Model Checking for Sentient Computing: An Axiomatic Approach

Eleftheria Katsiri
Laboratory For Communication Engineering
University of Cambridge
William Gates Building
15 JJ Thompson Avenue
Cambridge
CB3 0FD, UK
ek236@cam.ac.uk

Alan Mycroft
Computer Laboratory
University of Cambridge
William Gates Building
15 JJ Thompson Avenue
Cambridge
CB3 0FD, UK
am@cl.cam.ac.uk

ABSTRACT

Sentient Computing allows applications to best interact with their physical environment, by becoming *aware* of their surroundings. Awareness is achieved by means of a sensor infrastructure that helps maintain a *model* that represents the current state of the dynamically changing world. This model can be seen as a *concrete* interpretation of the physical environment and conceptually stands between the physical world and the *abstract* view of applications. A number of factors such as the non-homogeneity of physical space and the precision of the sensor technology may introduce errors and inconsistencies between the physical world and the model. On the other hand, the abstract view of the application domain needs to be correct and compatible with the concrete model, especially in the case of distributed, heterogeneous environments where applications need to interact *seamlessly* with several different concrete models.

The contribution of the work described in this paper is that it looks at the above problem as a *constraint-satisfaction* problem to which it applies classical, logical satisfiability, in order to produce a model-based solution. Our system is similar to a classical *model-checker*; it checks the *satisfiability* of the application requirements against the world model, as well as the consistency of the world model with the properties of the actual physical environment. Because it is model-based, it is appropriate for distributed, heterogenous environments. Our system forms part of *SCAFOS*, a generic, distributed middleware framework for context-awareness, implemented in the first author's PhD dissertation. An implementation that uses the theorem prover system *SPASS*, is also discussed in this paper.

1. INTRODUCTION

Model checking[4] was proposed by Clarke as a method for formally verifying finite-state concurrent systems, where specifications about the system are expressed as temporal logic formulas. Efficient symbolic algorithms are used to traverse the model defined by the system and check if the specification holds or not.

Sentient Computing [8] is a programming paradigm, in which applications are made more *aware* of their physical environment by means of a *sensor infrastructure*. Sensors help maintain a model of the physical environment, referred to as a *Sentient environment*. The authors' previous work [11, 12] proposes a model for inferring high-level, abstract knowledge from low-level, concrete knowledge, that is directly derived from sensors by means of frequent updates through the *SensorUpdate()* interface (Figure 1). Abstract state is deduced from concrete state by means of a definition in temporal first-order logic (TFOL). The *Abstract Event Definition Language (AESL)*, defined in [11], is used in order for an application to subscribe to changes in the truth values of high-level abstract state predicates, referred to as *abstract events*. A higher-order service, called an *Abstract Event Detection Service*, publishes its interface. This service takes an AESL definition as an argument, and in return publishes an interface to an abstract event detector that notifies transitions between the values true and false of the formula. For example, an application that is interested in locating the *closest, empty meeting room to Brian, who is a system administrator* needs to subscribe to the Abstract Event Detection Service providing a TFOL formula that defines the abstract state predicate *H_ClosestEmptyLocation(Brian, System Administrator, rid, Meeting Room.)*[1] The system keeps abstract state consistent with concrete state at all times.

However, this dual *abstract mapping*, i.e., from the physical environment to the concrete model and from the concrete model to the application layer, can be affected by a number of factors, in terms of *correctness, completeness and consistency* of both the model and the application specifications. The most important of these factors are summarised below:

- *Non-homogeneous space and the natural laws of physics.* The application specifications may contain logical fallacies that are caused by ignoring the physical constraints that are introduced by the spatial topology, i.e., a person can not be in more than one location

[1]Nomenclature taken from [12]

