A Software Engineering Framework for Context-Aware Pervasive Computing*

Karen Henricksen
CRC for Enterprise Distributed Systems Technology and
School of Information Technology and Electrical Engineering, The University of Queensland
kmh@dstc.edu.au

Jadwiga Indulska
School of Information Technology and Electrical Engineering, The University of Queensland
jaga@itee.uq.edu.au

Abstract

There is growing interest in the use of context-awareness as a technique for developing pervasive computing applications that are flexible, adaptable, and capable of acting autonomously on behalf of users. However, context-awareness introduces various software engineering challenges, as well as privacy and usability concerns. In this paper, we present a conceptual framework and software infrastructure that together address known software engineering challenges, and enable further practical exploration of social and usability issues by facilitating the prototyping and fine-tuning of context-aware applications.

1. Motivation

It is widely acknowledged that pervasive computing introduces a radically new set of design challenges when compared with traditional desktop computing. In particular, pervasive computing demands applications that are capable of operating in highly dynamic environments and of placing minimal demands on user attention. Context-aware applications aim to meet these requirements by adapting to selected aspects of the context of use, such as the current location, time and activities of the user.

In recent years, a variety of prototypical context-aware applications have been developed, such as context-aware guides that present tourists with information of relevance to the current location [8, 1]. There are also ongoing efforts to construct instrumented environments that monitor the activities of their occupants using context sensors, with objectives such as allowing the elderly to live as independently as possible while ensuring that emergencies are quickly detected [14, 21].

Despite the recent flurry of interest, context-aware applications have not yet made the transition out of the laboratory and into the marketplace. This is largely a result of high application development overheads, social barriers associated with privacy and usability, and an imperfect understanding of the truly compelling (and commercially viable) uses of context-awareness. This paper presents a software engineering framework that addresses these challenges: the first by simplifying design and implementation tasks associated with context-aware software, and the latter two by facilitating the types of rapid prototyping and experimentation that are required in order to overcome these obstacles. The framework is based around a set of novel conceptual foundations, including context modelling approaches that describe context at two different levels of granularity, a preference abstraction, and a pair of complementary programming models. These are introduced in Sections 2 to 4, and are then integrated into a software infrastructure for pervasive systems in Section 5. Section 6 presents the results of a case study that we carried out to evaluate the conceptual framework and infrastructure, and Section 7 concludes the paper with a discussion of key topics for future research.

2. Context modelling techniques

Much of the recent research in the field of context-awareness has adopted an infrastructure-centred view; that is, it has assumed that the complexity of engineering context-aware applications can be substantially reduced solely through the use of infrastructure responsible for the gathering, managing and dissemination of context information. To this end, a variety of solutions that acquire and interpret context information from sensors

* The work reported in this paper has been funded in part by the Cooperative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).

Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04)
0-7695-2090-1/04 $ 20.00 © 2004 IEEE

[11, 6], and manage integrated repositories of context information that are easily queried by applications [19], have been proposed. While such solutions are important, we argue that an infrastructure-centred view tends to lead to abstractions for describing and programming with context that are not the most natural ones. In an earlier paper [16], we observed that most of the proposed infrastructures are built upon context models that are both informal and lacking in expressive power.

It has been our goal, therefore, to develop a framework that integrates a set of well-defined context modelling and programming abstractions with the types of infrastructural support described above. To this end, we developed the conceptual foundations of our framework first, starting with context modelling as our primary interest. As we set out with the goal of creating tools that could support the engineer in a variety of tasks (not only implementation), we followed the approach advocated by Coutaz and Rey [9] of employing context models that can be refined incrementally throughout the software engineering life-cycle. In Sections 2.2 to 2.4 we present three separate yet closely integrated modelling approaches that we have developed to support (i) the exploration and specification of an application's context requirements, (ii) management of context information stored in a context repository, and (iii) specification of abstract classes of context that are close to the way the programmer and human user view context. First, however, we briefly discuss the features of context information that differentiate the context modelling problem from other information modelling problems.

2.1. Characteristics of context information

Context information can originate from a wide variety of sources, leading to extreme heterogeneity in terms of quality and persistence. While much of the previous research in context-awareness focuses only on fairly homogeneous sets of context information (usually sensed information, but sometimes only location data), we have found rich context models that integrate sensed, static, user-supplied (profiled) and derived information to be the most useful. These four classes of information each display their own distinctive characteristics [16]; for example, sensed context is usually highly dynamic and prone to noise and sensing errors, while user-supplied information is initially very reliable, but is often allowed by users to become out of date. Our context modelling abstractions accommodate all four types.

The problem of imperfect context information is well recognised, and some of its causes have already been described. Some context modelling solutions address part of this problem by allowing context information to be associated with quality metadata, such as certainty and freshness estimates [19]. This approach is not completely adequate, however, as it does not address the modelling of the ambiguous information that arises when multiple sources of context information report different information and the ambiguity cannot be resolved, nor does it allow the information that an aspect of the context is completely unknown to be explicitly represented. Our context modelling abstractions are unique in that they address all of these issues.

