

Simulation Model for Self-Adaptive Applications in Pervasive Computing

Markus C. Huebscher and Julie A. McCann
Imperial College London

Abstract—In Autonomic Computing, an application needs to be aware of its environment. While the term “environment” is not normally understood as being a physical environment, in Pervasive Computing many applications do actually need to monitor the physical environment in which they are deployed. Monitoring the environment often includes gathering information about the people working or living in this environment. Applications that self-adapt to changes in the monitored environment are known as context-aware. The environment is monitored using sensors, such as temperature, humidity, location sensors, etc., and use some form of logic to abduce a context. As the input of this context logic is environment sensor data, testing these applications usually requires deployment at a physical test location, often in a research laboratory. Our project aims to design a simulation model of contexts as a means to test the context logic of a context-aware application, by allowing sensor data to be produced from a description of contexts, i.e. the location and activities of people in this location, thereby allowing initial testing of a context-aware application without requiring physical deployment.

Index Terms—Pervasive computing, context-awareness, simulation.

I. INTRODUCTION

A FUNDAMENTAL aspect of an autonomic computing system is its reactivity to the environment. This involves monitoring the environment and mapping raw monitored data to high level notions in the adaptation model [1] (see Figure 1(a)). For instance, self-adaptive web servers monitor network traffic and react to changes in bandwidth consumption and latency. However, latency is not measured directly in the system, but might be computed knowing the replication factor of the servers and measuring the communication delay between clients and servers, the arrival rate of client requests and the service time of a request by a server. While there are very different approaches to making web servers autonomic, a common goal is creating a system that is extremely responsive to changes in network behaviour. Thus, any autonomic system’s approach requires testing in a simulated environment, to determine how well, how fast, and how often a system reacts to the environment. Simulations are also important because they allow initial testing of a system in a controlled environment, with reproducible behaviour. This improves the tractability of the adaptation problem, as it is often difficult to trace areas of poor performance. The need for simulation is similarly important for self-adaptive systems in pervasive computing.

The area of pervasive computing that deals the most with self-adaptiveness is probably context-aware computing. Here, the environment that the system reacts to is the physical environment in which the system is deployed, e.g. an office,

a home, or the surroundings of a self-aware mobile device. In particular, such systems often seek to “augment” offices or homes with sensors (usually coupled to embedded wireless devices) with the goal of improving the working or living experience of people and reducing the need for them to interact with computers, as these computers become increasingly self-adaptive and autonomous. For instance, an office building may be “augmented” to determine continuously the location of the employees. This can be done using ultrasonic badges [2], RFID-tags with readers at the doors [3], InfraRed badges that periodically emit unique IR pulses [4], or by other means. A variety of self-adaptive applications can then make use of the knowledge of the location of its users, e.g. by tracking colleagues [3] or teleporting our virtual desktop to the PC in front of us [2], wherever we may be. Sensors are extremely important to context-aware applications, as much of the context information deduced from the environment is acquired by means of sensors [5]. While certain types of context information are static (e.g. a person’s birthdate) others are dynamic. Among these, the persistence of dynamic context information can be highly variable. For example, relationships between office colleagues typically endure for months or years, while a person’s location and activity often change from one minute to the next. It is in these highly variable activities that sensors play a fundamental role. Furthermore, sensors allow context sensing to be unobtrusive, not requiring the user to explicitly input context information [6], thereby making computing “calmer”. Our work will focus on this type of context, i.e. the location and activities of a person, concentrating on domestic environments.

While initially most research on context-awareness focused on office or academic environments, in recent years there has been an increased number of projects focussing on the home. For example, Kidd et al. [7] describe a project where a two-family house was built as a laboratory for research in smart homes. Various research topics are being worked on there, including a “smart floor” that can identify and follow individuals based on their footsteps, and a “frequently lost objects” tracking system.

