University of Strathclyde

Department of Computer
and Information Sciences

# A Model and Architecture for Pervasive Situation Determination

by

Graham R. Thomson

A thesis presented in fulfilment of the requirements for the degree of

Doctor of Philosophy

2010


# Copyright Notice



Signed:

Date:

# Acknowledgements

I would like to thank all those at the University, and in the SmartLab in particular, who made my time there such a pleasure. I would also like to thank my supervisor, Sotirios, whose tireless enthusiasm and unceasing inspiration made this a fascinating and enriching experience. Furthermore, I must thank my family, for their unwavering support and encouragement. Finally, I would like to thank my wife, Emma, whose patience and kindness are without bound.

# Publications

1. "A Model and Architecture for Situation Determination", G. R. Thomson, S. Terzis and P. Nixon. *The 16th Annual International Conference on Computer Science and Software Engineering (CASCON).* October 2006.

2. "Situation Determination with Reusable Situation Specifications", G. R. Thomson, S. Terzis and P. Nixon. *Fourth Annual IEEE International Conference on Pervasive Computer and Communications (PerCom).* Pisa, Italy. March 2006. pp. 620-623.

3. "A Self-Managing Infrastructure for Ad-hoc Situation Determination", G. R. Thomson, G. Stevenson, S. Terzis and P. Nixon. *Smart Homes and Betond ICOST2006. Assistive Technologies Series.* Amsterdam, The Netherlands. June 2006. pp. 157-164.

4. "Situation Determination with Distributed Context Histories", G. R. Thomson, P. Nixon and S. Terzis. *1st International Workshop on Exploiting Context Histories in Smart Environments. 3rd International Conference on Pervasive Computing - Pervasive 2005.* Munich, Germany. May 2005.

5. "Towards Ad-hoc Situation Determination", G. R. Thomson, P. Nixon and S. Terzis. *First International Workshop on Advanced Context Modelling, Reasoning And Management. UbiComp 2004, The Sixth International Conference on Ubiquitous Computing.* Nottingham, England. September 2004.

6. "Towards Dynamic Context Discovery and Composition", G. R. Thomson, S. Terzis and P. Nixon. *1st UK-UbiNet Workshop.* London, England. September 2003.

7. "An Approach to Dynamic Context Discovery and Composition", G. R. Thomson, M. Richmond, S. Terzis and P. Nixon. *Proceedings of UbiSys '03, System Support for Ubiquitous Computing Workshop. UbiComp 2003, The Fifth Annual Conference on Ubiquitous Computing.* Seattle, Washington, USA. October 2003.

# Abstract

A situation determination system, at a simplistic level, detects the situations a user is interested in and reports them to situation-aware applications. These applications will react to the situations in ways the user has chosen, in order to achieve a desired set of outcomes. Ideally, such a system would be pervasive, providing support to users through the many situations and environments they encounter or are interested in, in their individual, day-to-day lives.

To realise the benefits of pervasive situation determination, this thesis argues that the system must meet a number of key requirements. These include support for end-user customisation of situations, rich situation models, adaptable recognition and inter-environment operation. This thesis addresses the problem of designing a system that can effectively meet these requirements.

The main outcomes of the resulting approach are an original model based on the critical idea of separating the description of the features of a situation from the specification of how to recognise it, and a novel software architecture that fully supports this model. Both the model and architecture are evaluated through the development of an extensive number of situations, specifications and customisations, spanning a range of domains and environments, and a suite of situation-aware applications. Not only did this provide several corroborative examples of the observation that customisations form environment-specific variations of more general situations, but also showed that the applications developed either relied upon, or could provide greater utility as a direct consequence of, the improved level of cover, precision, availability and reach that the approach offers. Furthermore, algorithmic analysis and performance measurements of a prototype instantiation confirm that its performance is sufficient for practical use. Overall, the evaluation shows that the approach presented in this thesis aptly supports pervasive situation determination.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Pervasive situation determination

A situation determination system, at a simplistic level, uses underlying sensor data gathered from the environment to detect situations a user is interested in and reports them to situation-aware applications. These applications will react to the situations in ways the user has chosen, in order to achieve a desired set of outcomes. For example, a user may wish to have their mobile phone present the situation a particular friend is currently involved in, before deciding whether to call their friend or to send a text message. In other occasions, the user may wish that their mobile phone automatically switch itself into a silent mode of operation upon detecting that the user is currently in a meeting. Ideally, such a system would be pervasive, providing support to users through the many situations and environments they encounter or are interested in, in their individual, day-to-day lives.

Earlier projects in the literature have focussed on detecting and exploiting individual pieces of information that characterize some aspect of a situation, such as the user's current location. These pieces have been termed 'context information'. The projects explored different ways of incorporating context information into applications and the avenues that this opened up for the creation of novel applications [1, 2, 3, 4].

Examples of these context-aware applications include a call-forwarding application that detected a user's identity and location to enable incoming land-line calls to be forwarded to the room the call's recipient currently occupied [5], Stick-e

Notes [6], which was a software post-it note application that displayed a note only when triggered by a particular piece of context information such as a user entering a particular location, and Dummbo [1], an application which used a badge reader to detect that a number of people were gathered around a whiteboard and automatically began recording the audio and drawings of the whiteboard session.

However, as the scope and expectation of these applications has grown, researchers have begun to realise that low-level context information is difficult to utilise within applications [1, 7, 8, 9, 10, 11]. Instead, a higher-level abstraction, namely a situation, seems to be more appropriate for use within applications as it provides a more natural point with which to associate an application's behaviours, giving rise to situation-aware applications.

Examples of more recent projects that focus on using the higher-level situation or activity as a basic computational unit include a system which monitors whether an elderly person can sufficiently perform household and personal hygiene activities by analysing their interactions with household objects through a miniature RFID tag reader worn as a glove [12]. Another is a situation-aware system that recognises different phases of product construction that a carpenter is performing by listening to the sounds that are being made by the tools they are using [13]. There is also a system that assists motorcar production line quality assurance inspectors by detecting which checkpoints of the quality inspection procedure have been carried out using data received from special suits replete with motion detectors that are worn by the inspectors [14].

Situation determination has become a key requirement for infrastructure supporting pervasive computing. Current approaches to situation determination however, like these examples, are commonly focused on a particular application area or a specialised environment. For a situation determination system to be pervasive, it must be capable of widespread operation, incorporating a number of users, applications and environments. But at the same time, deploying a network of similar environments in a "one-size fits all" approach is not desirable, as the system is of greater value to its users the more closely it can match their individual needs.

For example, the set of situations an individual is interested in will not be fixed. The situations a user is involved in, and so will want the system to recognise, will evolve over time. Additionally, this set may include distinctive situations that are particular to that individual user or their specific environment. The system must be able to include these new and custom situations.

Similarly, the environment will be dynamic, where the people and devices within it continually change. As certain devices enter the environment, the available sensing infrastructure is extended. Here, the system has the opportunity to recognise both additional situations and a finer level of detail of current situations, using these new sensors. Similarly, the system must strive to preserve the recognition of currently active situations when devices and sensors leave the environment, are switched off, or fail.

Furthermore, the system must be able to operate throughout a network of different environments. Not only may a user move between environments, but also the situations that the user requires to be recognised may involve the user him/herself in the local environment, or may involve a friend or colleague who is located in an external environment. In the latter case, the system requires some means of discovering situations that are occurring elsewhere in the network.

The benefits that such a pervasive situation determination system offers include increased cover of the situations the user is interested in, finer detail in the situations that are recognised, greater precision in stating how applications react and what the desired outcomes are, and also a higher level of availability and reach of both the system and users' applications. The work set out in this thesis aims to provide these benefits of a pervasive situation determination approach.

## 1.2 Problem Statement

To effectively realise the benefits of pervasive situation determination, there are a number of requirements that a supporting system must satisfy.

As noted above, the evolving set of situations the user may wish to have recognised by the system will be drawn from the user's individual, day-to-day life. The set will include both common, general situations and distinctive, custom situations that are particular to that individual user or their specific environment.

Herein however, exists a tension, in that it is the end-user who is aware of the special characteristics of the desired custom situations, but specifying how a high-level situation is detected from low-level sensor data is a difficult task. To create this mapping requires skill, as well as intimate knowledge of the situations that occur within the environment, the available sensing infrastructure, and the internal operation of the situation recognition process. Some approaches have addressed this tension by limiting the scope to a fixed set of situations recognised by a fixed, specific sensing infrastructure, and investing in specialists to create the

required mappings. Other approaches have attempted to have the system itself create these mappings by incorporating learning into the environment, but it is difficult to fully automate such learning, and attempts to do so have focussed on very simple situations. While semi-automated learning is also possible, it still requires a skilled administrator to conduct a training period in which the situations are learned, during which several examples of each situation would have to be collected and analysed before being used. These factors impede swift adaptation to the evolving set of situations that will occur in an environment over time. To address this tension, the system is required to facilitate customisation of situations by end-users themselves, empowering them to customise situations personally and immediately to suit their own individual needs and preferences.

Additionally, the user will have a set of desired outcomes that they will want to have occur in reaction to the situations they are interested in. For some outcomes, the user may require application responses that simply react to the occurrence of a situation, like in the case above of silencing a mobile phone when a user is in a meeting. For other outcomes, the user will require responses that react to more subtle elements of the details of a situation, such as the specific role the user is playing within it. For example, when the user is speaking in a presentation, they may wish that the laptop they are using to present their slides automatically disables its screen saver and power saving modes. If they were part of the audience, they may want their laptop to ready the notes they had made from previous presentations in the series. Therefore within the system, it should be possible to represent and react to the features and details of a situation, including the specific roles that people and devices may play, to allow both application responses and the resulting outcomes to be precisely defined.