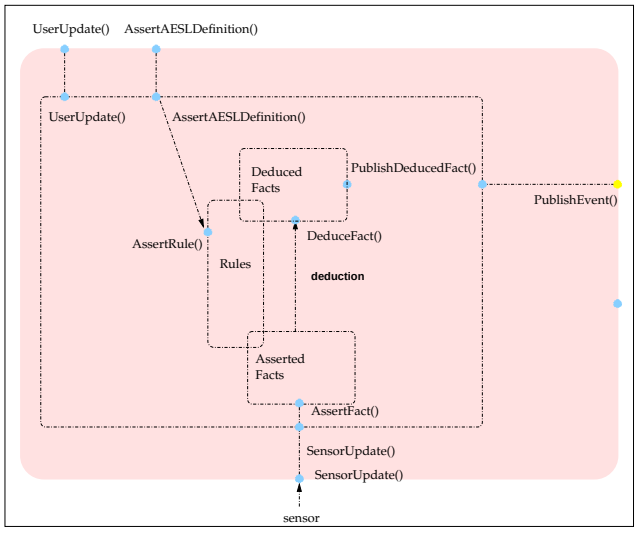

Figure 1: The deductive knowledge base and its API

simultaneously and he cannot move through walls.

- *Semantic sufficiency of the model.* The application is not aware of the correctness or the granularity of the model of the physical world which depends directly on the capabilities of the underlying location system. For example, a model that is updated by Active Badge [21] location sightings, only knows about rooms. Such a model cannot reason with any application requirements that involve positions, or regions smaller than rooms. For example, such a model cannot determine which PC in the room, the user is closer to, and therefore this application requirement would be *unsatisfiable* by the specific model. However, a model that depends on the Active BAT [7] for location information, knows the exact position a user is in and can deduce much more abstract state in order to satisfy the application requirements.

- *Correctness of abstract application specification.* The application specification may be incorrect in which case it will not be satisfiable by the model even if the model is semantically adequate. Errors can occur from violating logical constraints such as the ones that are derived from the functional operation of location predicates. For example, the situation where the C expert is typing at his keyboard while at the same time the systems administrator is having a coffee is impossible if the system administrator and the C expert are the same person. Feedback is returned to the application in such a case.

We propose a verification technique similar in concept to model checking that checks abstract knowledge definitions (AESL definitions) for correctness and semantic compatibility with the models of the sensor-driven domains. Correctness is established against a set of spatio-temporal constraints and against a set of logical constraints that make sure that logical fallacies are excluded. The Sentient model is also checked for correctness by checking each sensor update against a set of spatio-temporal specifications. Whenever a specification is found unsatisfiable, appropriate feedback can be given to the application that can be used for the selection of a more appropriate model, or an adaptation in the application's behavior.

2. FACTORS THAT AFFECT MODELING

The main factors that make the concrete model incomplete or inconsistent with the physical environment is the combination of the behavior of the sensor technology that instruments physical space and the non-homogeneity of physical space itself. The main sensor technology used for physical space modeling is *location* technology. *Location technologies* vary in precision, and an imprecise or an incorrect reading will introduce an error in the concrete model. Because space can differ so much from one cm to another, such an error can lead to an *illegal* model state. For example, a user can be seen floating in the air, or walking inside a wall.

Similarly, the application abstract view can be incomplete or inconsistent compared to the concrete model. Inconsistencies here can be introduced via incorrect application specifications e.g., an application specification may describe an abstract state where the application is interested in paging *the first available C++ expert, while the system administrator is unavailable*. This particular abstract state cannot be reached, as long as the system administrator is the only C++ expert.

Another constraint derives from the situation in which an application interacts with more than one concrete model at different levels of *knowledge* precision. For example, the above application specification would be *unsatisfiable* for a model that only knows which room a user is in and therefore cannot determine whether one is using a PC or not. The proposed system prevents the above problems by checking the concrete model against a set of specifications that prevent the modeling of illegal states.

3. SPECIFICATIONS

Specifications are described in terms of FOL axioms, and can be classified in three sets. *Spatial and logical* specs, *Model* specs and *Application* specs. *Spatial and logical specs* are used to check the consistency of the model against the physical environment. They represent physical constraints that apply to all Sentient environments (see Section 2). *Model* specs are used to determine whether the abstract specifications are compatible with the concrete model or whether the model is semantically sufficient for the application specification. *Application specs* [2] are provided by the applications in the form of axioms and aim to bind concrete state predicates to a high-level, abstract state predicate, referred to in this paper as the *goal theorem*. Goal theorems are checked for satisfiability against application specs, model specs and spatial and logical specs. Moreover, each sensor update is checked for satisfiability against spatial and logical specs.

