

Modelling Weiser’s “Sal” Scenario with CML

Jadwiga Indulska and Ricky Robinson

The University of Queensland
School of Information Technology and Electrical Engineering
St Lucia, Queensland, Australia

NICTA
Queensland Research Laboratory
St Lucia, Queensland, Australia

Email: jaga@itee.uq.edu.au, ricky.robinson@nicta.com.au

Abstract—Context modelling approaches differ in what concepts can be captured by application designers, in the expressive power of the context models, in the support they can provide for reasoning about context, and in the way they facilitate software engineering of context-aware applications. In this paper we respond to the challenge of CoMoRea’09 and show how the Context Modelling Language (CML) and the concepts associated with CML can be used to engineer applications that exhibit behaviour described in Mark Weiser’s “Sal” scenario. The paper presents the CML based models for applications from the “Sal” scenario and evaluates the approach based on the criteria defined for the challenge including knowledge and software engineering effort, flexibility, usability, and support for uncertain information.

I. INTRODUCTION

The goals of a context modelling formalism include support for software engineering of context-aware applications, reuse and sharing of context information, and support for application maintainability and evolvability. The current context modelling approaches differ in what concepts can be captured by application designers, in the expressive power of the context models, and in the support they can provide for reasoning about context. In this paper we respond to the challenge of CoMoRea’09 and show how the Context Modelling Language (CML) and the associated situation logic [1], [2] and preference abstraction [3] can be used to engineer applications that exhibit behaviour described in Mark Weiser’s “Sal” scenario. The emphasis of the paper is on CML’s ability to specify a wide range of context types and to address timeliness (histories) and imperfection (variable quality) of context, on levels of abstraction in context modelling (context facts and situations built from facts) and on reasoning about situations. We also show that the role of preferences is two fold: they are a means to introduce flexibility and personalisation into context-aware applications but they also allow (together with the context model) an externalisation of context reasoning from the context-aware application into middleware, therefore substantially reducing the software engineering effort and increasing maintainability and evolvability of context-aware applications.

As in the CML approach each context aware application has its own context model we analyse Mark Weiser’s “Sal” scenario to define context-aware applications that are contained in this scenario and describe what context facts, situations and preferences are used in these applications. This is followed by a detailed description of context, situation and preference modelling for three selected applications as the paper page limit does not allow such a detailed specification for all the applications. We also evaluate this modelling approach using the criteria defined in the CoMoRea’09 challenge.

The structure of the paper is as follows. Section II describes the context-aware applications in the “Sal” scenario and provides a high level description of the context types, situations and preferences that the applications can use. Section III provides a brief introduction to our approach to context modelling and shows the architecture of the middleware that supports the models. Section IV presents context fact types, situations, preferences and triggered adaptations for three selected applications. This is followed in Section V by an evaluation of the CML approach based on the criteria introduced in the challenge. Section VI concludes the paper.

II. IDENTIFYING THE CONTEXT-AWARE APPLICATIONS IN THE “SAL” SCENARIO

The “Sal” scenario includes three different environments (at home in the morning, travel to work, and at work) and many context-aware adaptations in these environments. While technically the whole scenario may be considered as one context-aware application, dividing the scenario into several applications increases the flexibility of the solution; each application can be used by many users and the whole “morning at home - travel to work - at work” scenario can be tailored differently for different users. We identified the following context-aware applications:

- “Alarm clock”. The context used in this application is Sal’s state of consciousness sensed by sensors. The context change from “asleep” to “waking” triggers a question from the application whether Sal would like coffee - the application starts brewing coffee if “yes” is the answer.