As a user moves between environments, it may not be possible to predict what sensing infrastructure will be available to detect situations. For example, in a work environment, there may be powerful, dedicated infrastructure available, while at a café, it may only be the ad hoc collection of mobile phones at the users' table that can be used. Similarly, within a single environment, the available sensing infrastructure will continually change as the people within it, and the devices they carry, come and go. Yet despite this, achieving the outcomes that the user desires will depend upon their associated system responses being successfully performed. To address this, the system must strive to make the capability to enact these responses available, even though the target environment and precise configuration of the available sensing infrastructure may not be known initially,

and may change throughout the duration of the situation. This requires that the recognition process must be able to dynamically adapt to the available sensing infrastructure.

Furthermore, the system is required to support large-scale, inter-environment operation, as the situations the user is interested in may extend beyond the local environment. For example, in the scenario where the user wishes their mobile phone to present the situations a particular friend is currently involved in, their friend could be in the same building or in another part of the country.

A pervasive situation determination system must address these two dimensions of variability in combination. It must handle the variation in the set of situations that are recognised, from common, general situations, to the specific, custom situations of a particular environment. Simultaneously, it must handle the variation in how the situations are recognised, by transparently adapting to a dynamically changing sensing infrastructure and/or environment.

The extent to which these requirements can be realised with current state-of-the-art approaches is limited. Both modelling and infrastructure support are lacking. The modelling aspects for which current approaches fail to provide support for include:

**Customised situations** - The situations that are recognised are typically general, created externally, and do not capture the distinctive features of situations that are particular to the individual user or their environment. None offer the capability for end-users themselves to create their own customised situations. It should be possible that customisation can be performed by end-users themselves, as this provides greater cover of the situations that the user is interested in without relying on specialist maintenance to achieve it.

**Rich situation models** - Representations of a situation are commonly coarse-grained, and lack details or a visible internal structure that can be exploited by situation-aware applications. Greater detail and specific roles should be captured in the situation to allow the desired system responses and outcomes to be more precisely defined and more closely matched to the user's needs.

**Alternative descriptions** - A description of how to recognise a situation generally relies upon particular sensing infrastructure being available and so cannot be used in its absence. To be more pervasive, the model of a situation should include multiple, alternative descriptions that specify how the situation can be recognised from a variety of different forms of sensing infrastructure.

**Multiple viewpoints** - Typically, a situation focuses exclusively on the view-

point of the single, local, current user of a situation-aware application. While doing so makes the system simpler, it also significantly limits how pervasive the system can be. The system should be capable of reporting the situations of any person, device or location to any user in the system.

The aspects of infrastructure support that are currently lacking include:

**Adaptable recognition** - In addition to the model being able to provide alternative descriptions of a particular situation, the infrastructure should be able to provide adaptable recognition for the process as a whole, incorporating new descriptions and sensing infrastructure as they appear in the environment, as well as adapting to the loss of existing sources. Enabling the system to fully exploit the dynamic environment in which it operates can result in greater availability of both the system and users' situation-aware applications.

**Resource management** - Recognising a situation can be a computationally expensive task, yet it often must be performed by resource-constrained devices. To mitigate the cost of this, the system should strive to manage the resources within it, shifting expensive tasks to the devices that can best afford them.

**Inter-environment operation** - None of the current approaches fully support inter-environment operation. This restricts the reach and scope of situation-aware applications as the situations a user is interested in may extend beyond the local environment.

**Situation discovery** - When the situations a user is interested in do extend beyond the local environment, the system must have a means of discovering situations that are occurring elsewhere within the network of environments. Currently no approach supports situation discovery.

This thesis addresses the problem of designing a model and a supporting infrastructure that can effectively meet these requirements and realise the benefits of pervasive situation determination.

## 1.3 Solution

The solution is comprised of two main parts. The first part is a model that provides a means of specifying situations, their customisations, and the mappings from low-level sensor data that recognise them. This part addresses the modelling goals identified above. The second part is an architecture that describes a process of recognising the situations and customisations, and addresses each of the infrastructure goals.

The modelling solution presented in this thesis is based on the observation that many customised situations bespoke to a particular environment may not be entirely distinct from each other, but rather form environment-specific variations of a more general situation. For example, many Universities will run lectures, laboratories and tutorials. However, the students and staff of a particular University will be interested in the specific lectures, laboratories and tutorials that take place there. The structure of these general situations and the information required by a specification of how to recognise them, are likely to be similar or follow a set of similar patterns. For example, a lecture would normally have the lecturer and the students of the same course present in a lecture hall. The variations at a particular University are likely to differ in the details of the situations, such as the specific lecturer or course, or the specific location or time.

The difficult part of correlating particular sensor data to a general situation can then be performed by a skilled administrator or an external source, and the result can be reused in multiple environments. The general situation can then be customised to recognise the bespoke variations that occur in a particular environment. Then, given that creating customisations is simple enough that it can be performed successfully by end-users themselves, the approach allows a large variety of customised situations to be recognised by the situation determination infrastructure but does not incur the penalty of requiring specialist maintenance.

Critical to achieving the identified goals is the novel idea of separating the description of the features of a situation from the specification of how to recognise the situation. The features provide common conceptual structures for situations. Then, low-level properties of the available sensing infrastructure can be mapped to these conceptual structures by skilled parties, while end-users can base their customisations on these structures in conceptual terms meaningful to them. This allows the features of a situation to be recognised in a number of different ways, by many different types of sensing infrastructure. As the end-user customisations are based on the features themselves and not their mappings, both the customisations and sensing infrastructure mappings applied to a situation can vary independently. Situations can then be defined with a rich set of features, and multiple mappings can be created that recognise as many of the features as possible given certain sensing infrastructure. Also, by incorporating several alternative mappings, recognition of users' customisations and situations can be maintained as the available sensing infrastructure changes due to a shift in the composition of the current environment or to the user moving to a different en-

vironment. Moreover, the user's situation-aware applications are based upon the features of situations, and are therefore oblivious to whether situations are detected using mappings from the local environment or from a remote, external environment. This key idea enables the approach to simultaneously address the goals of improved cover, precision, availability and reach.

The part of the solution that addresses the architectural goals involves viewing the system as a set of cooperating agents, providing an effective means to realise them. Adaptive recognition benefits from loosely coupled communication between agents and dynamic selection of information, resource management can be realised through cooperative recognition, and inter-environment operation is achieved through common interaction protocols in both local and remote environments.

Discovering a situation within a network of environments is a difficult task. Situations may have complex representations and the set of situations that are occurring will be large and constantly evolving. This creates a lot of information that the discovery process must cope with, and it may quickly become stale. The solution presented here addresses this by transforming the problem into a much simpler one. The transformation is afforded by a novel aspect of the model. As a situation can be considered from a set of explicit viewpoints, rather than search for an occurrence of a situation, the system has only to search for the person, device or location that takes the desired view of the situation. For example, when a user is interested in the situations of a colleague, the system can first discover the location of the colleague and then request their situations from the local environment. Realising this simpler task can then leverage existing discovery algorithms to create an effective, large-scale solution.

## 1.4 Contributions

The main contributions of the work presented in this thesis lie in the following three areas:

- The presentation of an original model that provides support for the goals of customised situations, rich situation models, alternative descriptions and multiple viewpoints, which are not addressed by current approaches.
- The development of a novel software architecture that fully supports the model while additionally addressing the architectural goals of adaptable

recognition, resource management, inter-environment operation and situation discovery, which are currently lacking in existing approaches.

- The evaluation of both the model and architecture through the development of an extensive number of situations, specifications, and customisations, spanning a range of domains and environments, and a suite of situation-aware applications.

The following chapters present the work that supports these contributions, and detail the novel ideas and techniques that were critical to achieving them.

## 1.5 Thesis structure

The proceeding chapters of this thesis are as follows:

Chapter 2 presents a review of related work. It includes several context-aware and situation-aware projects, along with discussion of the ways in which these existing approaches are limited in their support for pervasive situation determination.

Chapter 3 describes the situation modelling approach introduced in this work. Several distinct aspects of the approach are presented. These include the way in which the model represents a situation as an independent, reusable entity, where situation descriptions and their mappings from low-level sensor data exist separately from situation-aware applications. Also presented is how rich situation descriptions can be created, involving ad hoc groups of people and devices, and how the precise role a person or artefact is playing within the situation can be denoted. Also shown is how it is straightforward to incorporate measures of uncertainty not only about the information used to derive a situation, but also about the resulting situation itself. Additionally, this chapter presents the critical idea of separating a situation's specification from the definition of its features, and demonstrates how this is used to support end-user customisations which can be recognised over multiple types of, and possibly dynamically changing, sensing infrastructure.

The accompanying software architecture to this model is the focus of Chapter 4. Here, the design of the situation recognition process is given, as well as details of the interaction of the component parts of the system, including the sensing infrastructure, the situation determination middleware, and situation-aware

applications. It illustrates several powerful features that the architecture offers. These include simultaneously recognising multiple specifications for a given situation, allowing the recognition process not only to adapt to the changing available sensing infrastructure, but also to fuse the results of multiple specifications together to give a higher overall confidence. Also included are how the architecture automatically detects and switches between infrastructure-based and infrastructure-free styles of operation, and how it enables inter-environment situation determination.

Chapter 5 provides an evaluation of the situation modelling approach and a prototype middleware implementation of its accompanying situation determination architecture. The approach is assessed through the construction and use of several example situation-aware applications as well as the development of a significant library of situations, specifications and customisations. Each application is distinct in style, and together they fully exploit the middleware. The library of situations spans a range of domains and environments. Furthermore, a detailed performance analysis is also included in this chapter, which verifies that the middleware provides adequate performance at realistic deployment sizes.

A summary of the work and the contributions of this thesis is given in Chapter 6, as well as a brief look at potential extensions to the situation determination middleware and other possible future work.

# Chapter 2

# Related Work

## 2.1 Introduction

This chapter examines a number of existing works drawn from the literature, with the aim of identifying the level of support that is currently available for the requirements identified in the previous chapter as necessary to realise pervasive situation determination.

These requirements include both modelling and architectural aspects. The modelling aspect requires support for:

**Customised situations** - An approach must offer the capability for end-users themselves to create their own customised situations that capture the distinctive features of situations that are particular to the individual user or their environment.