II. PROJECT DESCRIPTION

More and more projects are working on building context-aware applications that monitor the home and improve the lifestyle, health or security of its inhabitants. However, when it comes to simulating the environment to measure responsiveness of the self-adaptive system, these applications usually

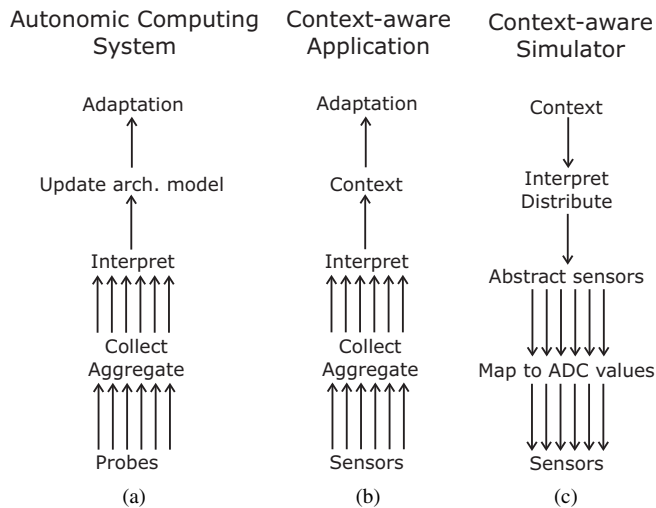


Fig. 1. Monitoring in autonomic computing systems (a) and context-aware applications with bottom-up flow of data (b) vs. context-aware simulator with top-down flow of data (c).

require setting up mock up rooms in laboratories and installing (often expensive) sensor hardware. However, we do not usually have the luxury to build an entire home as a laboratory (as was done in [7]), and quite often there is a lack of funds and space for smaller context-aware projects. Furthermore, when a real-life testing environment is used with physical sensors, it can be extremely difficult to reproduce testing conditions.

Consequently, the goal of this project is to attempt to design and build a simulation framework for context-aware applications. Our hypothesis is that the development of the context logic requires a careful study of the relationship between environment sensor data and abduced context, and this can be exploited to build a simulation model of this relationship, thus allowing context logic to be incrementally tested in a controlled environment already during development. The project will focus on domestic environments, although the framework must be flexible.

We describe a simulation model for context-aware applications by looking at how the standard approach to building context-aware applications can be inverted to build a simulator (see Figure 1). While context-aware applications start from raw sensor data and use functions to operate on the data, collect and aggregate them and then analyse them to determine a context (Figure 1 also shows how this maps to the standard monitoring paradigm in autonomic computing systems), our project takes this approach and inverts it (Figure 1(c)). We start from a given context and try to move down to sensor data. Both directions require an understanding of the relationship between context and sensor data. Also, the model needs to be flexible to allow extensions to new types of context and to sensors operating on different platforms. While there have been attempts to define application frameworks for context-aware applications, which abstract the sensors that provide context information, there is likewise a need in a simulation framework to detach context description from the sensors that use it to produce data.

This simulator can be considered a tool targeted at devel-

opers who want to test a context-aware application's context and autonomic logic prior to real-life deployment and testing. A simulated environment allows the policies for an autonomic system to be evaluated under controlled and repeatable circumstances. We start by studying first the various approaches to developing context-aware applications, to see what aspects of the real world are relevant to these applications and should be modelled in our simulation model. As a simulation model is only a simplification of the world, it is important to try and understand what elements of the real world are sensible to the model, so that it can fulfill its purpose. For instance, in network simulations, poisson distribution is usually used to model sources of network traffic, and while it does not necessarily approximate real sources well [8], it often provides an adequate abstraction.

An important requirement in the project is that an application running on the simulator must in no way contain code that is specific to the simulator: it must be possible to run the exact same implementation on real sensor networks.

Because the simulator ultimately produces sensor data, it is completely decoupled from an application using simulated sensor data. Thus, the correctness of the context logic of an application must be verified on the application side. This is no different than when the application is tested at a later stage at a physical location.

A. Description of simulation model

The proposed model includes two distinct types of descriptions. One model describes the activities of the inhabitants at a high level, e.g. "get out of bed, get dressed, have breakfast, brush teeth, go out of home". This document completely abstracts from the actual topology of a home. The other model (see Figure 2(a)) describes the 2D map of a home and how its layout maps to the high level activities of the former model. Based on ideas from [9], we choose to define "hotspots" in the 2D map where activities can take place. So, for example, going to bed is associated with the "bed" hotspot in the bedroom, brushing ones teeth with the "wash basin" hotspot in the bathroom and so forth. As a result, it is possible to apply a model describing people's activities to different home topologies, potentially obtaining substantially different results.