3.0.1 Overall Specification

The satisfiability of the overall specification is modeled by the predicate *SpecSuccess*, which is defined as equivalent to the conjunction of the predicates that signal the satisfiability of the application model and spatial specifications,

[2]Application specs are equivalent to AESL definitions.

respectively:

$$AppSuccess \wedge \neg ModelFailure \wedge SpatialSuccess$$
$$\Rightarrow SpecSuccess$$

4. FIRST-ORDER LOGIC DESCRIPTION OF A SENTIENT MODEL

Using the definitions and nomenclature proposed in [12] and extending the temporal notions of [9], we define a *state-based* model for distributed sensor-driven systems. We assume that a system consists of several physical *domains* such as an office domain. Each domain contains a set of physical objects such as mobile users and equipment. Modeled physical locations include *regions*, ranged over *rid*, and *positions (x,y,z)*. Each user *(uid)* is associated with a *role* and each region is associated with a *regional attribute (rattr)* that describes relevant contextual information such as its functionality or ownership, e.g., kitchen, Alan's office etc. This allows the expression of *semantic abstractions* such as *the closest system administrator who is not busy.* Objects, locations, roles and regional attributes have *states* which are either concrete or abstract. Concrete state predicates represent state that is directly derived from the sensors and in this paper are always prefixed with *'L_'* (Low-level). Example of concrete state predicates are those that represent a user's position in terms of his co-ordinates, rooms in a building and nested locations such as floors. These are modeled [12] with the first-order logic predicates:[3]

$$L\_UserAtPosition(uid, role, (x, y, z), timestamp)$$
$$L\_AtomicLocation(rid, rattr, polygon)$$
$$L\_NestedLocation(rid, rattr, rid\text{-}list)$$

For example,

$$L\_UserAtPosition(John, Phd, (13.4, 5.7, 1.7), 13{:}56)$$
$$L\_AtomicLocation(Room7, Meeting\text{-}room, Polygon37)$$
$$L\_NestedLocation(Floor4, [Room2, Room9])$$

A set of *function* predicates, as in Prolog, represent functions where the first argument represents the value of the function. The distance between a user's position and a fixed point in the polygon that defines a region is represented through the predicate $Distance(v, role, rid, rattr, uid)$.

Abstract state predicates represent high-level state that is derived from concrete state by means of TFOL on properties of interest. A user's high-level location in terms of the region that contains him, a user's presence or absence and the fact that one or more people are co-located are examples of such predicates. Initially, when the system is started up, only concrete state predicates exist. Abstract state predicates in this paper are always prefixed with *'H_'* (High-level), $H\_UserAtLocation(u, role, rid, rattr, t)$.

A *timestamp* denotes the moment when a *current* abstract predicate instance becomes active. Previous instances are stored as *historical* predicates and are associated with an additional timestamp that denotes the moment when the predicate instance became inactive. Timestamps allow for *temporal abstractions* such as *now, today* and *temporal operations* such as sequence, iteration, equality and inequality with temporal intervals, over current and historical data.

[3]As a convention, predicate attributes are in lower case and their values have the initial character capitalised, e.g., *polygon* is a variable whereas *Polygon27* denotes the value of the attribute *polygon* in a given coordinate system.

4.1 Spatial and Logical Specifications

Spatial and Logical Specifications aim to represent constraints that derive from the characteristics of physical space (see Section 2). Both application-provided theorems and new facts produced by the sensor infrastructure are checked against this set of specifications. The following axioms represent such specifications:

AXIOM 1. *Each user can be in only one position at a time.*

$$(x_1 = x_2) \wedge (y_1 = y_2) \wedge (z_1 = z_2)$$
$$\vee \neg L\_UserAtPosition(uid, role_1, x_1, y_1, z_1)$$
$$\vee \neg L\_UserAtPosition(uid, role_2, x_2, y_2, z_2)$$

The above axiom prevents the application from making a specification that implicitly requires the same user to be in two different positions at the same time, usually under a different role, e.g., a user who is both a system administrator and a C++ expert. We can also say that *L_UserAtPosition* is a *function* in the domain of the concrete model. A direct consequence of the above axiom is the following:

AXIOM 2. *If an object is contained in more than one region, these are nested.*

$$L\_NestedLocation(rid_2, rattr_2, site\text{-}list_2)$$
$$\wedge InList(rid_1, site\text{-}list_2)$$
$$\vee L\_NestedLocation(rid_1, rattr_1, site\text{-}list_1)$$
$$\wedge InList(rid_2, site\text{-}list_1)$$
$$\vee \neg H\_UserInLocation(uid, role_1, rid_1, rattr_1)$$
$$\vee \neg H\_UserInLocation(uid, role_2, rid_2, rattr_2)$$

The above axioms denotes that if a user is known by the system to be inside two different regions then the regions are nested [4].

AXIOM 3. *A user cannot be located inside a compact surface such as a wall.*

$$IsOpaque(x_1, y_1, z_1)$$
$$\Rightarrow \nexists L\_UserAtPosition(uid, role, x_1, y_1, z_1)$$

This axiom can be used to discover an erroneous location system sighting. The predicate *SpatialSuccess* is used in order to denote that all axioms in this category are satisfied.

4.2 Model Specifications

Model specifications are used in order to determine whether the concrete model is precise enough for the application specification. Model specifications are provided both by the application and the sensor-driven model side. The application provides an atomic formula:

$$RequiredPrecision(p, q)$$

[4]Here it is assumed that no regions are overlapping.

The pair p, q is a confidence level [5] that characterises the precision of the location modeling. The term p takes values from the set $\{reg, coord\}$ and denotes the type of the location technology (region-based or coordinate-based) , and $q \in [1 \cdots 10]$ represents the accuracy of the coordinate system [6]. The higher the accuracy, the higher q. The concrete model provides the atomic formula:

$$ProvidedPrecision(p, q)$$

The pair p, q is also a confidence level for location modeling precision. Similar predicates can be invented for other types of information.

AXIOM 4. *The knowledge precision required by the application should be no greater than the knowledge precision offered by the underlying model.*

$$\begin{aligned}
& RequiredPrecision(reg, 0) \wedge ProvidedPrecision(reg, 0) \\
\Rightarrow\ & \neg ModelFailure(reg, 0) \\
& RequiredPrecision(reg, 0) \wedge ProvidedPrecision(coord, q) \\
\Rightarrow\ & \neg ModelFailure(reg, coord) \\
& RequiredPrecision(coord, q_1) \wedge ProvidedPrecision(coord, q_2) \\
& \wedge > (q_1, q_2) \\
\Rightarrow\ & ModelFailure(q_1, q_2) \\
& RequiredPrecision(coord, q_1) \wedge ProvidedPrecision(coord, q_2) \\
& \wedge > (y, x) \\
\Rightarrow\ & \neg ModelFailure(q_1, q_2) \qquad (1)
\end{aligned}$$

Initially the predicate *ModelFailure* is set to true. Each axiom which is satisfied sets it to false. If *ModelFailure* equals false, the model specifications are satisfied.

4.3 Abstract Knowledge Definitions (AESL Definitions)

Assume an application which needs to determine the closest empty meeting room to the CTO of a company so that it can initiate a tele-conference. It is, therefore, interested in knowing when the predicate $ClosestEmptyLocation(uid, CTO, rid, Meeting\ Room)$ is true, as well as the value of rid. Because such a predicate does not exist in the system, the application needs to bind this to a set of facts that the system knows about in a logical way.

The *AESL definitions* can be seen below (for reasons of simplicity we assume that the predicate *ClosestLocation* is calculated by the system using the predicate *Distance*). Writing for brevity UL for $H\_UserInLocation$, EL for $H\_EmptyLocation$, CL for $H\_ClosestLocation$, CEL for $H\_ClosestEmptyLocation$ and D for $Distance$:

$$\begin{aligned}
& RequiredPrecision(reg, 0) \\
& \nexists uid\ UL(uid, role, rid, rattr) \Rightarrow EL(rid, rattr) \\
& D(v_1, uid, role, rid_2, rattr_2) > D(v_2, uid, role, rid_1, rattr_1) \\
\Rightarrow\ & CL(uid, role, rid_1, rattr_1) \\
& CL(uid, rid, role, rattr) \wedge EL(rid, rattr) \\
\Rightarrow\ & CEL(uid, rid, role, rattr) \qquad (2)
\end{aligned}$$

[5]Here, it is assumed either coordinate or containment granularity for the location system, and a confidence level for the precision of the coordinate system in 95% of the measurements (see section 5.2).