**Rich situation models** - The system's representation of a situation should provide a rich, fine-grained model of a situation. It should be possible to represent the structure of the situation, such as the specific roles that are played within it as well as the relationships between them. Furthermore, as the sensor data that will be used to detect the elements of a situation may suffer problems of accuracy or staleness that affect the confidence the system can have in that data, the representation must provide a convenient means of incorporating and handling measures of confidence.

**Alternative descriptions** - To allow pervasive recognition, the model of a situation must include multiple, alternative descriptions that specify how the situation can be recognised from a variety of different forms of sensing infrastructure

in a number of different environments.

**Multiple viewpoints** - To support pervasive reporting, the system should be capable of reporting the situations of any person, device or location to any user in the system. Therefore, the representation must support multiple viewpoints of a situation, including those from each of the people, devices and locations that are involved within the situation.

In attempting to model a situation, there are, at a high level, two distinct styles of approach that can be taken. The first is to have the model of the situation explicitly designed by one or more of the people who have some stake in being able to recognise the situation. The other style is to have the system itself attempt to infer the model.

Moreover, approaches in which the model is explicitly designed can be generally grouped into one of two broad categories. The first category includes those approaches that adopt a 'top-down' view, where the approach explores the best ways to model situations and high-level context information, and then how these relate to methods of detection and sensing infrastructure. These approaches focus on the representation of situations. The second category includes those approaches that take a 'bottom-up' view, in which the approach first identifies novel means of sensing some aspect of the environment and then expands upon the possibilities this creates for modelling the activity or situation that is occurring. These approaches focus on the recognition of situations.

Section 2.2 provides a comprehensive review of the most complete and pertinent examples of existing work from each of these styles and categories that relate to the modelling requirements listed above. Each of these styles and categories, and the support that currently exists for the modelling requirements, are explored across three sections.

First, in Section 2.2.1, approaches to representing a situation are presented, where the support offered for all four modelling requirements is considered. Next, in Section 2.2.2, approaches that explore novel methods of recognising a situation are reviewed. These projects give particular focus to practical systems for situation determination in real world settings, and the level of support for all four modelling requirements is again considered. Following this, approaches in which the system attempts to infer the model of a situation are presented in Section 2.2.3. While these are related to all four of the modelling requirements, additional focus is given here to the support that such approaches can offer for customised situations. Learning approaches potentially offer the possibility of

automating the customisation process. This section looks at the level of support that is possible from such approaches and their practical implications.

The architectural aspect requires support for:

**Adaptable recognition** - The architecture must be able to provide adaptable recognition for the process as a whole, incorporating new descriptions and sensing infrastructure as they appear in the environment, as well as adapting to the loss of existing sources of context information that are actively used to recognise situations.

**Resource management** - Recognising a situation can be a computationally expensive task, yet it often must be performed by resource-constrained devices. To mitigate the cost of this, the system should strive to manage the resources within it, shifting expensive tasks to the devices that can best afford them.

**Inter-environment operation** - The situations a user is interested in may extend beyond the local environment, and so too must the reach and scope of situation-aware applications. Therefore, the architecture must support inter-environment operation to enable such pervasive situation recognition.

**Situation discovery** - When the situations a user is interested in do extend beyond the local environment, the system must have a means of discovering situations that are occurring elsewhere within the network of environments. The architecture must provide the capability for situation discovery in order to support pervasive situation recognition.

A characteristic element of pervasive computing is the use of mobile, personal devices that assist the user in some way by providing services or information [15, 16, 2, 4]. This has led to the development of two distinct styles of architecture supporting pervasive computing.

Some approaches attempt to complement the typically limited resources of these mobile devices by providing additional infrastructure within the environment. Both the mobile and infrastructure devices are used in concert to provide user applications. This allows devices to be small enough that they can be embedded in clothing or other physical objects, but also allows the environment as a whole to be able to provide sophisticated services. This style is referred to as an infrastructure-based architecture.

Other approaches focus on scenarios where such supporting infrastructure is not available, and strive to provide the greatest level of support possible within the constraints of the host devices. In this style, it is solely the mobile devices that provide user applications. This allows greater flexibility in the types of

scenarios in which these approaches can be used. This style is referred to as an infrastructure-free architecture.

Pervasive situation determination demands support for both of these architectural styles, as well as fluid transition between them. Adaptation is crucial not only in supporting changes between infrastructure-based and infrastructure-free operational modes, but also in large-scale architectures for realising inter-environment operation and situation discovery.

Section 2.3 reviews the most complete and relevant examples of existing work that address these four different elements of infrastructure-based and infrastructure-free modes of operation, adaptation, and large-scale operation, and examines the level of support that is offered for the architectural requirements across four separate sections. Section 2.3.1 and Section 2.3.2 look at infrastructure-based and infrastructure-free architectures respectively, and the support they offer for situation-aware applications. All four architectural requirements are considered in both of these sections. Next, in Section 2.3.3, existing works that concentrate on methods of infrastructure-supported adaptation of context-information are presented in relation to their support of the adaptable situation recognition requirement. Finally, Section 2.3.4 looks at initial large-scale approaches and considers their support for inter-environment operation and situation discovery.

## 2.2 Modelling aspects

This section looks at the modelling aspects related to pervasive situation determination, reviewing existing approaches' support for end-user customisation, rich situation models, alternative situation descriptions and incorporating multiple viewpoints of a situation.

### 2.2.1 Representing situations

There are many different facets to representing a situation. These include what the basic elements of the model of the situation are, the way the model is structured, how the model is specified and the nature of the modelling process itself.

The literature features a number of different styles of representing a situation. This section provides a comprehensive review of these different styles, covering each of these facets, that pertain to pervasive situation determination. The approaches presented include the most complete examples of each style.

In several early context-awareness projects, situations were defined implicitly by the context-aware application developer who was required to state every specific piece of low-level context information that was required for each situation in each application. This implicit situation representation style, which has borne influence on what forms the basic elements of several later situation models, is presented in Section 2.2.1.1.

Following this, logic-based and ontology-based styles are discussed in Section 2.2.1.2 and Section 2.2.1.3 respectively. In the logic-based approaches, situations are constructed from logical predicates, whereas in the ontology-based approaches, ontology-defined concepts are used. These styles are not entirely dissimilar, and ontological elements may feature in a logic-based approach and vice-versa. These form explicit attempts at defining the basic elements, the structure, and how to specify a situation model.

A graphical representation style is then presented in Section 2.2.1.4, which focuses on means of identifying and specifying the context information required by an application pictorially. This work provides the most fully developed attempt at defining a situation modelling process.

In Section 2.2.1.5, a role-based style is presented in which the representation of a situation comprises a number of roles, as well as the entities that play these roles. This style demonstrates a very natural way to structure a situation.

Finally, location forms the most essential basic element of many situations, and Section 2.2.1.6 looks at approaches that focus on methods to represent location information.

#### 2.2.1.1 Implicit representation

In several approaches, a situation is implicitly represented as an aggregation of context information. One such approach is the much-referenced work of Daniel Salber, Anind Dey and Gregory Abowd [17], that laid the foundation on which a lot of later work on context-aware systems was built. This work includes a commonly-cited definition of context [18, 19]:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

They also contend that context-aware applications are concerned with the *who*, *what* (that is, what the user is doing), *when* and *where* of different entities and use this information to determine *why* the situation is occurring. The *why* was not actually determined by an application, but rather this was encoded by the designer of the application. Identity, activity, time and location are proposed as the primary context types for characterising the situation of a particular entity, as they relate to the who, what, when and where. A 'context widget' is used to provide an application with each of these context types in a generic manner, providing a layer of abstraction over how they are actually sensed.

An 'interpreter' widget is used to handle uncertain context information. The particular implementation of an interpreter may vary, depending upon the type of context information being processed. For example, an interpreter may perform simple filtering by rejecting any result whose confidence factor is below a given threshold, or it may consolidate results from multiple sources.

An interpreter provides a straightforward means of handling uncertainty associated with a particular piece of context information, providing a 'black box' that transforms uncertain context information into a definite result. When modelling a situation, other techniques may be more appropriate, as making the source context information definite on a per-piece basis may make it difficult to calculate a precise confidence factor for the overall situation.

The approach recognises activity as a primary context type in describing a situation. However, it offers no framework for explicitly describing nor automatically detecting the high-level situation or activity from other pieces of context. It is left up to the application developer to manually construct each situation their application requires.

Forcing developers to build their own implicit situation models is undesirable as the models are created in an ad hoc fashion and the effort to model similar situations must be repeated in each application. Furthermore, it lacks a common model of the same situation between different applications and similarly, there is no guarantee that a situation developed in one environment will be able to function in another. In addition, it places a heavier burden on the application developer, who must also put effort into modelling and capturing the situation, in addition to the work of designing and building the application itself.

These factors limit the level of support implicit representation can provide for the modelling requirements of pervasive situation determination. As altering the description of a situation requires skill in application programming, it becomes

too specialised to be performed by a typical end-user and essentially prohibits end-user customisation. As the developer must define their own model implicitly, it may be possible to build rich models and multiple viewpoints of a situation, but as noted above, as there is no structure or direct support for this, it must be created in an ad hoc fashion and requires repeated effort each time. The lack of a common model for the same situation inhibits the use of multiple, alternative specifications for adapting to the dynamically changing available sensing infrastructure within a pervasive computing environment.

The key benefits this approach does offer for pervasive situation determination are in using abstractions of context information that shield the complexities of how that context information is detected, and in installing the notion of modelling a high-level situation as a composition of lower-level pieces of context information. These concepts have influenced several later works and should be exploited.

#### 2.2.1.2 Logic-based representation

The section presents two logic-based approaches. The first is the Gaia project, which is the most comprehensive existing approach for specifying higher-level context information, providing an extensive range of mechanisms and features that could be employed to describe a situation. The second approach is the work of Anagnostopoulos et al. which is distinct among current approaches in that it not only attempts to model situations explicitly, but also in providing a detailed scheme for combining measures of uncertainty associated with an instance of a situation. Discussion of the support both lend to the modelling requirements for pervasive situation determination follow the presentation of the individual approaches.