When a person executes an activity, it moves to the hotspot assigned for this activity. The movement is implemented in the person object itself, which receives a Δt of elapsed time and has to update its position. Thus, it is possible to plug in different implementations of people with more or less realistic movement models.

B. Simulating sensors

Sensors in the simulation model use knowledge about the location and current activity of people to determine what data they should produce. A major difficulty that arises in this area is how to map high-level activities to sensor data. This is complicated by the fact that some activities can have a high degree of uncertainty, e.g. the sound measured when a person watches TV depends very much on the TV programme being watched. We intend to start and implement passive

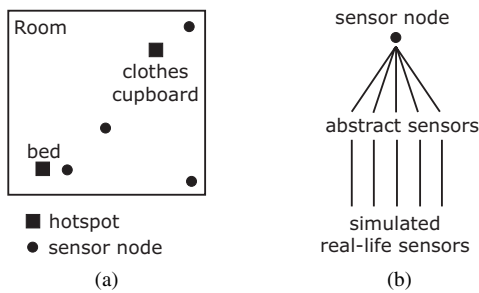


Fig. 2. (a): Model of a room. People's activities take place at hotspots. (b): A simulated sensor node.

proximity IR-sensors and pressure sensors (in beds, chairs and other locations) running on TinyOS nodes [10], most notably the Berkeley motes [11], and RFID-tags for proof-of-concept. These sensors are less dependent on the uncertainty factor of the current activity of a person. Other sensors may follow. However, the greater goal of the project is to design a framework for “plugging in” different types of sensors and activities.

The simulation model is not intended to simulate more complex sensors such as video cameras. On the other hand, Schmidt et al. [12] argue that it is preferable to augment devices with many simple sensors (the idea being that they can be made small and cheap, and thus ubiquitous) that each individually capture just a small aspect of an environment. The combination of these sensors, however, can result in a total picture that may be able to better characterise a situation than (location- or) vision-based context. In particular, vision-based context uses few very complex sensors that require a great amount of processing to derive information from just a single source (or small number of sources). Therefore, our simulator moves in the direction of multiple simple sensors, similarly to the vision of Schmidt et al. [12], [13]. But, a major difference is that, while they focus on self-awareness of mobile devices that have sensors attached to them, we look at awareness of a home environment, with sensors ubiquitously placed in the home.

In a room, nodes that contain sensors are placed (Figure 2). While abstract sensors provide environment-related values, e.g. temperature in room expressed in $^{\circ}\text{C}$ or true/false whether a proximity sensor detects activity, simulated real-life sensors map abstract sensor values to ADC (Analog to Digital Conversion) values for a particular sensor, e.g. a 10-bit number for typical sensors on Berkeley motes running TinyOS¹.

C. TOSSIM: a simulator for TinyOS

Researchers at Berkeley have developed a simulator for TinyOS, called TOSSIM [14], which allows users to compile and run TinyOS code on a PC to simulate a number of motes interacting. Applications can then connect to TOSSIM just as they would connect to a network of motes². Because TOSSIM

¹There are temperature sensors that send on the ADC port the temperature directly in $^{\circ}\text{C}$, using auto-calibration.

²More precisely, applications on a PC or PDA connect via a serial port to a mote which serves as a gateway to the sensor network (see also Figure 3).

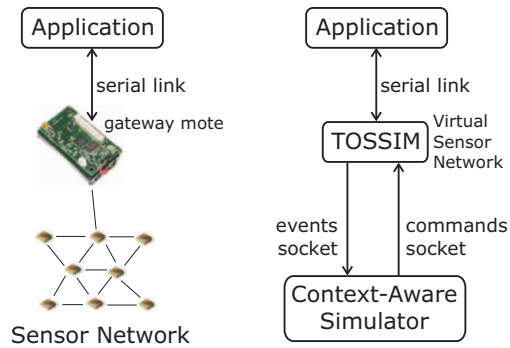


Fig. 3. Left: A context-aware application deployed on a real sensor network of motes. Right: The same application running with TOSSIM, with our simulator steering the properties of the motes in the virtual sensor network.