- “Electronic neighbour trails”. This application uses two types of context information - who Sal’s neighbours are and the histories of neighbours’ locations.
- “Virtual window to children’s room”. The application uses histories of the children’s locations.
- “Smart newspaper”. This application sends a request to a newspaper to send a particular quote to Sal’s office. The context information used is the newspaper’s name, date, section, page, and quote indicated by Sal’s pen. Some of Sal’s preferences can also be used to select a device/place the quote should appear in the office.
- “Foreview mirror”. The application is able to provide/visualise information about traffic ahead and locations of interest along the path of travel. In addition, it can help the driver to locate a free parking spot. The application uses traffic reports, information about shop locations and Sal’s car location as context information to provide the information in the foreview mirror. It also uses Sal’s preferences about types of shops to show and preferences on how the information should be visualised in the foreview mirror. When the car approaches a car park, the foreview mirror displays information about the nearest parking spot and how to navigate to it. The context information used is the sensed appearance of the car at the car park entrance and information about car park occupancy.
- “Preparation for work”. When an employee enters the company building in the morning the employee’s machine starts the login process. The situation that an employee entered the building can be defined/recognised based on various context information. It could, for example, be based on face recognition.
- “Weather report on the window”. The application displays the current weather report on the window. It could use some preferences, e.g. how often the information should be updated.
- “Activity indicator”. The application monitors meetings in the company’s various offices and shows the meetings in the activity indicator window. The application may use a variety of context information to recognise a meeting (e.g. co-location of co-workers in meeting spaces and calendar entries).
- “Fresh coffee notification”. Whenever fresh coffee is available the applications sends a notification to “the telltale by the door”.
- “Virtual office”. The application allows collaboration without physical co-location. This application is able to recognise several situations like co-location of people and devices and gesture events. The context used by the application is mostly gesture based, i.e. gestures are sensed and interpreted to recognise particular situations.

The scenario also uses smart objects, i.e., a manual which helps users to find it - when a code is pressed in a garage door opener the instruction manual starts beeping.

In the following section, we describe in detail the context facts, situations, preferences and triggered adaptations for three selected applications: “alarm clock”, “electronic neighbour trails” and “virtual office”.

III. OUR APPROACH TO CONTEXT-AWARENESS

In this section we describe our context modelling approach, and the architecture for the platform that uses the models created by our modelling technique.

The Context Modelling Language [2], or CML as it is known, derives from a conceptual fact-based information modelling formalism called ORM (Object Role Modelling) [4], which has an intuitive graphical representation. An ORM model consists of a set of object types, depicted as ellipses (or rounded rectangles in the newer notation), which represent the salient “things” in the real world domain that are relevant to the application under development. In addition, ORM captures n-ary relationships, called fact types, between these object types, depicted as sequences of “role boxes”, such that each role in the sequence is “played” by an attached object type. For example, we may have an object type called “Car” and another called “Colour”. We may introduce a binary fact type between these two object types that indicates that a “Car” has a “Colour”. In addition to these basic elements, a range of constraints may be introduced to the model, which indicate, for instance, whether a “Car” *must* have a “Colour” or whether a “Car” may have more than one “Colour”, and so on. Importantly, ORM has a formal semantics, which means that satisfiability checking of constraints is possible. CML extends ORM by introducing modelling elements that are key to context-aware applications. These include: fact type classification, enabling differentiated handling of data arising from heterogeneous sources (sensors, user profiles, static information and derived information); fact quality, which attaches fact type specific quality metadata to facts; temporality that denotes the time period during which the fact was relevant (the introduction of temporal fact types alters the formal semantics of the uniqueness constraints inherited from ORM, but a discussion of this is outside the scope of this paper); and fact type dependence, which indicates some (unspecified) dependency of one fact type upon another.

Context models expressed using CML form the basis of our approach. These models are inserted into our middleware for context-aware computing (see Figure 1), whereby the appropriate runtime data structures are created from the models. In our implementation, CML models are mapped to relational database schema. At this point, “facts” that conform to the various fact types specified in the models can be collected, inserted and managed by the context management system (see the shared repository of context facts in Figure 1). These facts may be collected from sensors, entered by the user, provided by an administrator at deployment time or derived from one or more of these former classes. For sensed information, we have developed an automated, standards-based solution for provisioning data from heterogeneous context sources that is

robust to mobility and sensor failure [5]. This data provisioning can dynamically find suitable sensors and compose pre-processing components for processing the raw sensor data into the abstract context facts required by the context-aware application. Context-aware applications adapt to changes in situations not to changes in single sensor values. Therefore to support ease of programming, CML fact types may be combined into higher level abstractions known as situations. In our approach to situation modelling these situations are expressed in a modified first order logic. Situations can then be used by triggers, that execute when a particular situation arises, or they may be used by preferences, which combine to yield one or more execution choices when a decision needs to be made by the context-aware application. These preferences remove the need for large sequences of *if-then-else* type constructs embedded directly in the code, and may be modified by the user at any time, making our approach very flexible. The situation models, preferences and triggers are managed by the Adaptation Manager as illustrated in Figure 1.