2.2. A graphical modelling approach

We developed our first context modelling approach, the Context Modelling Language (CML), as a tool to assist designers with the task of exploring and specifying the context requirements of a context-aware application. CML provides a graphical notation for describing types of information (in terms of *fact types*), their classifications (sensed, static, profiled or derived), relevant quality metadata, and dependencies between different types of information. CML also allows fact types to be annotated to indicate whether ambiguous information is permitted (e.g. multiple alternative location readings), and whether historical information is retained. Finally, it supports a variety of constraints, both general (such as cardinality of relationships) and special-purpose (such as snapshot and lifetime constraints on historical fact types).

Initially, we formulated CML independently of any established information modelling technique. This afforded the flexibility to express the desired concepts in the most flexible way. The results of this initial exploration are presented in [16]. Subsequently, we chose to reformulate the modelling concepts as extensions to Object-Role Modeling (ORM) [13]. ORM was chosen because of its closeness to our original modelling approach, its superior formality and expressiveness in comparison to solutions such as ER, and the presence of a mapping to the relational model (allowing a straightforward representation of a context model as a relational database if desired).

The example context model shown in Figure 1 (a) illustrates CML's graphical notation. This model captures (i) user activities in the form of a temporal fact type that covers past, present and future activities, (ii) associations between users and their communication channels and devices, and (iii) locations of users and devices (both absolute and relative, where the latter is represented as a derived fact type). Each ellipsis depicts an object type (with the value in parentheses describing the representation scheme used for the object type), while each box denotes a role played by an object type within a fact type. The model shows that user and device locations are both sensed and can be populated by alternative facts (i.e., each user or device can have multiple recorded locations, of which at most one is correct). Additionally, each fact about a user or device location has an associated certainty measure that takes the form of a probabil-

ity estimate. Finally, user activity is shown by the model to be dependent on user location. For further details of CML's modelling constructs, the reader is referred to [15].

2.3. Relational representation

We leverage the mapping of ORM to the relational model to create a relational representation of context information that is well suited to context management tasks, such as enforcement of the constraints captured by CML, storage within a database and querying by context-aware applications. Our extension of Halpin's relational mapping procedure [13] to incorporate the context modelling constructs of CML is described in a previous paper [17], and is not repeated here. However, the result of mapping of the example context model shown in Figure 1 (a) to a set of basic relations is shown in Figure 1 (b).

The relational mapping leads to a representation of context that captures abstract fact types as relations and atomic facts as tuples in a database. In order to support reasoning about contexts, we adopt a closed world assumption. This operates roughly as follows. Assume that R is the set of relations belonging to a context model, I is an instantiation of the model, $I(r)$ represents the set of tuples in I belonging to a relation $r \in R$, and $\mathbf{dom}$ is the set of constant values permitted within any instantiation of the model. Then an assertion of the form $r[c_1, ..., c_n]$ (where $r \in R$ and each $c_i \in \mathbf{dom}$ for $1 \leq i \leq n$) is true for I if there is a tuple $< c_1, ..., c_n >$ in $I(r)$, and false otherwise.

As it stands, this simple interpretation does not accommodate uncertain context information. Therefore, we extend it to deal with unknowns (represented by null values in tuples) and ambiguity (represented as alternative facts) using a three-valued logic. An assertion $r[c_1, ..., c_n]$ (where r and $c_1, ..., c_n$ are constrained as before) evaluates to the third logical value (*unknown*) when the tuple $< c_1, ..., c_n >$ is not present within $I(r)$, but a matching tuple is present when one or more of the constants c_i is replaced with the special $null$ value, *or* when the tuple is present, but is ambiguous (that is, there are alternative facts present[1]). An assertion is false when it is neither true nor unknown.

2.4. The situation abstraction

The graphical notation of CML is well suited for use when defining the context information used by a context-aware application, and its relational analogue is a natural choice for context storage and management, but neither serves as a natural programming abstraction. Both describe context information at a finer granularity than is required when describing the abstract contexts that determine application behaviour. In light of this, we developed the situation abstraction as a way to define high-level contexts in terms of the fact abstraction of CML. Situations can be combined, promoting reuse and enabling complex situations to be easily formed incrementally by the programmer. Our situation abstraction is similar to that proposed by Dey and Abowd [10] for use in their CybreMinder application, but is considerably more expressive.