[6]If p='reg' then q=0 by default.

The goal theorem is the theorem that needs to be checked for satisfiability. Satisfiability of the goal theorem implies the satisfiability of the application specification, i.e., the predicate *AppSuccess*, which is initially true, remains true. In this case,

$$\begin{aligned}
& \exists rid(CEL(uid, CTO, rid, Meeting\ Room)) \\
\Rightarrow\ & AppSuccess \qquad (3)
\end{aligned}$$

5. PROOF BY RESOLUTION AND SATISFIABILITY

The contribution of this work is based on using the concept of *satisfiability* as the foundation for evaluating the conformance of application specifications and sensor updates to the Sentient model.

A *satisfiability* problem in conjunctive normal form (CNF) consists of the conjunction of a number of clauses where a clause is a disjunction of a number of variables or their negations. Given a set of clauses $C_1, C_2, \cdots C_m$ on the variables $x_1, x_2, \cdots, x_n$, the satisfiability problem is to determine if the formula

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

is satisfiable, that is if there is an assignment of values to the variables so that the above formula evaluates to true. Clearly, this requires that each C_j evaluates to true.

Automatic theorem provers aim to find a *proof* for a theorem given a set of axioms that are known to be true. A common method for finding a proof is by *resolution*. According to proof by resolution, the set of axioms and the goal theorem are transformed to conjunctive normal form (CNF), and resolution is applied to the resulting set of clauses. Existence of proof is equivalent to the satisfiability of the set of clauses.

5.1 The Theorem Prover SPASS

SPASS [22] is a first-order logic theorem prover with support for equality, and it was used in a prototype implementation of a satisfiability service whose API is shown in Figure 2. SPASS also features a Web interface (Web SPASS) [20], as well as integrated support for transforming FOL formulas into a small number of conjunctive normal form (CNF) clauses [15] before testing for unsatisfiability.

5.2 Example

Let us assume that the above application specification needs to be tested against a model which knows of locations in terms of coordinates, with accuracy of 3 cm, in 95% of the cases, such as the Active BAT. This can be encoded with the predicate $ProvidedPrecision(coord, 9)$. When tested with SPASS, the goal theorem is found to be correct.

```
SPASS V 2.0
SPASS beiseite: Proof found.
Problem: /tmp/webspass-webform_2003-09-09_00:40:50_29751.txt
SPASS derived 2 clauses, backtracked 0 clauses
      and kept 57 clauses.
SPASS allocated 528 KBytes.
SPASS spent     0:00:00.33 on the problem.
                0:00:00.04 for the input.
                0:00:00.09 for the FLOTTER CNF translation.
                0:00:00.02 for inferences.
                0:00:00.00 for the backtracking.
                0:00:00.04 for the reduction.
```

The *LoadSentientModel()* interface is used for loading SAL facts that denote the specific implementation of each do-

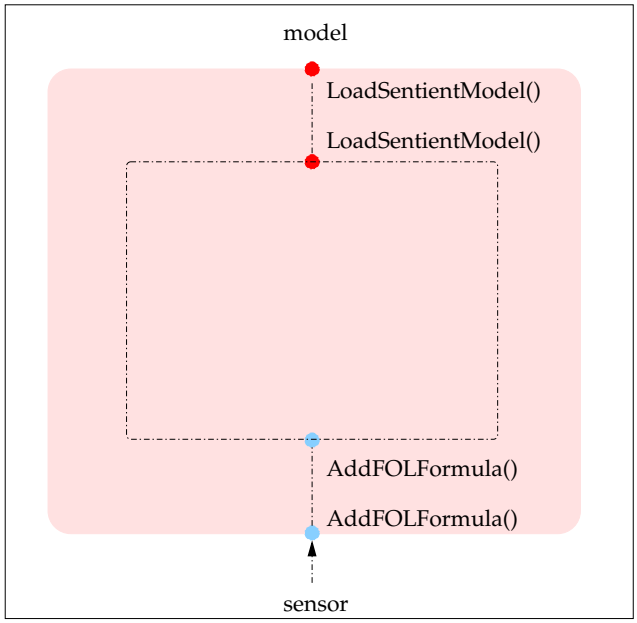

Figure 2: The Satisfiability Service and its API.