IV. CONTEXT MODELS FOR SELECTED APPLICATIONS

In this section we present the context models and corresponding relations, situations, triggers and preferences for the three selected applications.

The alarm clock application has a simple context model that captures a person’s state of consciousness (Figure 2). In addition, we define a situation that evaluates to true when a person is waking up (Figure 3).

To implement the required behaviour, we define a trigger whose body is executed when Sal is in the waking up situation (Figure 4). The body of the trigger defines a one time trigger that invokes the *makeCoffee()* procedure if a positive response is received to the question “Coffee?”. For the purposes of showing one of the features of our trigger system, the body of the trigger will only be invoked if Sal wakes up between 6am and 9am. The *makeCoffee()* operation is provided by the programmer. As can be seen from this simple example, our approach allows the context dependent aspects of an application to be quickly and easily captured and defined, enabling the developer to identify and isolate the non-context dependent parts in custom procedures (in this case, the *makeCoffee()* procedure).

The neighbour trails application is a very simple application that visualises the locations of one’s neighbours. The CML model captures people’s locations and the “neighbour” relationship (Figure 5). The notable aspect of this model is the use of “context histories”. These histories capture the movements of people over time.

The most involved application is the virtual office. A key aspect of this application is the way users gesture with “tabs” towards other devices. These gestures are mapped to actions via triggers. Our CML model captures the low-level context information such as the locations of people and devices, which device a tab is currently pointing at (if any) and which devices (screens) a person appears on (Figure 6). This application