Situations are expressed using a novel form of predicate logic that balances efficient evaluation against expressive power. They are defined as named logical expressions of the form $S(v_1, ..., v_n) : \varphi$, where S is the name of the situation, v_1 to v_n are variables, and φ is a logical expression in which the free variables correspond exactly to the set $\{v_1, ..., v_n\}$. The logical expression combines any number of basic expressions using the logical connectives, and ($\wedge$), or ($\vee$) and not ($\neg$), and special forms of the universal and existential quantifiers. The permitted basic expressions are either equalities (e.g. $t_1 = t_2$), inequalities (e.g. $t_1 \leq t_2$) or assertions of the form $r[t_1, ..., t_n]$, as described in the previous section.

As there are problems associated with evaluating unconstrained quantified expressions (both in terms of efficiency and in relation to so-called *unsafe* expressions [15]), we employ the following restricted forms of quantification that ensure that the quantified variables are immediately bound:

- $\forall x_1, ..., x_i \bullet r[t_1, ..., t_n] \bullet \varphi$
- $\exists x_1, ..., x_i \bullet r[t_1, ..., t_n] \bullet \varphi$

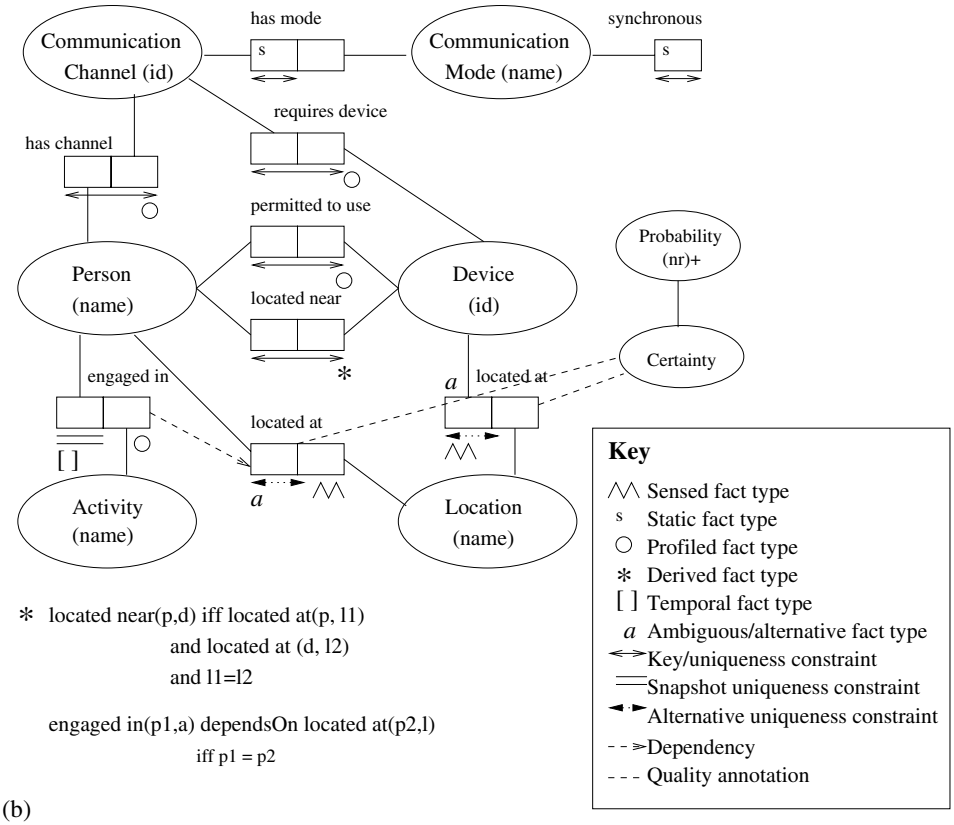
where $\{x_1, ..., x_i\} \subseteq \{t_1, ..., t_n\}$, φ is a logical expression and $r[t_1, ..., t_n]$ is an assertion, as described above[2]. The assertion in the middle of these expressions serves to restrict the possible values for the variables $x_1, ..., x_i$, so that φ is evaluated only over these values.

The evaluation of a situation S against a binding of values for its n variables, $v_1, ..., v_n$, and a context instance, I, occurs according to the usual semantics of the logical operators under a three-valued logic (with the modifications described above for the universal and existential quantifiers), and according to the closed-world interpretation of assertions that was outlined in the previous section. Typically, the variable bindings are supplied by the context-aware application, and describe selected aspects of the application context (such as the identities of participants in a call in the case of a communication application), whereas the context instance is the set of additional context information that is available through a context management system that is separate to the application (this is the set of context information captured by the CML model).