main, in terms of the topology, the number of users and the specific requirements of the instrumentation technology. The *AddFOLFormula* interface checks the satisfiability of a logical formula that represents an application definition, as described in this paper. Such application definitions correspond to AESL definitions discussed in [11]. Through the same interface, a SAL fact is asserted, derived from a sensor and it is checked for consistency with the FOL model.

6. RELATED WORK

A number of advanced sensing technologies were developed in the last few years in order to provide context-awareness. The Xerox Palo Alto Research Center's *Ubiquitous Research* program is one of the seminal projects in context-aware computing [18]. This project introduced the PARCTAB, a personal digital assistant (PDA) that communicates via infrared (IR) data-packets - including its position within a building - to a network of IR transceivers. The Active Badge system [21] was the first indoor badge sensing system; it was developed at Olivetti Research Laboratory. It consists of a cellular proximity system that uses infrared technology. A central server collects the data from fixed infrared sensors in the building, aggregates it, and provides an application programmable interface for using the data. The Active BAT [7] is a next-generation location system that uses an ultrasound time-of-flight trilateration[7] technique to provide more accurate physical positioning than the Active Badge. Users and objects carry Active BAT tags. The system can locate BATs to within 3 cm of their true position for 95 percent of the measurements. It can also compute orientation information. Other location technologies include the Global Positioning System (GPS), Cricket [16], which is also based on ultrasound, RADAR [1], which is based on IEEE 802.11 WaveLAN wireless networking technology and TRIP [13] (Target Recognition using Image Processing), a vision-based sensor system that uses a combination of 2-D circular barcode tags or *ringcodes* and inexpensive CCD cameras in order to identify and locate tagged objects in the camera's field of view. Last, Ultra Wideband (UWB) [6] is a radio technology that can also be used for very high-resolution radars and precision (sub-centimeter) location and tracking systems.

However, this rapid evolution of location technologies and systems, did not go hand-in-hand with comparable development in organising, interpreting and using the produced data. There is currently only a handful of sensor-driven data-management platforms. CoolTown[3], the Context-Toolkit [5], SPIRIT [7], LIME[14] and QoSDREAM [17], are among the most significant such systems. Although these systems have made important contributions in this field, none of these systems have investigated the issue of *correctness* and *satisfiability*. SCAFOS [10], is a next-generation, model-based, middleware platform implemented in the first author's PhD dissertation. SCAFOS enables uses to infer high-level knowledge from low-level sensor-driven concrete knowledge. The work described in this paper is part of SCAFOS.

Last, complementary to this work is the work of Scott et al [19], who use *mobile ambients* [2] to model user mobility and to design appropriate spatial policies.

[7]Trilateration is a method of surveying analogous to triangulation, in which each triangle is determined by the measurement of all three sides.

7. CONCLUSIONS AND FURTHER WORK

The work presented here is a model-based satisfiability checking tool, for context-awareness in sensor-driven systems such as Sentient Computing. Because of its model-based nature and its expressive, TFOL based language support, our tool constitutes a generic solution for distributed, heterogeneous context-aware environments. Our tool is part of SCAFOS [10], a framework for extracting high-level inferences from ubiquitous systems, developed in the first author's thesis [10]. TFOL specifications of high-level knowledge predicates and their transitions from *true* to *false* and vice-versa, (*AESL Definitions*) are checked by the Satisfiability Service for correctness and semantic compatibility with Sentient models was presented in this paper. Three types of correctness specifications have been discussed: spatial and logical specifications, model specifications and application specifications. The Satisfiability Service is based on the a novel application of a theorem proving system, such as SPASS, and programming logic has been developed that ensures that all specifications are satisfied for the overall application specification to hold.

Further work in this area could provide a model for mobility that limits the data that can be collected for a user as he moves. Furthermore, it will be very interesting to investigate the utility of *ontologies* in heterogeneous settings, such as the ones described here.

