Applying a Disciplined Approach to the Development of a Context-Aware Communication Application*

Ted McFadden[1] Karen Henricksen[1] Jadwiga Indulska[2] Peter Mascaro[1]

[1]CRC for Enterprise Distributed Systems Technology (DSTC)
Email: {mcfadden, kmh, mascaro}@dstc.edu.au
[2]School of Information Technology and Electrical Engineering
The University of Queensland, St Lucia QLD 4072 Australia
Email: {jaga}@itee.uq.edu.au

Abstract

Pervasive computing applications must be engineered to provide unprecedented levels of flexibility in order to reconfigure and adapt in response to changes in computing resources and user requirements. To meet these challenges, appropriate software engineering abstractions and infrastructure are required as a platform on which to build adaptive applications. In this paper, we demonstrate the use of a disciplined, model-based approach to engineer a context-aware, Session Initiation Protocol (SIP) based communication application. This disciplined approach builds on our previously developed conceptual models and infrastructural components, which enable the description, acquisition, management and exploitation of arbitrary types of context and user preference information to enable adaptation to context changes.

1. Introduction

In current computing environments characterised by mobility of users and heterogeneity of computing devices and networks, the context of computing applications (e.g., computing power, type of available networks, Quality of Service of communication and user preferences) may change. In order to adapt to these changes, applications need to be context-aware. That is, they need to understand the context and be able to adjust their behaviour to context changes.

To address the continuing challenges of building flexible and evolvable context-aware applications we have developed innovative approaches to modelling context and context-dependent user preferences and policies, as well as new software engineering abstractions and supporting infrastructure [8, 7, 9, 6]. In this paper, we illustrate how these models are applied to an example context-aware communication application. Communication has been widely explored as a compelling application domain for context-awareness. A variety of application types have been developed, including:

- text-based messaging and chat programs that provide users with information about other users' presence, location, mood and other relevant context [14, 15]; and
- communications applications that exploit a variety of information such as users' locations, activities, available devices, and channel types to assist in the routing of calls so that disruption is minimised and missed calls are avoided [17, 19, 15].

It is the second category of application that we use to illustrate our disciplined approach to context-aware application development in this paper. We consider a self-adapting communication application that exploits context and preference information to allow users to communicate seamlessly with one another. The application coordinates a set of devices and communication channels for each person such that configuration requirements are minimal and users need only specify the individual(s) they wish to contact, rather than the means by which this is done. Interactions between users can involve an arbitrary number of devices and a variety of communication modes, including videoconferencing, telephone calls and text messaging. This application is based on the Session Initiation Protocol (SIP) [16], the emerging standard for session and call control.

The application exploits a rich set of contextual information in order to self-configure and adapt, including the location of people and devices, activity types, device network,

*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 Education, Science, and Training).

status, and power. In addition, the application relies on system policies and user preference information to evaluate the suitability of available communication channels for the current context of use. Policies and preferences can be modified on-the-fly, in order to support highly flexible behaviour and accommodate an evolving set of system resources and user requirements.

This context-aware communication application builds on our experiences with developing an earlier communication prototype [8]. The new application described in this paper is considerably more powerful, as it removes the initial setup tasks, and is also capable of dynamically adapting communication channels mid-session. Additionally, the new application incorporates a refined context model (building on new sensor types), and an improved version of our context and preference management infrastructure. The case study we present in this paper underscores the importance of an expressive and flexible set of modelling and software engineering abstractions to support change.

The structure of the paper is as follows. Section 2 discusses context-aware system models and briefly reviews our previously developed conceptual models for context and preference description and management, and programming of context-aware applications. This description is supported by brief examples showing how the models are used in our context-aware communication application. Next, Section 3 provides an overview of the software infrastructure we implemented to support our conceptual models, while Section 4 shows the architecture of our context-aware communication application and its interactions with the infrastructure. Finally, Section 5 concludes the paper with a discussion of lessons learned in the development of the communication application and topics for future work.