1 For a full definition of alternative facts/tuples, see [15].

2 Note that the symbol "•" acts as a separator and has no special semantics here.

(a)



✻ located near(p,d) iff located at(p, l1)
and located at (d, l2)
and l1=l2

engaged in(p1,a) dependsOn located at(p2,l)
iff p1 = p2

(b)

HasChannel(Person, CommunicationChannel)

PermittedToUse(Person, Device)

EngagedIn(Person, Activity, StartTime, EndTime)

PersonLocatedAt(Person, Location, Probability)

DeviceLocatedAt(Device, Location, Probability)

HasMode(CommunicationChannel, CommunicationMode)

RequiresDevice(CommunicationChannel, Device)

Synchronous(CommunicationMode)

LocatedNear(Person, Device)

Figure 1. (a) An example context model, constructed for the context-aware communication application described in Section 6. (b) Relational mapping of the model shown in (a). Note that the $LocatedNear$ relation, which represents derived context information, would be implemented as a view rather than an ordinary relation.

Some example situations, related to the communication application described by way of a case study in Section 6, are shown in Figure 2. These assume the fact-based context model that was presented in Figure 1. The *Occupied* predicate indicates whether a given person is currently engaged in an activity that generally should not be interrupted ("in meeting" or "taking call"), on the basis of the temporal *EngagedIn* relation. This predicate examines exactly those activity facts for which the current time (returned by the function $timenow()$) overlaps with the recorded time interval (providing special treatment for facts that have no recorded start/end time). Similarly, *CanUseChannel* is satisfied for a given person, p, and communication channel, c, when all of the devices required in order to use c are located in close proximity to p, and p additionally has permission to use these devices. The *SynchronousMode* predicate holds for a given communication channel provided that the mode of this channel (as recorded by the *HasMode* relation) is synchronous (indicated by its appearance in the *Synchronous* relation). Finally, the simple *Urgent* predicate is satisfied

$Occupied(person)$:

$\exists t_1, t_2, activity \bullet EngagedIn[person, activity, t_1, t_2] \bullet$
$(t_1 \leq timenow() \wedge (timenow() \leq t_2 \vee isnull(t_2)) \vee$
$(t_1 \leq timenow() \vee isnull(t_1)) \wedge timenow() \leq t_2) \wedge$
$(activity =$ "in meeting" $\vee\ activity =$ "taking call")

$CanUseChannel(person, channel)$:

$\forall device \bullet RequiresDevice[channel, device] \bullet LocatedNear[person, device] \wedge PermittedToUse[person, device]$

$SynchronousMode(channel)$:

$\forall mode \bullet HasMode[channel, mode] \bullet Synchronous[mode]$

$Urgent(priority)$:

$priority =$ "high"

Figure 2. Example situation predicates for a context-aware communication application. These assume the context model shown in Figure 1.

whenever the $priority$ variable has the value "high".

3. Preference model

There is growing recognition of common usability challenges associated with context-aware software related largely to a lack of transparency of, and user control over, applications' actions [7], and a consequent need for improved techniques for eliciting and capturing user requirements and preferences [5, 18]. To date, however, there has been very little research addressing these issues. One exception is the recent work of Byun and Cheverst, which explored the integration of user modelling techniques into context-aware applications in order to automatically elicit and adapt to user requirements [5]. This work appears promising; however, we argue that explicit means of representing user preferences are also required. The use of an explicit representation allows users to formulate their own preferences if desired, and also provides a tool for exposing preference information and thereby providing transparency, so that users are able to understand the motivation for their applications' actions, and to make corrections as necessary. An explicit representation of preferences can also be compatible with automated learning techniques similar to those used by Byun and Cheverst.

We surveyed a variety of preference modelling approaches, both in the area of context-awareness and in other fields such as decision theory, information systems and document retrieval, with the aim of identifying a preference abstraction that we could exploit within our software engineering framework to support highly dynamic, customisable context-aware behaviour. Within context-aware systems, preferences are sometimes viewed as a type of context, and modelled in the same manner as other context information; this is the approach taken by CC/PP [20]. This solution is suitable for expressing very simple requirements (such as the set of languages that are acceptable for presenting information to a user), but not for more sophisticated, context-dependent preferences. We also encountered difficulties in our attempt to borrow a preference modelling solution from another field, as none of the approaches we evaluated offered a natural way to incorporate context as a determinant in preferences [15].

Accordingly, we developed a novel preference modelling technique based on the situation abstraction. This was designed to support straightforward composition of preferences (such that users can express a set of simple and possibly conflicting requirements and later combine these to form more exhaustive preference descriptions) and compatibility with automated preference elicitation techniques.

Our preference model supports the ranking of a set of candidate choices (such as communication channels that can be used for interactions between users in the case of a communication application) according to the context. Each preference takes the form of a named pair consisting of a scope and a scoring expression. The scope describes the context in which the preference applies in terms of situations. The scoring expression assigns a score to a candidate choice, where the score is either a numerical value in the range [0,1], such that increasing scores represent increasing desirability, or one of the special values $\natural$, $\bot$, $\overline{\wedge}$ or ?. Here, $\natural$ represents a veto (indicating that the candidate to which the score is assigned should *not* be selected in the corresponding context), while $\bot$ represents indifference or an absence of preference. The score $\overline{\wedge}$ represents obligation (that is, that the candidate to which the score is assigned *must* be selected in the corresponding context), and ? repre-

sents an undefined score (signalling an error condition).