##### The Gaia model

The Gaia project has developed middleware that aims to provide support for transforming physical spaces into programmable entities [20]. Work within the Gaia project has proposed a model for context based upon first-order logic [21]. In this model, context information is represented as first-order predicates, for example:

Location(Chris, Entering, Room 3231)
Temperature(Room 3231, =, 98 F)

Here, the name of the predicate is the type of context that is being described, such as location or temperature. The authors aim to use this as a simple, uniform representation for different kinds of context. Context-types are defined in an ontology, and type checking of predicates is performed against this ontology to ensure that a predicate makes sense.

More complex context expressions can be constructed by using Boolean operators such as conjunction, disjunction and negation over context predicates. Both universal and existential quantification over variables are permitted. Quantification is performed over the domain of all of the values of a variable the system is currently aware of, that is, all of the instances described in the ontology. For example, the set of all people consists of the names of all the people known to the system. Note that this becomes unmanageable as the system expands from operating in a single environment to a large, interconnected network of environments.

The model permits rules to be defined that provide the ability to deduce new context information based on existing context information. For example, the following rule infers that there is a party going on in a particular room if the level of sound in the room is high, stroboscopic lights are on and the number of people in the room is greater than the specified threshold value:

$$\begin{aligned}&\text{Sound(Room 3234, } >\text{, 40 dB)} \wedge \\ &\text{Lighting(Room 3234, Stroboscopic)} \wedge \\ &\text{NumberOfPeople(Room 3234, } >\text{, 6)} \\ &\rightarrow \text{SocialActivity(Room 3234, Party)}\end{aligned}$$

The rule mechanism can be used to combine context provided by context predicates defined in, or sensed by, the system, as well as inferred context from other rules.

Later extensions to this provide support for handling uncertain context information [22], and provides support for probabilistic logic, fuzzy logic and Bayesian networks. This provides considerable flexibility to a situation modeller as it offers a variety of options for handling uncertainty. However, it is not clear if it is possible for pieces of context information that employ different methods of handling uncertainty to be combined into the same situation description, or if, for example, a situation includes a piece of context information that is modelled as a Bayesian network, then the whole situation must be modelled as a Bayesian network. The inclusion of multiple methods of handling uncertainty also makes the approach

more complex, as modelling a situation or working with situation descriptions requires understanding the predicate-based first-order logic, probabilistic logic, fuzzy logic, as well as the Bayesian network representations.

### Anagnostopoulos et al.

Anagnostopoulos et al. presented an approach which supports situation representation and reasoning [11]. The approach permits building up situations as logically aggregated pieces of context information.

A situation is recognised by a rule that has the following form:

$$\bigwedge_{i=1}^{n} \text{context}(x_i, \text{user}) \rightarrow \text{isInvolvedIn}(\text{situation}, \text{user}),\ n > 0$$

A situation recognition rule such as this denotes that an isInvolvedIn predicate is implied by the logical conjunction of the context predicates. When each of the context predicates hold, it is implied that the user is involved in a certain type of situation.

Application actions can then be specified as:

$$\text{isInvolvedIn}(\text{situation}, \text{user}) \wedge \text{specified}(\text{situation}, \text{user}, \text{task}) \rightarrow \text{do}(\text{execute}(\text{task}, \text{option}), \text{situation})$$

An application action rule such as this denotes that if a user is involved in a situation and has specified in an associated profile a task to be executed, then the system should invoke that specific task.

The scheme this approach provides for estimating the uncertainty of a user's involvement in a situation is based on a complex model that uses a pre-defined set of exemplar situation descriptions and calculates the confidence that a user is involved in a situation as its distance from these exemplars. The distance from an exemplar is based on full, fuzzy logic-based, structural analyses of the current situation to be recognised and each of the exemplars. As the scheme is complicated, its inclusion makes the overall situation model more difficult to understand. Furthermore, the repeated structural comparisons and process of defuzzification [23] upon which they rely may become computationally expensive as the number and size of the situation specifications in use becomes large. More critically however, the scheme is not appropriate in the light of situation customisation. As the customisation will add additional constraints to its base situation, it will move farther away from its exemplar. The scheme would then report that

the user is involved in the customised situation at a lower confidence, even if all of the context information it was recognised from was reported at full confidence.

### Discussion

A straightforward way in which both approaches could potentially support end-user customisation would be to allow a situation predicate to be included in the left hand side of a customisation rule along with the additional constraints of the customisation. However, this severely limits the scope of the customisation, as the entities referenced by the situation rule will not be available to the customisation. To expand the scope, the situation rule would have to be rewritten to include the customisation's constraints directly. However, this then demands that the customisation author is familiar with the details of the situation rule that they are expanding. This is a burden which is made even greater when multiple, alternative rules for the situation exist. Attempting to support customisation in this manner heads towards specialist situation specification authoring and is not suitable for use by end-users.

It is in representing rich situation models that the Anagnostopoulos et al. approach offers its best support. The approach provides a common, general structure for a situation, and includes a scheme for estimating the uncertainty of a user's involvement in a situation based on uncertain context information. In this regard however, the Gaia model is limited in similar ways to the Dey approach presented above. This includes the lack of a standard situation structure, which places a greater burden on the situation-aware application developer as they must create their own ad hoc models each time.

Whilst both approaches make it easy to build up logic rules to infer higher-level context information by combining lower-level context using the standard logical operators, a greater level of expressiveness is required for describing rich situation models. In particular, the ability to access, count and perform arithmetic on the entities and properties referenced within a situation is necessary to capture concisely the dynamic and variable structure of a situation. For example, a 'group meeting' situation defined as a meeting in which 75% or more of the attendees belong to the same group, is difficult to express in a natural and general way using only logical operators. Furthermore, both models represent only whether a user is involved in a situation or not, and do not capture the specific roles the user or other people and entities may be playing within the situation.

It would be possible to provide alternative descriptions for a situation in both of these approaches by defining multiple rules that lead to the same conclusion. To use the Gaia example, a specification author could write several rules each with varying conditions on the left hand side of the implication, but which all result in the same "SocialActivity(Room 3234, Party)" situation predicate on the right hand side.

However, it would not be as simple to express multiple viewpoints. In the Gaia approach, whilst the multiple viewpoints of each of the different people and entities involved in the situation could be represented by including several situation predicates on the right hand side of the implication, for example including "SocialActivity(John, GroupMeeting)" in addition to "SocialActivity(Room 266, GroupMeeting)", the required set of conditions on the left hand side may become very complex. Moreover, note that situation predicates like this make reference to the class of situation "GroupMeeting" and not a particular instance of a situation. It is then difficult to relate entities and situation predicates that are associated with the same situation. For example, if there were also to be a situation predicate that represented whether John was one of the group member attendees, there is no direct way to relate this to the predicate that more generally states that John is involved in a group meeting or to the predicate stating there is a group meeting occurring in Room 266.

In the approach of Anagnostopoulos et al., expressing multiple viewpoints would also be difficult as the model is specifically grounded in the single viewpoint of the current user in the local environment. It is possible that a location or a device could be the focus of a situation rule in addition to a user, for example the context predicate could appear as 'context($x_i$, location)' or 'context($x_i$, device)' instead of 'context($x_i$, user)', but each situation rule may have to be rewritten in each case. Plus, this would also exhibit the problem of detecting whether reports of a situation from different viewpoints refer to the same instance of a situation or not.

There are several key advantages offered by a logic-based approach. These include the simplicity and clarity in combining different pieces of context information to make a situation, in a way that is familiar to programmers and is readily understandable. Furthermore, mature techniques and tool support are available for modelling, reasoning about and processing logic-based systems.

Both the Anagnostopoulos and the Gaia approaches support fuzzy logic for handling measures of uncertainty about context information. Though the spe-

cific schemes they use are complex, it is possible to derive a simpler and lighter fuzzy logic-based scheme that can incorporate and combine the certainty factors of the compositions of context information that comprise a situation. A final key advantage the Anagnostopoulos approach offers in supporting pervasive situation determination is that it provides an explicit, common, basic structure for representing a situation. In conclusion, a logic-based approach, with suitable extensions, can be an appealing way to model situations.

#### 2.2.1.3 Ontology-based representation

Several efforts have concentrated on attempting to provide a common ontology for pervasive environments and context-aware applications. In these approaches, context information is represented as documents containing instances of concepts that are defined in a standard, shared ontology.

One such effort is the COBRA-ONT ontology, a collection of ontologies for describing places, agents, events and their associated properties in an intelligent meeting room domain [24]. Related to this is the SOUPA ontology, which has been designed to model and support pervasive computing applications and includes modular component vocabularies to represent intelligent agents with associated beliefs, desires and intentions, as well as time, space, events, user profiles, actions and policies for security and privacy [25]. Together, these form the most comprehensive ontology project for pervasive computing.

The CONON project [26] provides an upper context ontology that captures general concepts about basic context, and also provides extensibility for adding domain-specific ontologies in a hierarchical manner. This project is distinct in that it attempts to provide explicit support for modelling an activity.

Activity and situation modelling are supported in an abstract manner, that is, an 'Activity' class is defined that has 'Scheduled Activity' and 'Deduced Activity' subclasses. A Scheduled Activity represents an activity that is assumed to always occur at a known time, while a Deduced Activity represents an activity that must be detected dynamically from the available context information in the environment. Although a Deduced Activity concept is provided, it is defined abstractly - its structure is empty. The structure of any specific subclasses or instances must be defined by the application developer. This then shares the drawbacks of forcing an application developer to create his/her own implicit or ad hoc models of a situation with the logic-based approaches presented earlier,

specifically the lack of a common model of the same situation between different applications and environments, and the greater, perhaps redundant, effort that is required of the application developer.