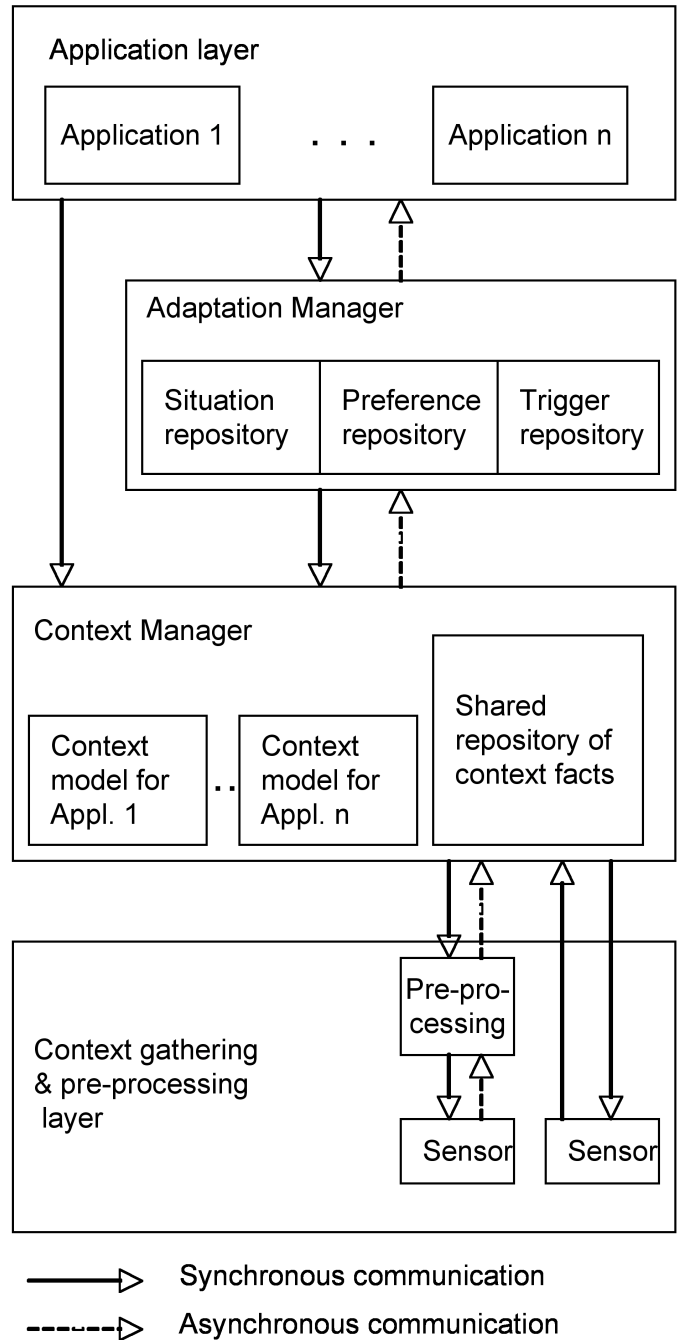


Fig. 1: The architecture of our middleware for context-aware computing

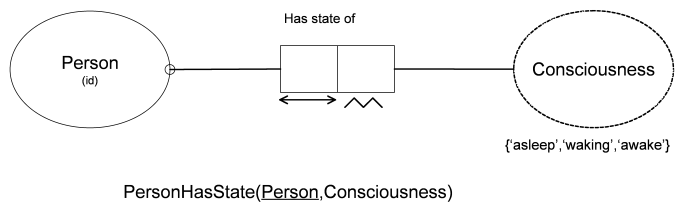


Fig. 2: The CML model and corresponding relation for the alarm clock application.

WakingUp(person) :

$$\exists \text{state} \bullet \text{PersonHasState}[\text{person}, \text{state}] \bullet$$

$$\text{state} = \text{"waking"}$$

Fig. 3: The situation required by the alarm clock application.

```

upon EnterTrue(WakingUp("Sal"))
when true
do upon EnterTrue(
  AffirmativeResponse(response))
  when true
  do makeCoffee()
  once

  ask "Coffee?"
  from 6am until 9am

```

Fig. 4: The trigger required for the alarm clock application. It sets up a one-time trigger and asks Sal if she wants coffee. The one-time trigger waits for Sal’s response and takes action if her response is in the affirmative.

also showcases CML’s ability to capture the quality of context information.

We define multiple situations that capture high-level contexts (Figure 7). The first situation builds on the *PointingAt* and *HeldBy* relations. It tells us when a gesture event occurs between a tab and another device. A condition of a gesture is that the action must be made by a person (to rule out cases where a tab happens to pointing at a device while it is lying on a desk or in someone’s pocket). The next situation is used to establish whether a person wishes to initiate a virtual office session with another person. We assume that the act

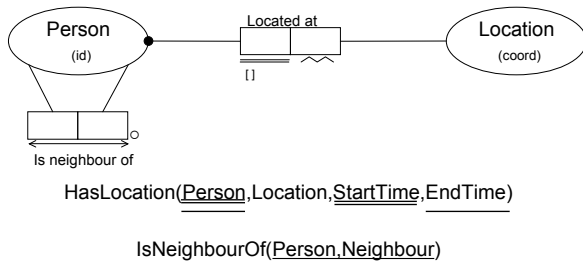


Fig. 5: The CML model and corresponding relations for the neighbour trails application

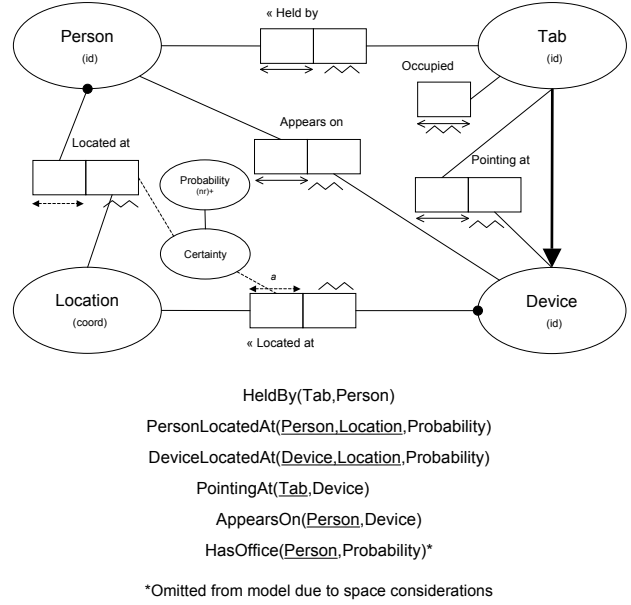


Fig. 6: The CML model and corresponding relations for the virtual office application.

of gesturing to a screen on which a person appears sets up the users’ virtual office environment (i.e., it makes available relevant files and folders, and in Sal’s case, makes sure that the tabs on her desk are showing miniature versions of Joe’s tabs and pads). The other situations are used to decide under what conditions to set up the virtual office environment, and which tab(s) to route alerts to.