simulates mote communication at bit-level, it is very useful for testing ad-hoc routing algorithms or data aggregation algorithms. However, it is too low-level to easily test context-aware applications, which are less interested in the inner workings of the motes and more interested in the actions that cause sensor data to change. Fortunately, TOSSIM comes with a command port that allows a running simulation to be “steered”. As an example of the possibilities of the command port, TOSSIM comes with a graphical Java application, TinyViz [14], which allows the user to change simulation parameters in an executing simulation. Our simulator can therefore connect to this command port similarly to TinyViz and continuously instruct TOSSIM on what data the sensors on the motes should provide (by default, TOSSIM just generates random values and is completely ignorant of the different types of sensors on the different sensor ports). Also, motes can be moved over time as people in the simulation move around the home and carry objects that have been augmented with motes. Figure 3 illustrates this idea. If the application running on TOSSIM is TinyDB [15], then it is possible to trace on a graph the ADC values of a mote over time (see Figure 4, in which the light value is traced), thus showing how an external application can control simulation execution. In the graph in Figure 4, we initially let TOSSIM steer the light sensor. Then, after a few seconds, we manually cause the light sensor value to be set to 0 and increase at 100 increments up to 600, after which we decrease it again. As one can see in the graph, in the first few seconds, when the light sensor value is controlled by TOSSIM, TOSSIM just generates random values for the light sensor. To TOSSIM, the sensor ports are just ports that emit some random 10 bit value. However, once we start setting our own values, the light sensor in TOSSIM keeps this value until we change it again. Finally, another interesting control feature that can be used in our simulator is the ability to turn off motes at particular times. This can model mote failure and could be used to test the *self-healing* properties of an application. Self-healing is important when sensor nodes are involved, as node failures are a common occurrence and they may be deployed in a location where they cannot easily be replaced.

An important consequence of building our simulator on top of TOSSIM is that we have the guarantee provided by TOSSIM that applications that run on the simulator can be

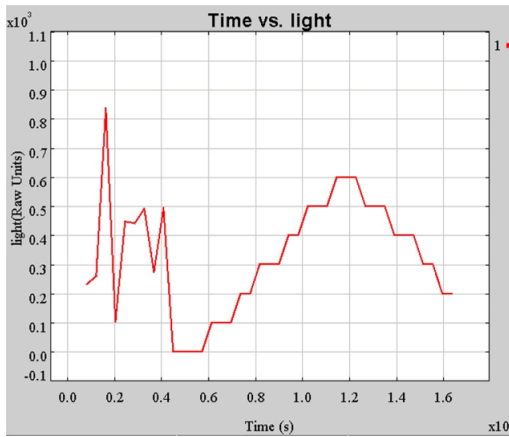


Fig. 4. TinyDB showing light sensor value being changed by example application.

recompiled to run with actual TinyOS motes without any change to application code.

A disadvantage of building on top of TOSSIM is that TOSSIM simulates the TinyOS network at bit-level precision. This requires a lot of computing power and as a consequence running a network of ca. 10,000 motes for 10 virtual seconds can take a few hours³. This strongly hinders running a simulation in near-real time. A solution to this would be to modify TOSSIM to use a higher-level of abstraction. For example, it has been shown that writing a simulation version of the RadioCRCPacket component of TinyOS that does not model packet collisions or MAC delays allows the aforementioned simulation to run in under a minute [14]. On the other hand, we do not envisage a home having so many sensors — hundreds of sensors will probably represent a realistic number — in which case the performance of TOSSIM in real time should prove to be sufficient with the performance of the latest PCs.

D. Simulation time

As our first prototype builds on top of TOSSIM, it is TOSSIM that generates the virtual flow of time for a simulation execution and informs our simulation of the current time. This happens every time TOSSIM sends some notification over its event communication port. However, TOSSIM does indirectly allow an external application to pause and resume or slow down and speed up a simulation. Pause and resume can be controlled externally by deciding whether or not to read events from TOSSIM, while simulation speed can be slowed down by introducing a delay between receipt of an event from TOSSIM and sending the associated acknowledgement.

When the simulation framework is used to perform simulations that do not involve TinyOS nodes and therefore TOSSIM, then our simulator naturally uses its own virtual clock for the flow of time.