2. Generic models for context-aware systems

2.1. Context models

Developing context-aware applications continues to present software engineering challenges. Well known solutions that assist with acquiring and processing context information from sensors have been proposed by Dey et al. [4], Schmidt et al. [17] and Chen and Kotz [1]. In addition, a variety of context management systems that maintain repositories of context information and provide sophisticated query facilities to context-aware applications have also recently appeared [12, 11]. These solutions focus on removing complex functionality from the applications and placing it within shared infrastructure, but do not place much emphasis on providing context modelling techniques that are natural to use for the application developer. To address this problem, we have developed context modelling techniques that provide two levels of abstraction: facts and situations. The latter allow programmers to define high-level classes of context that are appropriate for use in describing context-aware functionality. Situations are considerably closer than collections of facts to the way humans conceptualise context, and also support composition and reuse so that complex situations can be easily formed incrementally by the programmer.

Modelling context facts. Context facts suitable for management in context repositories are modelled using our Context Modelling Language (CML) [7]. Figure 1 shows how CML's generic modelling constructs have been used to model the context information required by our context-aware communication application. The model describes key entities of the system including users, devices, absolute and relative locations, SIP contact URIs, and device characteristics such as power levels and supported communication modes. The fact types, which define relationships between these entities, are denoted by sequences of role boxes, and can be classified as static, profiled, sensed, or derived. The context model is instantiated as a set of facts stored in a context database, and the fact types are used to define higher-order situations as shown in the next subsection.

Modelling situations. We model abstract classes of context as situations. Situations are defined by constraints on context facts expressed using a variant of predicate logic, and can be easily combined, allowing complex situations to be incrementally formed by the programmer. We showed in [8] that our situation abstraction is conceptually similar to the model proposed by Dey and Abowd [3], but is considerably more expressive.

Some example situations for our context-aware communication application are shown below.

- ```
  PowerLow(device):
      exists power
      . HasPower[device, power]
      . power <  0.1
  ```
- ```
  CommunicationModeAvailable(person, mode):
      exists device
      . PersonNear[person, device]
      . SupportsMode[device, mode] and
        HasStatus[device, "available"])
  ```

The first example represents the condition in which a device has low battery power, while the second is a situation that indicates whether a communication mode (voice, text, video, etc.) is currently accessible to a person. According to the situation, the mode is available if there is a device near the person that supports the mode and is available for use.

2.2. Preference modelling and management

There is growing recognition of usability challenges associated with context-aware software related largely to a