Preferences are grouped into sets, and combined according to policies such that a single score is produced for each choice that reflects all of the preferences in the set. User feedback can be exploited to adapt the policy to better meet user requirements (for example, by redistributing weights assigned to individual preferences) or to add new preferences to the set.

Figure 3 presents some example preferences. The preference name is shown at the left, while the scope and scoring expression are preceded by the keywords `when` and `rate`, respectively. The first preference forbids the use of synchronous channels, such as telephone and videoconferencing channels, when the user does not have access to all of the requisite devices. Preferences `p2` and `p3` together imply that synchronous channels are the preferred choice for urgent calls: `p2` assigns these the highest possible score (1), while `p3` assigns all asynchronous channels (such as email and SMS) a score of 0.5.

It should be noted that the preference format shown in Figure 3 is not exposed directly to users. Instead, users typically select from standard preferences that come prepackaged with their applications, or construct and combine their own preferences using graphical editing tools that supply libraries of predefined situations and scoring policies.

Like situations, preferences are evaluated at run-time against a context instance and a set of choice-specific variable bindings. The use of preferences to program flexible, context-aware behaviour is covered in the following section.

4. Programming models

The crucial role of appropriate abstractions and programming models in the development of flexible context-aware applications has been long recognised [3]; however, very little progress has been made in this area. Most context-aware applications are still programmed with traditional software engineering techniques, which embed the use of context information directly into the source code, leading to largely static behaviour and applications that are difficult to maintain. Similarly, the few programming models that have been proposed, such as the Stick-e note model [2], are generally applicable only to narrow application domains.

In the following sections we describe two general programming models that can be used in conjunction with our situation and preference abstractions. The branching model offers a novel and flexible means to insert context- and preference-dependent decision points into the normal flow of application logic. In contrast, the triggering model has been widely used previously in the programming of adaptive and context-aware applications, but is reformulated here to exploit the situation abstraction as a means of describing context changes.

4.1. Branching

The branching model supports context-dependent choice amongst a set of alternatives. Some example uses of this model are in context-aware information retrieval [4], and in the choice of appropriate communication channels for interactions between users. In these applications, context-dependent choices are typically realised using if or case statements. However, these primitive solutions result in a tight binding of the context model to the application logic, making it difficult to later change the context model as the sensing infrastructure and user requirements evolve. To overcome this problem, we exploit the preference model described in Section 3 in our model of branching. User preference information forms the link between the context and the chosen action(s); that is, preferences assign ratings to the alternatives according to the context and other application parameters, and, based on these ratings, the application selects and invokes one or more actions associated with these. This solution is extremely flexible, as preference information is expressed in an application-neutral format that enables modification and fine-tuning when required (even at run-time) and facilitates sharing of preferences between applications, allowing a set of context-aware applications to provide consistent and coordinated behaviour.

We have implemented support for the branching model in the form of a Java programming toolkit. Selected methods provided by this toolkit are shown in Figure 4. The `rate` method has as its parameters a set of choices, a preference (which is likely to be a composite preference encompassing a diverse set of user requirements), a valuation binding variables contained in the preference to constant values according to the application context, and a `Context` object, which is a wrapper for a repository of context information. It uses these to compute a mapping of choices to scores, which the application can then act on as desired. The next two methods select the single best and the best n choices on the basis of scores assigned by the supplied preference. Similarly, `selectAbove` returns the set of choices whose (numerical) scores lie above a specified threshold, and `selectMandatory` returns the set of choices that are assigned the obligation ($\overline{\wedge}$) score. Each of the last four methods also has an companion method (not shown), which, instead of returning a set of choices, automatically invokes an action associated with the choice. Our experiences with using the toolkit to implement a context-aware communication application are described later in Section 6.

4.2. Triggering

To support an asynchronous style of programming in which actions are automatically invoked in response to context changes, we also provide a trigger mechanism that is

```
p1 =   when  SynchronousMode(channel) ∧ ¬CanUseChannel(callee, channel)
       rate  ♮
p2 =   when  Urgent(priority) ∧ SynchronousMode(channel)
       rate  1
p3 =   when  Urgent(priority) ∧ ¬SynchronousMode(channel)
       rate  0.5
```

Figure 3. Example preferences for the context-aware communication application.

```
Scores rate(Choice[] c, Preference p, Valuation v, Context cx);
Choice selectBest(Choice[] c, Preference p, Valuation v, Context cx);
Choice[] selectBestN(int n, Choice[] c, Preference p, Valuation v, Context cx);
Choice[] selectAbove(Score threshold, Choice[] c, Preference p, Valuation v, Context cx);
Choice[] selectMandatory(Choice[] c, Preference p, Valuation v, Context cx);
```

Figure 4. Selected methods of the branching toolkit.

built upon the situation abstraction. Context changes are described as changes in situation states. As there are three possible states (true, false and unknown), there are six distinct state transitions. Triggers can be associated with any of these transitions, or with sequences of transitions (written $t_1 \rightarrow ... \rightarrow t_n$, where t_1 to t_n are transitions), or with sets of alternative triggers (written $t_1|...|t_n$), only one of which needs to occur in order to invoke the trigger.