The recently proposed Ontonym project [27] provides a modern, comprehensive and integrated set of ontologies that represent core concepts of pervasive computing. The ontology supports modelling context information as well as associated uncertainty, provenance, sensor capability and temporal properties. The ontology includes a broad range of concepts relating to location, people, time, events and sensing. The approach does not define structures for representing situations directly, though it is possible that the 'SpatioInstantEvent', 'SpatioIntervalEvent' and 'Role' classes drawn from the events ontology could provide suitable base concepts upon which general situation structures could be created.

The main achievement of these ontology-based approaches has been in facilitating the exchange of independent pieces of low-level context information through a common context model. However, as their focus has not been on composing pieces of context information into situations, the level of support for this exhibited by these ontology-based approaches is similar to that of the Salber et al. approach presented earlier. There, the author required skill in programming to compose pieces of code ('context widgets') to describe a situation. Here, the author must compose concepts from an ontology to create the description. Altering the description of a situation would require skill in ontological modelling and experience with related languages and tools, and is therefore unsuitable for use as a means to perform customisation by a typical end-user. Similarly, while it could be possible to create rich models, multiple viewpoints, and alternative descriptions, the situation author or application developer must build the support for this by him/herself in each case.

While neither the SOUPA nor CONON ontologies originally support representing the confidence factor of a situation, the open nature of the ontological-based approach allows for straightforward extension to include such a feature. However, as it is not included as part of the model per se, it may lead to inconsistent use across the different sources of context information used to recognise a situation.

The ways in which the ontology-based approaches support the requirements of pervasive situation determination lie in the notion of a shared, common representation of context information between systems and applications, and in representing context information in high-level terms that are meaningful to the end-user.

These are key concepts that can be used to address the inherent open nature of pervasive situation determination systems, and can assist in making situation models and end-user customisations easy to comprehend.

#### 2.2.1.4 Graphical representation

Karen Henricksen and Jadwiga Indulska produced a framework for modelling and building context-aware applications [8, 28, 29, 30]. They were motivated by the observation that recent research in the field of context-awareness had predominantly adopted an infrastructure-centred approach that had assumed that the complexity of engineering context-aware applications can be substantially reduced solely through the use of infrastructure capable of gathering, managing and disseminating context information to applications that require it. They argue that an infrastructure-centred view leads to abstractions for describing and programming with context that are not the most natural ones, observing that most of the proposed infrastructures are based on context models that are informal and lacking in expressive power.

A graphical context modelling approach was developed to provide a tool to assist context-aware application designers in exploring and specifying the context information requirements of a context-aware application. It provided modelling constructs to describe individual types of context information, called 'fact types', classifications of their sources, quality-related metadata, as well as dependencies and constraints amongst different types of context information.

The modelling approach is presented as an extension to Object-Role Modelling (ORM), which is a method for designing and querying database models at a conceptual level [31]. Basing the approach on ORM allows adoption of its greater formality and expressiveness in comparison to other database modelling techniques, as well as its mapping to the relational model, allowing a straightforward representation of a context model in terms of a relational database.

The approach includes a 'situation abstraction' for modelling higher-level, derived context information. However, this construct does not offer a general structure for describing and recognising high-level activities that is sought, but rather offers only a way to express rules within the model, similar to the mechanisms presented earlier in the Gaia approach. Accordingly, its support for pervasive situation determination is lacking in similar ways.

In order to accommodate uncertain context information, the model offers sup-

port for a three-valued logic. Assertions can either be true, false or possibly true, where possibly true denotes a value that is unknown or is ambiguous. Additionally, context may be annotated with its source, that is, whether it is a static fact, taken from a profile, derived from other context or sensed from the environment. This annotation may be used to infer different aspects of the quality of the information. The three-valued logic and source annotations provide a discrete means to incorporate uncertainty into their logic-based model of context. This provides a convenient and appropriate means to rapidly create high-level context requirements and identify sources of uncertainty. However, to translate this into an operational model that can be used to recognise a situation, this approach would be lacking in that it does not provide a mapping to probability-based uncertainty measures commonly found in a pervasive environment. Several location systems for example, report locations with a probability of error [32, 33, 34]. This forces the context-aware application developer to make arbitrary cut-offs for true, possibly true and false, which could lead to inconsistent behaviour across applications.

Also, note that although an activity does appear within this model as a fact type, it is modelled as a user-supplied data type, it is not inferred. Its representation is simply a string label denoting the activity, say gleaned from a calendar or personal organiser application, and has no internal structure. However, their approach focuses on providing a framework to help application designers establish the requirements for a fixed set of independent pieces of context information required by a context-aware application they wish to build, rather than modelling activities and relationships between groups of people and devices. Modelling an activity as a string label simplifies the process, and so is appropriate in their case. In a deployed system, the label could be inferred by using a pervasive situation determination system such as that described in this thesis.

A distinct advantage that Henricksen's approach offers in supporting pervasive situation determination, is being able to graphically partition the required context information into smaller, related subsets, making it more manageable to work with larger sets of context information. This idea can be built upon to provide a similar means of partitioning the elements of a situation, to provide a more manageable means of working with the large amounts of context information that will be involved in a complex situation description.

#### 2.2.1.5 Role-based representation

A high-level context representation was proposed by Crowley et al. [35, 36, 37]. The approach is uncommon not only in that it provides a well-defined structure for a situation, but also in that it provides some initial work on explicitly modelling distinct roles within a situation.

The representation involves a user perspective and a system perspective, where each perspective defines a specialised model of a situation.

The user's perspective concerns the user's task and activity. The context for a user and task is defined as a composition of situations, which all share the same set of roles and relations to be observed that are relevant to the task.

A role is defined as a potential set of actions within a task. The actions of a role may be enabled by certain entities whose properties meet a specified set of constraints. An example given is that an object may serve as a pointer if it is of a graspable size and appropriately elongated [35]. The system may assign entities to roles, tagging the assignment with a confidence factor. One or more roles may be played by an entity, and a role may be played by one or several entities.

In the user's perspective a situation is a particular assignment of entities to roles completed by a set of relations between the entities. A situation is viewed as the 'state' of the user with respect to his/her task. The system perspective extends this such that entities may include internal, observational processes of the system itself, and the relations may include the connections between processes.

In this model, a context is viewed as a network of situations defined in a common state space, where a change in the relation between entities, or a change in the assignment of entities to roles is represented as a change in situation.

This model defines a basic, general structure for a situation. It affords a richer level of detail by including explicit representation of different, abstract roles within a situation and the relations that hold between them.

When applied to pervasive situation determination, the Crowley approach provides a good level of support for creating rich models of situations. However, creating descriptions that recognise a situation demands a significant level of technical skill on the part of the author. Situation descriptions are composed from 'contextors', which are low-level processing units of individual pieces of context information and are chained together in a workflow that processes the recognition of the desired situation. The author is required to be familiar with signal processing techniques such as Bayesian estimation [38] and Kalman filters [39] to

address any uncertainty associated with a situation's source context information. Composing descriptions in this manner is akin to programming, and leaves little scope for end-user customisation.

Whilst the approach provides both a system and a user view of a situation, the system view is actually an extension of the user view that additionally includes representations of internal system processes. The approach does not fully support multiple viewpoints of a situation as is required for pervasive situation determination. A report of a situation would be produced by the final output contextor of the description's workflow and its type is fixed for a particular description. Therefore, to represent different viewpoints, it may be necessary to create alternative descriptions with a distinct output type, for example a location or device type instead of a person. This then may also suffer the same difficulties as presented earlier in this section of determining whether reports from multiple viewpoints refer to the same instance of a situation. Moreover, no direct support is given to representing alternative situation descriptions.

A key advantage of this approach, similar to the graphical approach presented above, is that a role provides a way to partition the potentially large amount of context information used to recognise a situation. While a role is given the functional definition of a set of potential actions here, it is possible to generalise the notion of a role such that it not only provides a means of abstracting over the detail of the person, device or location playing the role or the sensing infrastructure used to detect it, but also provides a very natural way to structure the characteristic high-level features of a situation in terms that are close to the user and the real-world entities involved in the situation.

#### 2.2.1.6 Location representation

Location information is commonly regarded as essential for describing situations [1]. Several location systems exist [40, 41, 42], and location is typically represented in either a geometric or symbolic co-ordinate based scheme. In geometric co-ordinate schemes, a location is represented by a single point, or an area or volume delimited by a set of points, within a multi-dimensional space. The Global Positioning System (GPS) is an example of a popular geometric co-ordinate based scheme [43]. In symbolic co-ordinate schemes, a location is defined by an abstract symbol. Street names are a common example of a symbolic co-ordinate based scheme.

In developing pervasive computing applications, some projects have found that it is necessary to extend beyond the common capabilities of symbolic schemes, and introduce higher-level information about a location. This section reviews three such approaches that provide different styles of extensions that may assist in modelling situations, where each forms the most fully developed example of that style.

The first of these is the NEXUS project, which has developed an extended class schema for describing location and spatial models [44]. The schema includes classes for representing different kinds of spaces and locations. For example, a space defined as a building may be further classified as a restaurant or a museum.

The Activity Zones project developed a system that allowed a model of the areas within a room to be annotated with labels of the activities that commonly occur within those areas for the particular environment [45]. These areas could be learnt semi-automatically, and the associated activity labels could be used to influence the behaviour of a context-aware application. Note that this approach by itself is not sufficient to detect situations. The labels act only as hints at which situations might occur within a particular area. Without further detection, it is not possible to determine which situations are actually occurring.

The Location Awareness Information Representation (LAIR) project permitted locations to be modelled in terms of geographical relationships between spaces [46]. In addition to modelling common properties such as the name of a particular space and which other spaces it is contained within, it is also possible to include a list of paths that the space falls on, as well as a list of other spaces that can be seen from within the space.

Concerning their support for rich situation models, what is missing from these approaches is the ability to denote fine-grained types for a particular space. For example, within a room intended to host presentations, there will commonly be a speaker area and audience area. In supporting this, the NEXUS schema is too general. As it associates an activity with a space, the Activity Zones approach is too coarse-grained. The focus of LAIR is on the relationships between locations, rather than on the types of the location itself.