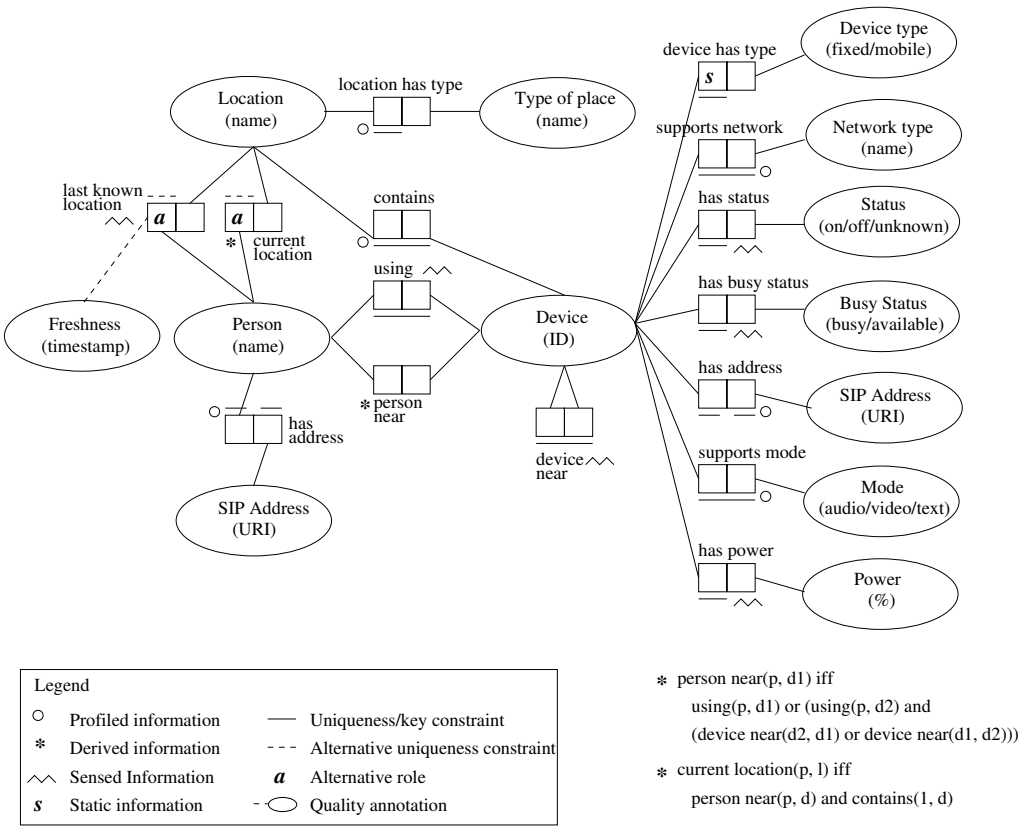

Figure 1. Context fact model for the communication application

lack of user control over applications' actions. After evaluating existing approaches to preference modelling and elicitation such as [2, 5], we developed a novel preference modelling technique based on the situation abstraction [8]. The preference model supports the ranking of a set of choices 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 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 `prohibit`, `indifferent`, `oblige` or `undefined`.

Below is a simple user preference set from our context-aware communication application that asserts a user's preference for selecting a fixed-line phone over a mobile device:

- `UserPrefs`:
  - `FixedPref = when Fixed(dev) rate 1`
  - `MobilePref = when Mobile(dev) rate 0.5`

The above example is very simple, but user preferences can also be considerably more sophisticated, incorporating arbitrary types of context information.

In our application, individual user preferences are used in conjunction with system preferences, a sample selection of which is illustrated below:

- `SysPrefs`:
  - `NearPref = when NearDevice(user,dev) rate 1`
  - `AvailablePref = when not Available(dev) rate prohibit`
  - `PowerPref = when PowerLow(dev) rate 0.1`
- `CompositePrefs`:
  - `AvgSysPrefs = when true rate average(SysPrefs)`
  - `AvgUserPrefs = when true rate average(UserPrefs)`
  - `FinalEval = when true rate average(AvgSysPrefs,AvgUserPrefs)`

The first system preference set, `SysPrefs`, defines standard preference rules that are applied across our communication system for device selection, while

`CompositePrefs` provides additional composite preferences that describe how the ratings produced by the individual preferences in the `SysPrefs` set and other preference sets are combined to obtain aggregate ratings. To illustrate, using `AvgSysPrefs` as an example, this composite preference will always prohibit the use of a device that is not available and assign a rating of 1 to a device that is near the user, unless the device has low power, in which case the rating will be 0.55 [(0.1+1)/2]. The system preferences are augmented with a dynamically defined and evolvable set of user-specific preferences, `UserPrefs`. The `FinalEval` preference combines the aggregate score produced by the `AvgUserPrefs` preference with the aggregate `AvgSysPrefs` score by averaging. The low-level preference format shown in the examples is not directly exposed to users, who manipulate their preferences through a user-friendly graphical interface.

2.3. Programming models

In addition to developing these context and preference modelling abstractions, we have also created further abstractions and tools to support the software engineering process. Most current approaches to building context-aware applications embed simple decision logic directly into the source code, creating an undesirable degree of coupling between applications and the context models. To reduce the coupling and allow applications to gracefully evolve when the context models are modified to reflect changes in the underlying sensing infrastructure or in user requirements, we created two new models of programming (triggering and branching models) that exploit our situation and preference abstractions. Triggering is already commonly used in context-aware computing. The branching model supports context-dependent choice amongst a set of alternatives, based on context situation and user preferences, via several methods including `rank`, and `selectBest`. These methods form an API and allow context and preferences to be evaluated outside the application logic. The models are described fully in a previous paper [8], and their use for our communication application is demonstrated in section 4.

3. Infrastructure

We integrated the conceptual foundations described in Section 2 into a software infrastructure that provides context and preference management functions and implements toolkit support for the programming models. The infrastructure is organised into the following layers (illustrated in the bottom half of Figure 2.)

- Context gathering layer. This layer acquires context information and processes this information using interpretation and data fusion. Elvin [18], an event notification scheme that employs content-based routing, provides a loose coupling between components of this layer.

- Context and preference management layer. The gathered context information is stored as context facts by the context manager, which is also responsible for maintaining the situation definitions. The preference manager stores and evaluates user and system preference information.

- Context toolkit layer. This layer provides a programmer's toolkit that implements the triggering and branching models, and provides means for querying the context and preference managers.