³They show that a network-intensive application running on 8,192 simulated nodes for 10 virtual seconds takes 2.75 hours to simulate on a 1.8 GHz Pentium 4 machine with 1GB of memory running Linux 2.4.18.

III. RELATED WORK

Meyer and Rakotonirainy [6] have published a comprehensive survey of research on context-aware homes. They point out that the infrastructure of a context-aware home application has to be able to function without the constant surveillance of an on-site administrator, as is the case with offices. Thus, there is a need for these infrastructures to be less complex and more manageable by the average user, i.e. they need to be more *self-managing* and *self-healing* than context-aware applications in an office environment need to be. They define context as *the circumstances or situations in which a computing task takes place*. Further, they define the context of an entity A as *any measurable and relevant information that can affect the behaviour of A*. In the research involving middleware, two influential projects they mention are the Context Toolkit by Dey et al. [16] and the sensor architecture TEA by Schmidt et al. [13].

The Context Toolkit [16] is a framework aimed at facilitating the development and deployment of context-aware applications. It abstracts context services, e.g. a location service, from the sensors that acquire the necessary raw data to deliver the service through a network API. Further, the Context Toolkit allows sharing of context data through a distributed infrastructure and collection of storage data to create a history. Applications can poll for context information, or they can also subscribe to context data, allowing the application to be notified of context changes. Context widgets (much like GUI widgets) are reusable units that provides context services, e.g. an activity widget sensing activity in a room. Widgets take care of acquiring raw sensor data, interpreting them and abstracting them to high-level context information. In relation to our simulator, if context widgets for TinyOS sensors were created in the Context Toolkit, then our simulator could produce sensor data for these context widgets that would use them to derive some context. From an autonomic computing point of view, it would be interesting to see how the Context Toolkit could be extended to provide the basic building blocks for self-managing context-aware applications: for instance, an application deployed in a new location could adapt to the services that are available, regardless of what types of sensors are installed. Also, when a sensor fails, the Context Toolkit could rebind the service to a different context widget that uses a different set of sensors, therefore providing self-healing mechanisms.

The Technology Enabling Awareness (TEA) sensor architecture by Schmidt et al. [13] is a middleware layer that allows context information to be determined by a group of simple sensors. In particular, the aim is to derive more context information from a group of sensors than the sum of context derived from individual sensors [12]. The architecture is layered: the lowest layer retrieves raw sensor data, the next layer (cues) derives features from individual sensors (there can be more than one cue per sensor) and the context layer derives context from multiple cues, using logical recognition rules. TEA is different from the Context Toolkit in that it is more specifically aimed at self-contained awareness devices, e.g. a mobile phone with various sensors attached to it that

deduces context on its own.

As for smart homes, there have been various projects that involved a physical home or individual rooms as a laboratory for smart home projects. [17] offers an overview of such projects and others more generally related to ubiquitous computing.

The work that is most similar to our project is probably GLS [18], the Generic Location event Simulator that has been developed in the context of the QoSReAM middleware project [19] at Cambridge University. The focus of the project is to model realistically the movement of people in an environment. Control theory is used to model people the way autonomous guided vehicles are usually modelled. People, called “locatables”, automatically find a shortest path to a destination and avoid obstacles and other people, whereby differential equations are used to realistically model changes in velocity. Furthermore, different locatables have different turning rates and turn radii. Mechanics and Queuing Models define the behaviour of locatables, while Sensor and Environment/Error Models simulate the physics in the room, e.g. how ambient light can affect an infrared-based sensor. Finally, the World Model represents the building’s geometry. As locatables move in a map, simulated location sensors output data in the same format as real world sensors would.

IV. CONCLUSION

Our project aims to describe a simulation model for context-aware applications by looking at how the standard approach to build context-aware application frameworks can be inverted to build a simulator for such applications. The model needs to decouple the elements that provide context from the sensors that use them to determine sensor data, just like in a context-aware application framework the inverse is true, i.e. raw sensor data is decoupled from the context that is derived from them. This is reflected in traditional autonomic systems by the need to decouple high level notions in the adaptation model from raw monitoring data acquired by probes in the environment (here intended as a software or network environment, where a probe might be measuring the number of occurrences of a particular system call). For a specific context-aware project, the simulation model should be defined in parallel to the development of the context logic — similarly to the incremental Unit testing in XP — as both require an understanding of the relationship between high-level context on low-level sensor data.