Although it does not lend direct support to this itself, the recently proposed LOC8 location framework [47] allows a developer to incorporate additional semantics into the location model. This presents a straightforward means of introducing fine-grained location types into an existing location model of the environment.

Providing support for fine-grained location types can offer both greater con-

venience and generality in writing rich situation descriptions.

### 2.2.2 Recognising situations

This section looks at existing projects that focus on the process of recognising situations and considers their support for pervasive situation determination. The projects presented here were selected as they all have a strong real world focus, and are examples of some of the most fully developed practical attempts at situation determination from the current literature.

Three different styles of approach are explored in this section, covering projects that focus on using specialised hardware, others that focus on using specialised algorithms and others still that have developed application or domain specific capabilities for recognising situations. Each of these styles is reviewed in turn, examining the level of support offered for all four of the modelling requirements, including customised situations, rich situation models, alternative descriptions and multiple viewpoints.

#### 2.2.2.1 Novel hardware approaches

There are several projects that, rather than try to model and detect activities or situations in general, explore a limited set of activities or physical states that can be detected using novel hardware or a novel application of particular sensing infrastructure.

A number of projects have centred around detecting the user's physical state with regard to their state of motion. These states typically include sitting, standing, walking, running, climbing or descending stairs, cycling and riding an elevator. These works have centred around exploiting novel hardware worn on the body, and include the work of Lee and Mase [48] in which they use a belt-attached PDA connected to accelerometers, a digital compass, and an angular velocity sensor worn on the thigh. The work of Bao and Intille built on this in their study which involves accelerometers attached to the wrist, elbow, thigh and shin, and a comparison of the recognition accuracy of different classifier algorithms [49]. Lukowicz et al. demonstrate a system based on force sensors that are thin enough to be embedded in a wearer's clothes, that by detecting muscle activity, can infer the user's state of motion [50]. Similar results were achieved by Lester et al., though their project was based on using a single sensor board that could be embedded in a mobile device, such as a mobile phone, and also showed

that similar levels of accuracy for recognising several states of motion could be achieved irrespectively of whether the device hosting the sensor board was worn on the wrist, waist or shoulder [51].

Philipose et al. developed a special glove that mounted an RFID reader that could read RFID tags embedded in a number of household objects [12]. Using this set-up, they had limited success in detecting activities of the wearer such as holding a telephone receiver, adjusting a thermostat and washing dishes.

As well as detecting activities through body-worn sensors, there are also hardware projects that have experimented with attaching sensors to physical objects in the environment. The CapToolKit project has developed a system which allows capacitive sensors to be embedded into objects [52]. Using this toolkit, the authors were able to construct a table that could detect the position of peoples' hands above the table, and a kitchen cupboard enhanced such that it could detect which items had been taken out or put back in the cupboard. Nishida et al. proposed a system that is based on 3D ultrasonic tags [53]. By attaching the tags to a coffee cup, a stapler and a box of tissues, they could reliably detect when any of these items were being used. Patel et al. [54] developed a hardware sensor that detects the electrical noise on residential power lines created by electrical devices during operation and when being switched on or off. Machine learning techniques were used to recognize events such as turning on or off a particular light switch, a television set or an electric oven.

Though the projects presented in this section are concerned with activity recognition, the activities that are recognised are very specific, quite low-level, and intimately coupled to the specific hardware that recognises them. As such, they offer little scope for rich situation modelling, multiple specifications or viewpoints, and no support for customisation. The limits of the hardware strictly dictate which and how the activities are recognised.

Where these projects excel, is in recognising the kind of activities that could be categorised as 'middle-level' context information. That is, the resulting activity is at a higher level than the stream of raw context data that is used to detect it. For example, taking and returning items from and to a kitchen cupboard presents a higher-level activity than the signals firing from the switches and sensors fitted to the cupboard and items. While at the same time, the activity is still too finely grained to reveal the full, higher-level situation that is being performed. By considering this wider context, for example, by examining a number of items that have been taken out together and other information sensed from the kitchen,

the system may be able to establish different situations that are occurring, such as the user is baking a cake, or that they are preparing some drinks. The support that these approaches offer for pervasive situation determination is to provide such 'middle-level' context information, forming a specialised, component part of the overall, larger system.

#### 2.2.2.2 Novel algorithmic approaches

Other projects have focussed on novel algorithmic techniques to infer higher-level context from low-level sensors. These include the work of Lester et al., which developed a system that could accurately detect if two devices were being carried by the same person [55]. The system achieved this by applying a specially developed correlation algorithm to the devices' accelerometer data.

Korpipää et al. presented a system that recognises simple situations from a set of different types of sensors [56]. Data from accelerometers, light, temperature, humidity and skin conductivity sensors, as well as a microphone, were processed using algorithms from the recent MPEG-7 standard [57], and combined using a naive Bayes classifier. Using this system, they had limited success in identifying simple situations involving the device wearer, such as driving a car, listening to music, being engaged in conversation, and whether the wearer was indoors or outdoors.

Fishkin et al. reported a system that could detect the activity of multiple household objects being moved and rotated within the environment using RFID readers and tagged objects [58]. The approach is based on a specially developed object motion algorithm allowing standard protocols and RFID hardware to be used.

These software-focused activity recognition approaches share similar characteristics with the hardware-focussed projects presented in the previous section. They are specifically built to recognise particular activities, the activities are quite low-level, and they assume that particular, though more standard, sensing infrastructure is available. As such, they offer a similar level of support for pervasive situation determination. They are limited in facilitating rich situation models, multiple descriptions and viewpoints, or customised situations. However, they also share the same key advantage, in providing specialised sources of 'middle-level' context information that may be used by a pervasive situation determination system in recognising high-level situations.

#### 2.2.2.3 Application and domain specific approaches

There are several projects that have focussed on attempting to detect situations and activities within a particular domain or for a particular application. Such domains include medical, health, office and home settings, as well as different types of manufacturing.

The work of Vurgun et al. is a project that focuses on the medical domain, and developed a system that reminds elderly patients to take their medication [59]. The system would only remind the patient however, if certain simple, application-specific activities had not been detected, such as the patient has indicated that the medication had previously been taken within a specific time period, or if the patient is not at home.

Projects that have focussed on health and fitness include a system developed by Chang et al., that by analysing data collected from accelerometers worn on the wrists and waist, could recognise nine different weight-lifting activities, which could then be used in a digital personal trainer application [60]. Also by Chang is a system that monitors the eating activity of occupants of a home through tagging food containers with RFID tags, and reading these via an RFID reader located under the dining table [61].

Several projects have studied the problem of trying to detect situations that occur in the home. As part of the House_n project, a large number of household objects were attached with reed switches or piezoelectric switches which acted as movement detectors [62]. Data gathered from the collection of switches was used with limited success to detect daily household activities of a single resident, such as preparing a meal, toileting, and washing clothes or dishes. Similar projects include Georgia Institute of Technology's Aware Home project [63], Microsoft's EasyLiving [64] and the Adaptive House project [65].

In the manufacturing domain, example projects include that of Lukowicz et al. [13] which mixed audio and motion sensor data to automatically track the progress of assembly tasks in a carpenter's workshop, using body-worn sensors. In this setting, activities such as sawing, drilling and hammering were successfully detected. Also included, is the project of Stiefmeier et al. [14], in which the check-points of the quality inspection procedure in a motorcar production line could be detected, through an array of motion sensors fitted in the quality-assurance inspector's suit.

The situations that were recognised in these domain and application specific

approaches are at a higher level, and are closer to the high-level situations that are sought to be detected in a pervasive situation determination system. The approaches presented here also illustrate the wide applicability and value of situation determination systems in real world settings.

The systems here have been specifically built to support their particular target application and situations. Each has relied on a team of skilled researchers to produce the situation descriptions and the accompanying software to recognise them. No direct support is given to recognising custom situations, nor to adaptive recognition through multiple situation descriptions or other means.

It is also worth noting that all of the situations recognised by these approaches are explicitly single-user. Switching from recognising a situation that centres around a single user, to recognising a situation that involves many people is significant, as it makes the situation more complex to model and recognise. The problem changes from modelling *the* person to *a* person, and in doing so changes the computational class of the problem. The recognition process must shift from checking simple constraints against the properties of a single entity to trying to solve an instance of the many pattern/many object problem [66]. This is an essential capability for supporting rich situation models and multiple viewpoints of a situation.

### 2.2.3 Learning situations

A number of works have attempted to automatically learn situation models by analysing captured context information. Using such techniques has the potential to provide a means to recognise new and custom situations as well as to adapt to changes in the available sensing infrastructure in the environment, two necessary capabilities in supporting pervasive situation determination. This section presents a review of these works and discusses the issues involved in using such approaches.

The first approach is that of Thomson et al., which presented an approach to learning situations inspired by information retrieval methods [67]. In this approach, observable relations between entities in a room are treated as terms in a document. A document is created by taking a snapshot of all the relations visible to the system at a given instant in time. A document is labelled with the high-level situation that is occurring in the room at the time. The system then analyses collections of such labelled documents using Support Vector Machine

techniques [68], to build up situation models that can be used to detect future instances of the situations. The system was evaluated on situations drawn from a computer science research department, and required on average at least 30 examples of a situation to achieve a recognition accuracy of 80% or more.

In the project described by Mühlenbrock et al., a Bayesian network was created using device location, PC and phone usage, ambient sound, and time of day context information as inputs, and trained to recognise three office situations labelled 'using PC', 'at desk' and 'discussing' [69]. The network was trained using over sixty examples, and achieved a fair level of recognition accuracy.

Hauptmann et al. report on a system that analysed video streams captured from nursing home cameras, and used computer vision techniques to try to track individuals as they moved through the home, and to identify dining and performing personal hygiene situations [70]. Tracking individuals achieved a high degree of accuracy using over ninety training examples per individual, though the accuracy of the situation detection was limited.