The context manager is the central and most complex component. Features of the context manager include synchronous query and update, as well as subscription-based, asynchronous notification of context events.

Our current infrastructure represents a substantial refinement of our earlier prototype [8], which incompletely implemented the functionality of the context manager and supported only one of our two programming models.

4. Context-aware communication application

Our application leverages context and preference information to allow for seamless communication between individuals. The application relies on context and preference evaluation for both call initiation and mid-stream call transfer in response to context change. The former makes use of our branching programming model, and the latter both the trigger and branching models. Our application controls communications devices (i.e., instantiates communication sessions between devices) using the SIP protocol. In the following subsections we describe the design of the application, including necessary extensions to the SIP protocol, and the interactions of the application with our infrastructure for pervasive computing.

4.1. Context and preference manager usage

To set up the adaptive communications application, the context manager is initially loaded with the appropriate fact and situation definitions. The context manager is then populated with static and profiled context facts such as the device types (fixed or mobile) and supported networks.

While the application is running, various sensors and software agents generate raw context information that is

processed and presented to the context manager. For example, location and device status updates are fed as corresponding context facts into the context manager. The context manager evaluates whether the update requires any registered context fact or situation subscriptions to receive an event trigger. In our system, the SIP proxy subscribes to situation information concerning active calls.

The preference manager is initially loaded with the default application system preference set. Users of the system may then register their own preference sets. The preference sets for this application rate the suitability of a given device for communication in a specific context. Key context elements used in the preferences include the user location, user-device proximity, device status and mode.

4.2. SIP extension

SIP is a text-based, application-level, signalling protocol allowing for the setup, control, and tear-down of communications sessions between two or more SIP user agents. A user agent can take a number of forms such as a phone, instant messenger program, pager, PDA or PC softphone, or an automated software service. SIP uses URIs, such as 'sip:emma@pace.dstc.edu.au' to identify logical user (or service) contact points independently of specific user agents, network addresses, or communication session endpoints. Communication endpoints may be re-negotiated mid-session.

The standard SIP architecture defines proxies and registrars that aid in routing SIP requests. A SIP proxy is conceptually similar to a HTTP proxy, except that no content information flows over the control channel. A registrar works in conjunction with, or as part of, a proxy to provide call-redirection capabilities.

SIP user agents may contact a registrar (or locator) proxy service and register specific forwarding contact SIP URIs for a logical URI identity that is in a SIP domain of the registrar. SIP proxies make use of this registration information to route SIP requests to the appropriate next hop. For example, a SIP proxy for the SIP domain 'dstc.edu.au' could forward a request directed to 'sip:pace-user1@dstc.edu.au' to 'sip:pace-user1@ipaq57.dstc.edu.au', if the ipaq57 device was currently registered as the contact for pace-user1@dstc.edu.au.

Although SIP registrar and proxy behaviour are standardised, the interface between a SIP registrar and proxy is not. The registrar feature of SIP permits some very useful dynamic call routing to occur, however it places the primary burden for dynamic behaviour on the individual user agents and provides limited flexibility in coordinating the behaviour of multiple user agents.

In our system, the somewhat static SIP process for registration and proxy lookup has been replaced with a more dynamic, context-sensitive approach that also permits mid-stream call redirection (transfer) that need not be initiated by an individual user agent. This is done in a plug-compatible way with SIP, such that our system appears as a standard SIP proxy to other SIP agents, as illustrated in Figure 2. This allows us to use our application with existing SIP user agents and services.

In a standard SIP environment, user agent requests, such as the SIP INVITE used to initiate a call in Figure 2, can be mediated by a proxy. When a proxy receives a request, it may query a locator component (associated with a registrar) to determine if the SIP URI specified in the request should be redirected to a different contact URI. In our system, the proxy issues an asynchronous 'Locate' message for the requested URI to any available system locators (SIP Locator 1-N in Figure 2) to determine a contact URI. These locators do not have static registration tables. Instead, they use the context and preference information, evaluated through the context toolkit (as described in the next subsection), to determine a contact URI, if available. Locators that successfully determine a contact URI asynchronously reply to the proxy with a 'ForwardTo' message. The ForwardTo message elements include the forwarding URI and a rating produced by combining preference scores for the URI. The SIP proxy uses any received ForwardTo SIP URIs and ratings to determine where to route the original request. If no ForwardTo's are received within a specified timeout period, the request is rejected.

