 2001 Vol. 2, No. 7, November 2001.

[9] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt *Gaia: A Middleware Infrastructure to Enable Active Spaces* in IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

[10] Judea Pearl *Bayesian Networks Handbook of Brain Theory and Neural Networks*, MIT Press, 2001.

[11] Avelino J. Gonzalez, Robert Ahlers *Context-Based Representation of Intelligent Behaviour in Training Simulations* Transactions of the Society for Computer Simulation International, Vol. 15, No. 4, March 1999.

[12] NASA *CLIPS: A Tool for Building Expert Systems* `http://www.ghg.net/clips/CLIPS.html`

[13] Forgy, C.L *Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem* Artificial Intelligence 19, pp. 17-37, 1982.

[14] Gonzalez, A. J. and Ahlers, R. H. *Context-Based Representation of Intelligent Behavior in Simulated Opponents* Computer Generated Forces and Behavior Representation Conference, 1996.