We follow the event-condition-action (ECA) model, in which each trigger includes a precondition on the invocation of the specified action that is evaluated upon detection of the trigger event. Like the event, this is specified in terms of situations. Our model also associates each trigger with a lifetime, which is one of the following: `once`, `from <start> until <end>`, `until <end>`, `once` or `n times`.

Two example triggers are shown in Figure 5. The events, conditions and actions are prefixed by the keywords `upon`, `when` and `do`, respectively. Actions are described in natural language for simplicity, but would usually take the form of invocations of relevant source code. The first trigger has the effect of notifying the user ("Emma May") at the conclusion of an engagement (defined in Figure 2 as a meeting or a phone call) of any recent missed calls. The second trigger assumes that a pair of users are involved in a phone call, and monitors their ability to use the current telephone lines, invoking adaptation of the channel when this condition ceases to hold. In both triggers, there are no additional preconditions beyond the detection of the specified event, so the condition is simply the expression $true$.

5. Software infrastructure

Run-time support for the two programming models, as well as for related tasks such as management of context and preference information, is provided by a software infrastructure. In this section, we present a brief overview of the architecture of this infrastructure and the partial implementation we have developed as proof-of-concept.

The infrastructure is organised into loosely coupled layers as shown in Figure 6. The context gathering layer acquires context information from sensors and processes this information, using techniques such as interpretation and data fusion, to bridge the gap between raw sensor output and the level of abstraction (and frequency of updates) required by the context management system (recall from earlier sections that this stores atomic facts). An event notification scheme is used to achieve a loose coupling between the sensing and processing components and the reception layer. This minimises the problems associated with component failures, disconnections and evolution of the sensing infrastructure.

The context reception layer provides a bidirectional mapping between the context gathering and management layers. That is, it translates inputs from the former into the fact-based representation of the latter, and routes queries from the latter to the appropriate components of the former.

The context management layer is responsible for maintaining a set of context models and instantiations of these using the relational representation described in Section 2.3. Typically, each application has its own distinct model (however, applications that perform related tasks may share their models). It is our intention that the context management layer be distributed, to provide good query performance and tolerance of failures and disconnections; however, we currently implement it as a single shared repository built around a relational database.

The query layer provides applications and the adaptation layer with a convenient interface with which to query

upon	$EnterFalse(Occupied(\text{“Emma May”}))$
when	$true$
do	Notify of recent missed calls
always	
upon	$ExitTrue(CanUseChannel(\text{“Emma May”}, \text{“3365 5637”}))$ | $ExitTrue(CanUseChannel(\text{“Michelle Williams”}, \text{“3365 9387”}))$
when	$true$
do	Negotiate new communication channel
once	

Figure 5. Example triggers.

the context management system using the fact and situation abstractions. This supports synchronous queries, as well as asynchronous notifications of the situation changes described in Section 4.2.

The adaptation layer manages common repositories of situation, preference and trigger definitions, and evaluates these on behalf of applications using the services of the query layer. A single repository is shared by a logical grouping of applications, where a group typically comprises all of the applications residing on a single device or belonging to a given user.

Finally, the application layer provides toolkit support for the two programming models. The core functions of the branching toolkit were already described in Section 4.1. The triggering toolkit provides methods for dynamically creating new triggers, as well as for activating and deactivating the existing triggers.

Currently, we have a working prototype that implements the basic functions of the context management, query, adaptation and application layers, focusing on the branching functionality at the latter two. As mentioned earlier, our context management system is based upon a relational database, and implements all of the fact types and some of the constraint types of CML. Our implementation of the query layer is relatively simple, performing a direct mapping of fact and situation queries to SQL queries that can be directly executed on the relational database. The asynchronous notification feature has not yet been implemented, as it is not required by the branching model. The adaptation layer currently stores preference and situation information (in textual form) within files that can be edited directly by the application developer and sophisticated users. We have fully implemented the branching toolkit described in Section 4.1, and we present an evaluation of this toolkit and its underlying abstractions in the following section.

6. Case study: context-aware communication

We developed a context-aware communication application as a testbed for experimenting with the branching and preference models and our context modelling techniques. This application functions as an integrated communication platform, in which the choice of communication channels for interactions between users is mediated by context-aware agents. Each agent manages and records a history of the interactions of a given user over a variety of communication channels, including telephone, email, text messaging and videoconferencing. Sequences of related interactions are organised into threads, termed dialogues.