The communication between SIP proxies, locator services, and some other components makes use of the Elvin notification service to decouple specific component instances and allow for graceful evolution of the system.

4.3. Context toolkit

For a SIP locator component to process the 'Locate' messages described above, each locator uses the context toolkit to perform the following steps:

1. The Context Manager is queried to discover devices near the user (using the "person near" fact type).

2. The devices are rated against the `FinalEval` preference, which combines all user and system preferences, using the branching model API's `rate` method.

3. If any device gets an `oblige` score, it is selected for use. Otherwise, if at least one device receives a numerical score, the device with the highest score is selected. Otherwise, a device with an indifferent score is selected arbitrarily (or if there are no devices with indifferent scores, the locator concludes that there are no suitable devices and does not respond).

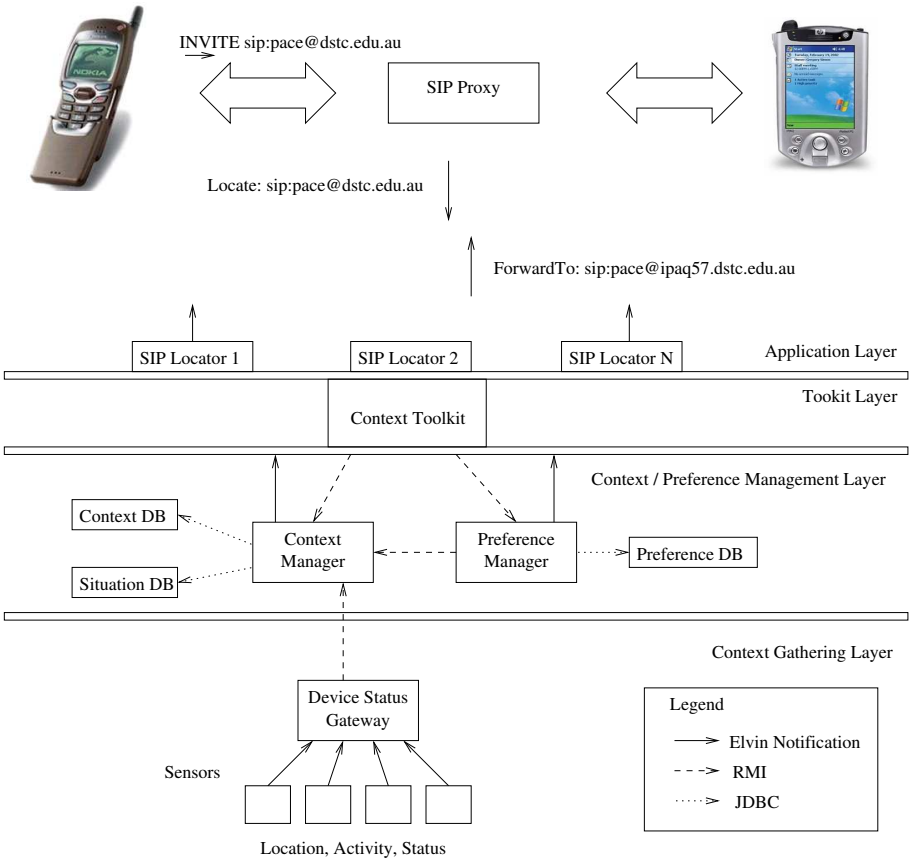

Figure 2. Cooperation between context-aware application and infrastructure

4. The SIP address for the device selected at step 3 is determined from the "has address" fact type which maps devices to SIP addresses.

If the locator finds a suitable device, it sends a 'ForwardTo' message containing the contact URI and the choice rank value. Alternative locator behaviors are possible but are not discussed in this paper.

For the SIP proxy to be able to respond to changes in context affecting currently active calls, the proxy subscribes for triggers (event notifications) with the context manager for any situation changes that could affect device preferences such as a change in location, low power, or on/off status for a device. In ECA (Event, Condition, Action) model terms, the precondition in this case is always true, and the action is to re-evaluate the preferences for the call of concern. If this re-evaluation results in the selection of any new devices, the call is re-routed.