8. ACKNOWLEDGEMENTS

The authors would like to thank Mike Gordon and Andy Hopper for discussion and guidance. Eleftheria Katsiri would like to thank the University of Cambridge and Clare College for partially funding this work.

9. REFERENCES

[1] P. Bahl and V.N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *Proceedings of IEEE INFOCOM 2000 (2)*, pages 775–784, Tel-Aviv, Israel, Mar. 2000. IEEE Computer Society Press.

[2] L. Cardelli and A. Gordon. Mobile Ambients. In *FoSSaCS '98: Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structures*, pages 140–155, Lisbon, Portugal, Mar.–Apr. 1998. Springer-Verlag.

[3] D. Caswell and P. Debaty. Creating Web Representations for Places. In *HUC '00: Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing*, pages 114–126, Bristol, UK, Sep. 2000. Springer-Verlag.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[5] A. K. Dey and G. D. Abowd. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, 2000.

[6] R. Fleming and C. Kushner. Low-Power, Miniature, Distributed Position Location and Communication Devices Using Ultra-Wideband, Non-Sinusoidal Communication Technology. Technical report, Aether Wire Location, 1995.

[7] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. In *MobiCom '99: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 59–68, Seattle, WA, Aug. 1999. ACM Press.

[8] A. Hopper. The Clifford Paterson Lecture: Sentient Computing. *Philosophical Transactions of the Royal Society of London*, 358(1773):2349–2358, August 1999.

[9] D. Ipiña and E. Katsiri. A Rule-Matching Service for Simpler Develpment of Reactive Applications. *IEEE Distributed Systems Online*, 2(7), 2001.

[10] E. Katsiri. Middleware Support for Modelling Context-Awareness in Sensor-Driven Systems. Technical Report UCAM-CL-TR-620, University of Cambridge, 2005.

[11] E. Katsiri, J. Bacon, and A. Mycroft. An Extended Publish/Subscribe Protocol for Transparent Submissions to Distributed Abstract State in Sensor-Driven Systems Using Abstract Events. In *International Workshop on Distributed Event-Based Systems (DEBS '04)*, pages 68–73, Edinburgh,UK, 2004.

[12] E. Katsiri and A. Mycroft. Knowledge-Representation and Abstract Reasoning for Sentient Computing. In *Proceedings of 1st Workshop on Challenges and Novel Applications of Automated Reasoning, in conjunction with CADE-19*, pages 73–82, Miami Beach, FL, Jul.–Aug. 2003.

[13] D. Lopez de Ipiña. TRIP: A Distributed Vision-Based Sensor System. Technical report, University of Cambridge, 1999.

[14] A. L. Murphy, G. P. Picco, and G.-C. Roman. Developing Mobile Computing Applications with Lime. In *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, pages 766–769, Limerick, Ireland, June 2002. ACM Press.

[15] A. Nonnengart, G. Rock, and C. Weidenbach. On Generating Small Clause Normal Forms. In *CADE-15: Proceedings of the 15th International Conference on Automated Deduction*, pages 397–411, Lindau, Germany, July 1998. Springer-Verlag.

[16] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *MobiCom '00: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 32–43, Boston, MA, Aug. 2000. ACM Press.

[17] QoSDREAM: Quality of Service for Distributed REconfigurable Adaptive Multimedia. http://www-lce.eng.cam.ac.uk/qosdream/.

[18] B. N. Schilit, N. Adams, R. Gold, M. Tso, and R. Want. The PARCTAB Mobile Computing System. Technical Report CSL-93-20, Xerox PARC, Oct. 1993.

[19] D. Scott, A. Beresford, and A. Mycroft. Spatial Security Policies for Mobile Agents in a Sentient Computing Environment. In *Proceedings of the IEEE 4th International Workshop for Policies for Distributed Systems and Networks*, pages 147–157, Lake Como, Italy, June 2003. IEEE Computer Society Press.

[20] SPASS. An Automated Theorem Prover for First-Order Logic with Equality. http://spass.mpi-sb.mpg.de/index.html.

[21] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transaction on Information Systems*, 10(1):91–102, 1992.

[22] C. Weidenbach. *The Theory of SPASS version 2.0*. Max-Planck-Institut fur Informatik.