We also define two triggers (Figure 8). The first determines when to set up the virtual office environment for two users, and the second determines when the contents of a screen should be copied to another screen. The latter is used to facilitate the scenario whereby Sal wishes to circle some words on a document written by Joe.

In addition, we define a procedure *routeAlert()* that makes use of situation-based branching and the preferences in Figure 9. The *routeAlert()* procedure is invoked when a person attempts to initiate communication with another person. In the case of the “Sal” scenario, Joe contacts Sal to discuss a document. The purpose of the *routeAlert()* procedure is to choose the device on which to display an alert, and it does this by utilising preferences. Since tabs are intended to be “active Post-It notes”, they are numerous, cheap and fungible. Therefore, we cannot assume that all alerts are sent to a single “personal” tab. Instead, we must route incoming calls to a contextually appropriate tab.

These examples illustrate the relative ease with which context-aware applications can be created with CML, our situation logic, triggers and preferences. While we have not, of course, implemented these particular applications, we have used our approach to implement several context-aware appli-

```

Gesturing(tab, device) :
  ∃person • HeldBy[tab, person]•
  PointingAt[tab, device]

GestureToPerson(p1, p2) :
  ∃tab • HeldBy[tab, p1]•
  (∃device • PointingAt[tab, device]•
  AppearsOn[p2, device])

InPhysicalOffice(p) :
  ∃loc • LocatedAt[p, loc]•
  HasOffice[p, loc]

LocatedWithin(p, d, delta) :
  ∃pLoc, pProb • PersonLocatedAt[p, pLoc, pProb]•
  (∃dLoc, dProb • DeviceLocatedAt[d, dLoc, dProb]•
  distance(pLoc, dLoc) < delta ∧ dProb > 0.8) ∧
  pProb > 0.8

InUse(tab) :
  Occupied[tab]

```

Fig. 7: The situations required for the office application.

```

upon EnterTrue(GestureToPerson("Sal", "Joe"))
when InPhysicalOffice("Sal") ∧
  InPhysicalOffice("Joe")
do setupEnvironment()
always

upon EnterTrue(Gesturing(tab, device))
when true
do copyRemoteScreenTo(device)
always

```

Fig. 8: The triggers for the office application.

```

DefaultAlertPrefs = {
  p1 = WHEN NOT InUse(tab)
  RATE 1
};

SalAlertPrefs = {
  p2 = WHEN LocatedWithin(person, tab, 2.0)
  RATE 1,
  p3 = WHEN NOT LocatedWithin(person, tab, 2.0)
  AND LocatedWithin(person, tab, 5.0)
  RATE 0.5,
  p4 = WHEN NOT LocatedWithin(person, tab, 5.0)
  RATE 0.1
};

AllAlertPrefs =
  WHEN TRUE
  RATE average(<DefaultAlertPrefs,
  SalAlertPrefs>)

```

Fig. 9: The preferences used by the routeAlert() procedure.

cations in the past.

V. EVALUATION

The previous section illustrated the CML and associated models for selected applications. In this section we provide a brief evaluation of our approach based on the criteria defined in the CoMoRea'09 challenge. This evaluation takes into account the features of (i) our modelling approach, i.e. CML modelling context facts, situation abstraction, and preference modelling, (ii) our context and preference management systems, and (iii) the programming toolkit which is an interface to the context and preference management systems and implements adaptation triggers for situations and also branching (queries for selecting entities based on context and preferences).

Knowledge engineering effort. After defining the applications in the “Sal” scenario and selecting three applications indicated in Section II, we needed a few hours to define and draw the CML context model, to define situations and to define triggers that invoke adaptations to particular situations.

Flexibility. Any of the shown models, i.e. the CML context model, situations, and preference models can be easily extended or modified. CML allows addition/modification of (i) context facts, (ii) quality of information associated with context facts, and (ii) timelines (including histories) of context information. The situation model is based on first order logic and the situation expressions can be modified within the scope of the logic. It is easy to modify preferences as these are decoupled from the program logic, and can be added to, updated or removed at any time. The preferences are combined together and evaluated at runtime when situations are evaluated. It should be noted that each application has its own model (a model of context fact types used by the application) and the model and context facts gathered from various context sources

based on this model are maintained and manipulated by the context manager. The situations, preferences and triggers are maintained and acted upon by separate managers. Therefore the change to any of the models or triggers associated with the application is conveyed to the appropriate managing entity showing the intended change in the behaviour of a context-aware application.