When a user requires an interaction with another person, the communication agent is invoked. The agent, in cooperation with the agents of the other participants, consults the contexts and preferences of each party in order to recommend an appropriate channel. The main parameters in this choice, which we implemented using the branching model, are user activities, priority of interactions, relationships between participants, and available communication devices.

In designing the application, we conducted an informal study to investigate patterns of interpersonal communication and the factors that users cited for their choices of communication channel. Based on the results of the study, we constructed a context model using CML and a list of situations that covered the main classes of context that we believed users would need to specify their preferences. Next, we defined a set of default preferences for the application that users could use without modification if desired. These steps were carried out in an iterative fashion; that is, when we encountered preferences that required new types of context information, or situations that we could not express in terms of the fact-based model, we extended the context models to accommodate these. They were also driven by practical considerations: for example, when modelling the context with CML it was necessary to consider whether the modelled information could be gathered from users or sensors with the required level of quality.

Next, we mapped the context model manually into the context management system, generated a set of sample context data for simulation purposes, and implemented the communication application using the branching toolkit.

On the whole, the results of the case study were very positive. We found both CML and the situation abstraction nat-

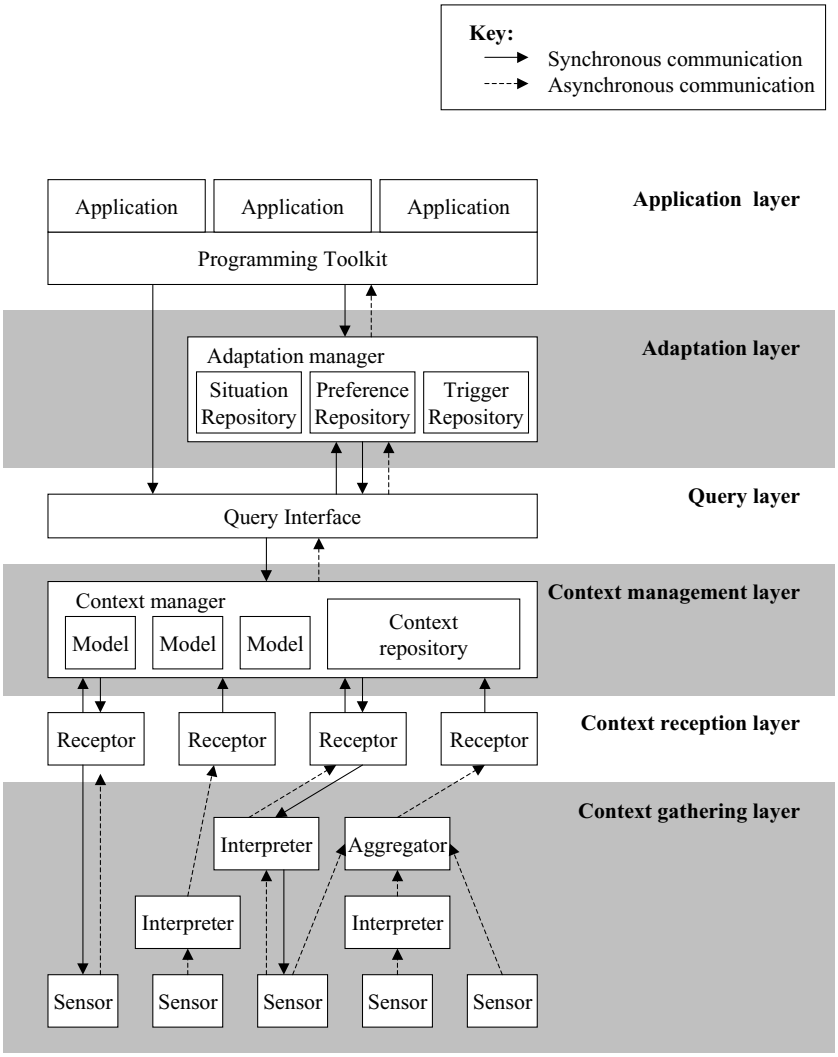


Figure 6. Architecture.

ural and easy to use, and our experience of mapping the CML model to a relational database was a painless one. However, we encountered some unexpected results with the preference model in relation to preferences that assigned frequent indifferent scores. This occurred when the preference scopes incompletely covered the set of possible contexts - for example, when users specified preferences only in relation to urgent interactions. The result was that the agents would sometimes fail to suggest any suitable communication channel. Fortunately, there are two simple solutions to this problem: to augment the preferences to provide complete coverage of the possible contexts, or to program the application to provide better handling of indifferent scores. We adopted the first solution as it did not require changes to the source code; however, when developing future applications we intend to design them to follow the second solution, as it is more robust. Our experiences with the programming toolkit were very positive; however, we do plan to extend the toolkit in the future to provide improved handling of the indifferent and veto scores.