5. Discussion and future work

This paper presented a self-adapting, context-aware communication application implemented using conceptual models and infrastructure that we developed as a basis for building context-aware pervasive systems. The use of a disciplined, model-based approach allowed us to exploit a rich and diverse set of context information. Moreover, it enabled a clean separation between the application and the underlying context and preference information, leading to considerable flexibility as the latter can be easily evolved over time to support changes in user requirements and resources with minimal impact on the application. We have exploited this facility to experiment with a variety of context types and preference sets, and anticipate that we will continue to evolve the context and preferences over time. Our experiences suggest that this type of experimentation and fine-tuning will be essential in all but the simplest context-aware applications. In these applications, the tasks of self-configuration and adaptation must accommodate many different types of context information, balance diverse and often conflicting objectives, and exploit new resources as they become available. The resulting complexity will sometimes lead to unexpected emergent behaviours, which will only be fully explored and overcome by trial-and-error.

In our application, this problem became evident in a phenomenon we refer to as "preference surprise". This oc-

curred when the communication device selected for an interaction did not match the one expected by the user. One cause for this is that users' preference sets typically capture many different requirements, and although our preference model makes it relatively easy to specify each of these in isolation, the outcome of combining them is not always intuitive. Adjustment of the relative weights assigned to the individual preferences is sometimes needed to bring the device selection in line with the user's expectation. Currently, we perform a manual adjustment, but we anticipate that learning algorithms could be used to exploit user feedback (when the user overrides the system's device selection) to automatically evolve the preference set.

As part of our ongoing research, we are continuing to develop additional context-aware applications using our approach, both in communications and other domains. Most recently, we have designed a common context model for a family of applications supporting independent living of the elderly, illustrating the potential of our model-based approach to support reuse of context facts, situation definitions, and user preferences [10]. We are also designing and implementing various extensions to our software infrastructure and programming toolkits to provide privacy support and improved handling of imperfect and incomplete context and preference information.

Finally, the use of a consistent model-based approach to context-aware application development has also allowed us to begin introducing tools that reduce the overall effort in programming context-aware applications by automating aspects of deploying, administering, and programming using our infrastructure. The first tools, built around a common compiler front-end for a text based CML context schema representation, provide generation of SQL context databases, context manipulation libraries in multiple programming languages, and messaging support for content-based notification systems [13].

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] T. Erickson. Some problems with the notion of context-aware computing. *Communications of the ACM*, 45(2):102–104, February 2002.

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

[7] K. Henricksen and J. Indulska. Modelling and using imperfect context information. In *Workshop on Context Modeling and Reasoning (CoMoRea), 2nd IEEE Conference on Pervasive Computing and Communications (PerCom)*, Orlando, March 2004.

[8] K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *2nd IEEE Conference on Pervasive Computing and Communications (PerCom)*, Orlando, March 2004.

[9] 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.

[10] J. Indulska, K. Henricksen, T. McFadden, and P. Mascaro. Towards a common context model for virtual community applications. In *2nd International Conference On Smart homes and health Telematics*, Singapore, September 2004.

[11] G. Judd and P. Steenkiste. Providing contextual information to pervasive computing applications. In *1st IEEE Conference on Pervasive Computing and Communications (PerCom)*, pages 133–142, Fort Worth, March 2003.

[12] 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.

[13] T. McFadden, K. Henricksen, and J. Indulska. Automating context-aware application development. In *Ubicomp'2004 Workshop on Advanced Context Modeling, Reasoning and Management*, Nottingham, September 2004.

[14] A. J. H. Peddemors, M. M. Lankhorst, and J. de Heer. Presence, location, and instant messaging in a context-aware application framework. In *4th International Conference on Mobile Data Management (MDM)*, volume 2574 of *Lecture Notes in Computer Science*, pages 325–330. Springer, 2003.

[15] A. Ranganathan and H. Lei. Context-aware communication. *IEEE Computer*, 36(4):90–92, 2003.

[16] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, and E. Schooler. *Session Initiation Protocol (SIP) RFC 3261*. IETF, A, 2002.

[17] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. In *1st International Symposium on Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 1999.

[18] B. Segal, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with elvin4. In *Proceedings of the AUUG2K Conference*, 2000.

[19] D. Siewiorek, A. Smailagic, J. Furukawa, N. Moraveji, K. Reiger, and J. Shaffer. SenSay: A context-aware mobile phone. Technical report, School of Computer Science, Carnegie Mellon University, 2003.