Usability from an application developer's perspective. The CML modelling provides the designers with an easy to understand graphical model of context information used by a particular context-aware application and therefore allows easy modification/tuning of the model. The supporting middleware (the context and adaptation managers) are able to reveal to the designer the situation evaluation (including the context facts and preferences used in the evaluation and also the adaptation logic included in triggers) supporting the designer in testing and in usability study of the developed application and therefore supporting its modification/tuning [6].

Software engineering effort. The CML fact type models semi-automatically map to relations as shown in Figures 2, 5 and 6, and this allows an easy addition of the application context model to the context management system. As shown in Section IV, the whole software engineering effort includes defining all the fact types (relations) for an application (around 10 fact types in total for the three selected applications), defining situations from the fact types, defining preferences that should be evaluated for particular situations, and defining triggers that show which adaptation should be applied to particular situations.

Expressiveness. The CML based approach can define a wide range of context fact types (static, sensed, and derived). As the fact types used in the CML modelling are an extension of the Object Role Modelling (ORM) [4] approach they can easily model any real world facts. The expressiveness of situations is defined by the expressiveness of the first order logic. Users define simple preferences but the preferences are combined at run time based on the preference scope allowing evaluation of complex preferences.

Uncertainty; user feedback on context quality. The CML model can model quality of context facts [2], [7] and can provide this information to users at runtime if requested [6]. If a situation which led to a particular adaptation includes several context facts, the quality of each of these facts can be provided as the feedback, however the system is not able to calculate the quality of the situation decisions.

Consistency checking. Although CML itself affords automated consistency checking, no automated consistency checking tools for CML have been developed at this point in time. However, several ORM tools, including Microsoft's, prevent inconsistent constraints from being applied and perform consistency checking prior to mapping the ORM model to a relational schema (or other implementation). The additional constraint types and modelling concepts added to ORM by CML (lifetime/snapshot constraints, alternative fact types, fact dependencies, etc.) can also be consistency checked.

VI. CONCLUSION

Our approach to modelling and engineering context-aware applications with CML and the associated situation logic, trigger rules and preference language has been shown to be viable and intuitive through a number of proof-of-concept applications. In this paper we show how that approach can be applied to what is perhaps the most famous motivating scenario in ubiquitous computing. Our approach places emphasis on eliciting application requirements and translating those quickly and accurately to conceptual models, situations, triggers and preferences, thus greatly simplifying the development task and reducing the required programming effort. We acknowledge a current lack of tool support; however, we are rectifying this situation by developing tools for creating graphical CML models, and for mapping these models automatically to relational schemas and other "runtime" formats. In addition, we're developing tools that enable developers and users to understand why their applications adapt in certain ways. These latter tools simplify the tuning of application adaptations.

ACKNOWLEDGMENT

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program; and the Queensland Government.

REFERENCES

- [1] K. Henricksen and J. Indulska, "A software engineering framework for context-aware pervasive computing," in *Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*. IEEE Computer Society, 2004, p. 77.
- [2] —, "Developing context-aware pervasive computing applications: Models and approach," *Pervasive and Mobile Computing*, vol. 2, no. 1, pp. 37–64, 2006.
- [3] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Using context and preferences to implement self-adapting pervasive computing applications," *Journal of Software Practice and Experience, Special Issue on Auto-Adaptive and Reconfigurable Systems*, vol. 36, pp. 1307–1330, 2006.
- [4] T. A. Halpin, *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. San Francisco: Morgan Kaufman, 2001.
- [5] P. Hu, J. Indulska, and R. Robinson, "An autonomic context management system for pervasive computing," in *Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 213–223.
- [6] B. Hardian, J. Indulska, and K. Henricksen, "Exposing contextual information for balancing software autonomy and user control in context-aware systems," in *Proc. of the Pervasive 2008 Workshop on Context-Aware Pervasive Communities: Infrastructures, Services and Applications*, 2008.
- [7] K. Henricksen and J. Indulska, "Modelling and using imperfect context information," in *1st Workshop on Context Modeling and Reasoning (Co-MoRe), PerCom'04 Workshop Proceedings*. IEEE Computer Society, March 2004, pp. 33–37.