Overall, the case study showed that our conceptual framework and software architecture are extremely successful in terms of our original goal of facilitating the development of context-aware applications that are flexible, adaptable and autonomous. The use of the multi-layered context modelling techniques and the branching model mean that we are able to easily evolve the underlying context model without changing the source code, and both the developer and the end users can easily adapt and fine-tune the choice of communication channels simply by editing the preferences. These features facilitate the types of experimentation that are necessary to gain a better understanding of the most compelling uses of context information and to explore the usability problems de-

scribed in Section 1. Similarly, the use of customisable preferences means that a considerable degree of autonomy can be achieved by the communication agent without removing control from the user.

7. Conclusions and future work

We set out to develop conceptual models and a supporting software infrastructure capable of facilitating a variety of software engineering tasks involved in the development of context-aware software. While we believe that we were successful in this goal, we also argue that there remains considerable scope for future work in this area. In particular, there is a need for a better understanding of the software life-cycle associated with context-aware software as a whole, and for further design tools that address challenges unique to context-aware applications. Work is just now beginning in the latter area; for example, Gray and Salber recently proposed a design framework that supports an informal exploration of the types and characteristics of *sensed* context information required by an application, focusing on quality of service [12]. Further tools and methodologies are needed to address additional problems such as the design of suitable privacy policies (and application functionality that conforms to these), in order to protect context information from possible abuses, and of user interfaces that address known usability challenges, such as the balance of application autonomy to user control.

References

[1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, 1997.

[2] P. J. Brown. The stick-e document: a framework for creating context-aware applications. In *Electronic Publishing*, pages 259–272, Palo Alto, 1996.

[3] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, October 1997.

[4] P. J. Brown and G. J. F. Jones. Context-aware retrieval: exploring a new environment for information retrieval and information filtering. *Personal and Ubiquitous Computing*, 5(4):253–263, December 2001.

[5] H. E. Byun and K. Cheverst. Exploiting user models and context-awareness to support personal daily activities. In *UM2001 Workshop on User Modeling for Context-Aware Applications*, Sonthofen, July 2001.

[6] G. Chen and D. Kotz. Context aggregation and dissemination in ubiquitous computing systems. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, Callicoon, June 2002.

[7] K. Cheverst, N. Davies, K. Mitchell, and C. Efstratiou. Using context as a crystal ball: Rewards and pitfalls. *Personal and Ubiquitous Computing*, 5(1):8–11, February 2001.

[8] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a context-aware tourist guide: the GUIDE project. In *6th International Conference on Mobile Computing and Networking (MOBICOM)*, pages 20–31, Boston, August 2000.

[9] J. Coutaz and G. Rey. Recovering foundations for a theory of contextors. In *4th International Conference on Computer-Aided Design of User Interfaces (CADUI)*, Valenciennes, May 2002.

[10] A. K. Dey and G. D. Abowd. CybreMinder: A context-aware system for supporting reminders. In *2nd International Symposium on Handheld and Ubiquitous Computing*, volume 1927 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2000.

[11] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2-4):97–166, 2001.

[12] P. Gray and D. Salber. Modelling and using sensed context information in the design of interactive applications. In *8th IFIP International Conference on Engineering for Human-Computer Interaction*, volume 2254 of *Lecture Notes in Computer Science*, pages 317–336. Springer, 2001.

[13] T. A. Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Francisco, 2001.

[14] S. Helal, B. Winkler, C. Lee, Y. Kaddourah, L. Ran, C. Giraldo, and W. Mann. Enabling location-aware pervasive computing applications for the elderly. In *1st IEEE Conference on Pervasive Computing and Communications (PerCom)*, Fort Worth, March 2003.

[15] K. Henricksen. *A framework for context-aware pervasive computing applications*. PhD thesis, School of Information Technology and Electrical Engineering, The University of Queensland, Submitted September 2003.

[16] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In *1st International Conference on Pervasive Computing (Pervasive)*, volume 2414 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 2002.

[17] K. Henricksen, J. Indulska, and A. Rakotonirainy. Generating context management infrastructure from context models. In *4th International Conference on Mobile Data Management (MDM) - Industrial Track*, Melbourne, January 2003.

[18] A. Jameson. Modeling both the context and the user. *Personal and Ubiquitous Computing*, 5(1):29–33, February 2001.

[19] H. Lei, D. M. Sow, J. S. Davis, G. Banavar, and M. R. Ebling. The design and applications of a context service. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):45–55, October 2002.

[20] M. Nilsson, J. Hjelm, and H. Ohto. Composite capabilities/preference profiles: Requirements and architecture. W3C Working Draft, 21 July 2000.

[21] V. Stanford. Using pervasive computing to deliver elder care. *IEEE Pervasive Computing*, 1(1):10–13, January-March 2002.