Oliver et al. created an application that could recognise a small set of six situations for a single user within a private office, including talking on the phone, talking to someone else in the room, and whether the user was present in or away from the office [71]. Example situations were manually collected and annotated, which combined input from a video camera, room microphones, and a computer's keyboard and mouse, and were used to construct layered hidden Markov models that could recognise future instances of the situations. Using 18 examples drawn from 3 hours of annotated sensor data, the system achieved an average 99.7% recognition activity for the small set of situations they studied.

Learning was also explored within the Gaia project, in which one week's worth of annotated sensor data from their Active Space research environment, including computer application state, light and sound levels, and the number of people in the room, was used to learn Bayesian network models for detecting a small, fixed set of situations that typically occurred within the active space including demonstrations, meetings, seminars, and the space being 'idle', which achieved an 84% recognition accuracy [22, 9].

In recent work that also focussed on supervised learning of situations, an approach was presented that specifically addresses the problem of trying to learn situations from a small number of examples [72]. A system was developed that analysed data from the MIT PlaceLab live-in laboratory [73, 74] to recognise a set of 24 simple, single-person household activities such as cleaning a surface,

preparing a meal, sweeping and listening to music. Using an average of 4.5 examples taken from a total of 4 hours annotated activity data, an average of 85% recognition accuracy was achieved.

Another interesting approach, also based on PlaceLab data, is that of Ye et al. [75]. The approach proposes a 'situation lattice' structure that specifies correlations between sensor data and situations, and can be used to infer situations and analyse the sufficiency of sensor sets. The approach attempts to integrate the advantages of both learning- and specification-based approaches by allowing environment experts to express and incorporate the semantics of particular context information and their domain knowledge, to help improve the learning process. Based on 40 hours of sensor data annotated with the corresponding household activities, the approach demonstrated excellent levels of recall, though relatively lower levels of precision in comparison to Bayesian network and decision tree based approaches on the same data.

The learning approaches presented above required a reasonable amount of example data and achieved a fairly good level of recognition accuracy. However, each of these are supervised learning approaches, meaning that they require a skilled administrator to manage a training period in which example data is collected, annotated and processed.

Unsupervised learning is also possible, where the learning process does not require an expert to manually classify example data according to its associated situation. For example, Mozer, in the Adaptive Home project, attempted to automatically regulate the lighting in the home, based on data from a collection of sensors including light, sound, temperature, and motion sensors, that were fitted throughout the home [76]. The home starts with default settings for regulating the lighting that are corrected over time by the home's inhabitants. With each correction, the home updates its model of the conditions in the home and their mapping to the desired light settings. Given enough corrections, the home will eventually learn the ideal, regular light settings preferred by the home's inhabitants. The prototype application for this required approximately 2,000 examples collected over 24 days before stabilising on the desired settings.

Patterson et al. explored a system that learns to infer whether a person is travelling by foot, by bus or by car, by analysing the raw data provided by a mobile GPS receiver carried by the person, in an unsupervised manner [77]. After analysing 29 examples, which collectively totalled 12 hours of data, the system achieved an estimation accuracy of 84%.

Roy et al. created a system which attempted to automatically learn occupancy patterns of rooms within a house, and adjust the temperature of a room to the preferred level of the person or group of people that it predicts will soon enter the room [78]. In this project, it took a period of approximately 60 days to learn the occupancy patterns with a prediction accuracy of 90%.

In all, the situation learning approaches presented explored a variety of different techniques to try to learn activities and situations from examples of sensor data, in medical, office and home settings. Some approaches achieved a high level of recognition accuracy, and the work of Albinali et al. managed a fair level of accuracy using only a small number of examples for each situation. They demonstrate that it could be possible to provide support for learning new, custom situations, and/or adapting current situations to new sensing infrastructure.

However, a common drawback across each of these methods is that the system is learning only to detect a label, or the category, of the situation. For example, in the approach of Mühlenbrock et al., a particular situation is represented purely by the string "discussing". These approaches lack a structure for the situation. Though the specific learning algorithm that the approach uses may have some internal structure, such as a network of floating point variables in a Bayesian network, or a partitioned, multidimensional space in an SVM, there is no guarantee that these structures map to real world entities or are directly useable by an application.

Learning or adapting situations without any user involvement at all is particularly difficult as the system itself must identify which data are important or relevant and which are not, and previously attempted unsupervised approaches such as those above of Mozer, Roy and Patterson have focussed on very simple situations and have taken relatively large amounts of time to learn.

Supervised learning approaches have shown promise, though the collection, annotation and processing of example data that occur throughout the training period must still be managed by an environment expert.

Another factor of both supervised and unsupervised approaches is that a time penalty will be incurred as examples of each of the custom situations are collected. This penalty must be paid again each time a situation is adapted to new sensing infrastructure. This limits the immediacy with which it is possible to adapt to new sensing infrastructure. This penalty also prevents being able to use the custom situations immediately, detecting short-lived situations or reacting to initial occurrences of a custom situation.

These factors restrict the support that learning-based approaches can provide for pervasive situation determination. Whilst some of the learning approaches presented were able to classify a situation according to a given label, it is desired that a situation be represented with a rich, queryable structure that portrays not only if a person is involved in a situation, but also the specific roles they play. In addition, whilst the majority of the approaches relied on an environment expert to manage the learning process, a pervasive approach should facilitate customisation of situations by end-users themselves, in terms of the features and entities involved in the situation. Learning customised situations and alternative descriptions takes time, yet creating and introducing them are sought to be immediate. Ideally, the support for customised situations and alternative descriptions should be orthogonal and complementary. That is, customisations can be created without regard to what future alternative descriptions and sensing infrastructure may be used to detect them, and when a change in which description is used or the available sensing infrastructure does occur, it is possible to continue to utilise the same customised situation without alteration. If a learning approach were to be used, it would be difficult to make these aspects orthogonal, and may require that each customisation be re-learnt for each alternative description.

## 2.3 Architectures

This section looks at the architectural aspects related to pervasive situation determination, covering both infrastructure-based and infrastructure-free architectures in Section 2.3.1 and Section 2.3.2, as well as adaptive and large-scale architectures in Section 2.3.3 and Section 2.3.4. It examines the suitability of existing approaches in supporting adaptable recognition, resource management, inter-environment operation and situation discovery.

### 2.3.1 Infrastructure-based architectures

Several context-awareness projects have adopted an infrastructure-based architecture. These approaches contend that providing an infrastructure capable of gathering, managing and disseminating context information can substantially reduce the complexity of developing context-aware applications. This section includes some of the most complete examples of these infrastructure-based architectures from the current literature, and considers their support for the architectural re-

quirements of pervasive situation determination.

Solar presents a centralised architecture realised as a graph-based abstraction for context collection, aggregation and dissemination [79]. In this approach, context-aware applications issue subscriptions to the context events they are interested in to a central 'star' process, which then deploys the necessary context operators on appropriate 'planets' as necessary, where a planet is any device that can execute an operator. As the number of application subscriptions grows, automated load balancing is achieved through the system managing the mapping of the resulting operator graph onto the network of planet devices.

Similar work had been conducted in the Context Tailor project, however this architecture includes the capability to limit the dissemination of context information based on a centrally enforced privacy policy. It also provides the capability to predict possible values of future contexts based on historical context information, using an integrated pattern analyser component [80, 81].

The one.world project presents an architecture in which all applications and components of the system expose all relevant contextual changes by publishing events to a common event bus [82]. This allows any application or component to detect these and appropriately adapt to its new operating context. The overall aim of the architecture is to allow all of the different devices in a pervasive computing environment that run the one.world platform to freely exchange context information and context-aware application instances between them.

Some projects have experimented with using an agent-based architecture for context-aware systems. This includes the work of Chen et al. in the CoBrA system [83]. In this approach, sensors, devices and applications are all represented as an agent. All context information is represented as instances of concepts defined in a common ontology. A context broker agent gathers low-level context information and generates higher-level context and resolves inconsistent context information via logical inference. Each user in the system is represented by a personal agent. After negotiation with a context broker agent, a personal agent may be granted permission to use a particular device or application in the environment, or perform a particular query on the current context information in the environment. Similar work is reported by Ranganathan et al. as part of the Gaia project, which supports agents with pluggable reasoning components [9].

In supporting pervasive situation determination, these projects provide good support for resource management. Each provide suitable mechanisms for the dissemination of context information by the system and its acquisition by appli-

cations, and the system can optimise this, for example, through Solar's automated load balancing mechanisms. The agent approaches are particularly well suited to this as they provide a common representation of context information through an ontology, and the agent substrates upon which they are based provide a convenient programmatic means of autonomously discovering and communicating with other context-producing agents hosted on a variety of devices.

However, none of these approaches feature explicit support for situations, and consequently no direct support for adaptable recognition of situations. While it is possible that a situation-aware application could subscribe to multiple, alternative sources of context information to detect a situation, the onus is then on the application developer to implement considerable functionality by him/herself. This would include the recognition of the situation, fusion of the results, as well as creating multiple descriptions of the situation, potentially for each situation and application that they wish to develop, and without any guarantee that they could be recognised in environments other than that for which they were specifically developed.

The agent-based approaches offer limited support for inter-environment operation, again due to the agent substrates upon which they are based, that typically provide agent communication protocols to send and receive messages to and from other environments. However, none of the approaches offer direct support for recognising nor discovering situations that are occurring in external or remote environments.

Also note that each of these approaches utilise a centralised architecture. To fully support pervasive situation determination, operating exclusively with a centralised, infrastructure-based architecture is insufficient. Limiting context information aggregation and situation recognition to a single machine acts not only as a single point of failure, but also as a bottleneck when operating with a large number of situations, people and devices, and/or many locations. Furthermore, situation-aware applications must depend upon the infrastructure to operate, and so cannot be used in ad hoc, mobile or other resource-constrained settings where the infrastructure is not available.

The key ideas which can be taken from these projects and built upon in support of pervasive situation determination include resource management, system autonomy and open communication.