REFERENCES

- [1] D. Garlan and B. Schmerl, “Exploiting architectural design knowledge to support self-repairing systems,” in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, July 2002.
- [2] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, “The anatomy of a context-aware application,” *Wireless Networks, ACM*, vol. 8, no. 2/3, Mar. 2002.
- [3] W. Trumler, F. Bagci, J. Petzold, and T. Ungerer, “Smart doorplate,” *Personal Ubiquitous Comput.*, vol. 7, no. 3-4, pp. 221–226, 2003.
- [4] V. Stanford, “Using pervasive computing to deliver elder care,” *IEEE Pervasive Computing*, vol. 1, no. 1, pp. 10–13, 2002.
- [5] K. Henriksen, J. Indulska, and A. Rakotonirainy, “Modeling context information in pervasive computing systems,” in *Proceedings of the First International Conference on Pervasive Computing*. Springer-Verlag, 2002, pp. 167–180.

- [6] S. Meyer and A. Rakotonirainy, “A survey of research on context-aware homes,” in *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*. Australian Computer Society, Inc., 2003, pp. 159–168.
- [7] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. D. Mynatt, T. Starner, and W. Newstetter, “The Aware Home: A living laboratory for ubiquitous computing research,” in *Cooperative Buildings*, 1999, pp. 191–198. [Online]. Available: citeseer.nj.nec.com/kidd99aware.html
- [8] D. K. Arrowsmith and M. Woolf, “Modelling of TCP packet traffic in a large interactive growth network,” in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, 2004.
- [9] A. Crabtree, T. Hemmings, and T. Rodden, “Pattern-based support for interactive design in domestic settings,” in *Proceedings of the conference on Designing interactive systems*. ACM Press, 2002, pp. 265–276.
- [10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System architecture directions for networked sensors,” *SIGPLAN Notices*, vol. 35, no. 11, pp. 93–104, Nov. 2000.
- [11] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, “Connecting the physical world with pervasive networks,” *Pervasive Computing, IEEE*, vol. 1, no. 1, pp. 59–69, Jan.-Mar. 2002.
- [12] H. W. Gellersen, A. Schmidt, and M. Beigl, “Multi-sensor context-awareness in mobile devices and smart artifacts,” *Mob. Netw. Appl.*, vol. 7, no. 5, pp. 341–351, 2002.
- [13] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde, “Advanced interaction in context,” *Lecture Notes in Computer Science*, vol. 1707, 1999. [Online]. Available: citeseer.nj.nec.com/schmidt99advanced.html
- [14] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: accurate and scalable simulation of entire tinyos applications,” in *Proceedings of the first international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 126–137.
- [15] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “The design of an acquisitional query processor for sensor networks,” in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM Press, 2003, pp. 491–502.
- [16] D. Salber, A. K. Dey, and G. D. Abowd, “The context toolkit: aiding the development of context-enabled applications,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, 1999, pp. 434–441.
- [17] P. Phillips, “Smart environment research,” Workshop on Ubiquitous Computing in Domestic Environments, Nottingham, 13–14 Sept. 2001.
- [18] K. Sanmugalingam and G. Coulouris, “A generic location event simulator,” in *Proceedings of the 4th international conference on Ubiquitous Computing*. Springer-Verlag, 2002, pp. 308–315.
- [19] H. Nagueib and G. Coulouris, “Location information management,” in *UbiComp*, 2001, pp. 35–41. [Online]. Available: citeseer.ist.psu.edu/nagueib01location.html



Markus C. Huebscher received his Diplom degree at the Swiss Federal Institute of Technology Zürich (ETH) in 2002 and is now working on his PhD thesis at Imperial College London. His research interests include self-adaptiveness in context-aware applications, wireless sensor networks and mobile hand-held devices.



Julie A. McCann Julie McCann received her PhD doctorate in 1992 in Ireland, and then joined City University, Computer Science Dept. where she became the leader of the Systems Architecture Research Centre, managing many Operating System, Database and Distributed systems research projects. She currently lectures in Database Systems Operating Systems. Her research lies in adaptive systems architectures for improved quality of service and performance. Julie is a Chartered Engineer and a Member of the British Computer Society.