As in the Solar project, providing resource management capabilities can offer particular advantages in pervasive situation determination systems, as many de-

vices active within the system will be resource-constrained, mobile devices. Situation determination can involve significant computation and wireless network communication. The ability to manage where and which situations are recognised can allow not only more complex situations to be recognised than a single resource-constrained device could process alone, but also to preserve the battery life of the mobile devices involved.

Agent-based systems provide an appropriate substrate to autonomously manage the discovery and interaction of system and application components in collaboratively recognising the situations occurring within the environment.

Likewise, open communication, like that achieved by one.world's common event bus, can provide particular benefit for pervasive situation determination. Enabling arbitrary components of the system, perhaps previously unknown to each other, to communicate with each other, is an essential element in providing an open, collaborative situation recognition process.

### 2.3.2 Infrastructure-free architectures

Some projects have explored infrastructure-free architectures that operate independently, and perhaps collaboratively, on mobile devices, rather than rely on an external infrastructure. This section includes some of the most fully developed examples of these architectures. Provision of both infrastructure-based and infrastructure-free modes of operation are necessary for pervasive situation determination, and this section explores the support that current infrastructure-free architectures can give to each of the four architectural requirements.

The ContextPhone is one such project [84]. In this architecture, mobile phones are fitted with a variety of software and hardware sensors that can detect information such as which applications and features of the phone are being used, the user's location, and the identity of other mobile phones within Bluetooth range. Summaries of this information can then be displayed on the user's phone, or sent upon request via GPRS and displayed on other users' phones. This approach is almost the antithesis of situation determination as no attempt is made to infer the user's situation from the available context information. Instead, it focuses on gathering and disseminating the summaries, and it is up to the user receiving the summary to determine which situation, if any, is occurring. Leaving it up to the user to determine the situation based on a visual log of raw context places a significant burden on the user, which would only be exacerbated as the amount

of context information grows, or if key contexts which the user expects are not currently available, and precludes the capability of autonomously performing actions, which is one of the principle goals of situation-aware systems.

An infrastructure-free architecture that does attempt to infer the user's situation is the SPECs project [85]. The aim of the project is to identify the situation of a single user "at large in their world". It does so through mobile devices called 'SPECs' which are small enough to be attached to personal belongings, clothing, or worn as a watch or necklace. A SPEC is a basic computing device that can detect other SPECs in proximity, can switch a visible LED on and off, and can be loaded with simple controller programs. Though limited, a situation-aware reminder application was implemented with this architecture. It detected a schoolboy's situation as he went to school, that is, it recorded which SPEC-tagged belongings were in proximity to his personal SPEC between 08:00 and 08:30. Then, it detected his situation on his way home from school, that is, which SPEC-tagged belongings were in proximity to his personal SPEC between 15:30 and 16:00. If the set of belongings differed on the way home, the LED on the personal SPEC would be lit, to remind him that some personal belonging had been left behind.

Interesting work has been reported by Strohbach et al. in which independent, self-contained devices with sensory capability interact to co-operatively determine their common situation [86, 87]. This architecture is realised by fitting physical objects with Smart-It devices, configured with basic sensing capability, wireless communication, and sufficient computing capability to run rule-inference software [88]. Facts generated from sensor readings or inferred by a Smart-It's rule engine can be transmitted to other Smart-Its. For a set of Smart-Its, the rule engine of each can be loaded with different rule sets that recognise fragments of a situation, which can be combined to determine the whole situation. This architecture has been used in a domestic setting, where a kitchen table, jug and two glasses were fitted with Smart-Its and were able to co-operatively determine simple activities such as placing a glass on the table, and filling a glass with water [86]. In another deployment, Smart-Its were attached to chemical storage containers that could co-operatively detect the potentially hazardous situation of inappropriate chemicals being stored in proximity of one another [87].

The SPECs and Smart-Its projects demonstrate the utility that can be offered by situation determination using only small, embedded, autonomous devices. However, in both these architectures, a situation is not defined as an

independent, reusable entity, nor is there support for adaptable recognition or situation customisation. Programming the original situations in the SPECs and Smart-Its projects is quite demanding, the former requiring pattern classification specifications based on the temporal proximity of sightings of the SPEC devices in range, and distributed logic programs to be written for the latter. In all approaches, the applications that were written target a specific device, relying on the fixed sensing infrastructure it provides. In addition, the application developer must still provide an ad hoc definition for each situation that each application requires. The latter approaches focus exclusively on the local environment and do not consider inter-environment operation nor situation discovery.

The work of Strohbach et al. is interesting in that the overall situation is computed co-operatively by each of the devices involved in the situation. Such co-operation would be particularly advantageous when attempting to detect multi-person and/or multi-device situations in environments where no dedicated sensing infrastructure is available, and the situations must be recognised by an ad hoc collection of mobile, resource-constrained devices. However, in the Strohbach approach, many aspects are fixed, allowing the system to operate under a closed, fully specified model. For example, the number of devices and/or the number of objects involved in the situation are known and fixed, the specific sensing infrastructure is known and fixed, as is the single situation which is to be recognised and the single application which is run. To adequately support pervasive situation determination, these ideas of co-operatively recognising a situation must be extended to operate in a dynamic and open-ended environment, as there are many aspects that may not be known a-priori and will vary. Such aspects include the number of people and devices involved in the situations to be recognised, as well as the particular sensing infrastructure that will be available. In addition, there is the set of situation-aware applications that may be run, the set of situations these applications will require, and also the users' customisations of these situations.

### 2.3.3 Adaptive architectures

Adapting to a changing sensing infrastructure is one of the key capabilities required for pervasive situation determination, and this section looks at projects which have focussed on methods of context-oriented adaptation and the support they can lend to adaptable situation recognition.

One such project is the work of Huebscher et al. that provides an architecture for adapting the delivery of context information [89]. In this approach, applications not only define which context information they require, but also a utility function based on quality metrics of the required context information. Then, given multiple alternatives for providing the same type of context, the architecture applies the utility function to each alternative and delivers the one with maximum utility. Example quality metrics that may be used in a utility function include precision and freshness.

Dobson et al. [10] provide an architecture that exploits multiple, related forms of context information to more reliably detect a person's location. It uses a voting mechanism that, based on models of the precision, decay and confidence of each source, combines readings from a location system, door sensors, and indications that a person is using a computer at a known, fixed location to produce a composite measure of confidence over a set of possible locations. The system essentially adapts its recognition of a location given various forms of available context information.

These works provide examples of how the delivery of context information can be adapted using multiple, alternative forms of source context information available in the environment to increase the confidence of the delivered context information to an application. Both of these approaches focus exclusively on individual, atomic pieces of context information. To support adaptable recognition for pervasive situation determination, such approaches must be extended to be able to incorporate multiple, alternative situation descriptions, each of which may comprise a complex composition of context information.

### 2.3.4 Large-scale architectures

A pervasive situation determination system must address the requirements for inter-environment operation and situation discovery to enable the system to recognise situations not only in the local environment, but also locate and recognise situations occurring in external and possibly distant environments. This section presents preliminary existing work that looks at large-scale, inter-environment operation of context-aware systems, and what support it provides in these areas.

One such work is the Super Spaces extensions [90] of the Gaia project [20]. Within Gaia, an Active Space represents a physical space, typically a room, that provides a context-aware infrastructure that allows applications to detect and

react to changes in context within the space [20]. A Super Space represents a linked collection of Active Spaces which permits the management of multiple spaces using a single interface, allowing applications to perform actions in, or gather information from, multiple spaces [90]. In this large-scale architecture, Super Spaces may be composed in a hierarchical manner, allowing operations on a Super Space to be applied recursively on each of the Active or nested Super Spaces it contains, or also in a peer-to-peer fashion, allowing applications to compose services and/or context information from one or more Active Spaces. However, details of how Active or Super Spaces discover and interact with each other, or how context information within them is discovered and disseminated are not given.

The GLOSS project developed the Hearsay architecture to support global-scale location-aware applications [91]. This approach offers similar capability to Pascoe's Stick-e Note architecture developed earlier [6], though at a global scale. Information with an associated profile can be attached to a physical space in potentially any geographical location. When a user who has a matching profile enters that space, the information is delivered to that user. The network of computing nodes that store and deliver the messages form a hybrid hierarchy of peers architecture. Nodes at a similar level of granularity are linked in a peer-to-peer network, and these sets of peers are linked hierarchically. For example, countries form a set of peers, and the regions within a particular country form another set, and the cities within a region another set, and so on. This architecture is presented as being well suited to the geo-spatial nature of Hearsay applications, as it allows peer topologies and protocols to be tailored to exploit locality knowledge, and computing nodes need not be co-located with the geo-spatial region that they represent, only the peers need know the peering relationships.

Other works that intend to feature large-scale architectures supporting inter-environment operation include Project Aura [92, 93] and the PICO project [94], though details of how these architectures are implemented are not given.

The Hearsay architecture, though a global scale system, supports processing context information (the user's location, and profile matching) and reacting to it (displaying information to the user) only locally. In the Super Spaces approach, inter-environment operation is limited to using different services from multiple environments, and sending messages or commands to several environments at once. Pervasive situation determination requires an architecture that enables applications to query the situations of any person, artefact or location, regardless

of whether they happen to be in the same environment or in an external, distant environment.

A key benefit for supporting pervasive situation determination is demonstrated by the GLOSS project's use of a peer-to-peer network. Such a network has several advantages. It offers simple operation, allows information contained within it to be found rapidly by the system and applications anywhere in the network, can scale to very large network sizes, and is robust to changes in the network, adapting automatically when host machines join and leave. This provides an appropriate foundation for both the inter-environment operation and situation discovery requirements. Though to fully support pervasive situation determination, it is necessary to expand upon profile matching at a fixed location, to provide dynamic, inter-environment situation recognition, where the target environment is discovered at runtime.

## 2.4 Summary

This chapter has presented a review of existing works with regard to the support they offer for pervasive situation determination. This is summarised below for each of the requirements of pervasive situation determination.

First, the modelling requirements:

**Customised situations** - In order to capture the distinctive features of situations that are particular to the individual user or their environment, end-users themselves must be able to create their own customised situations, without relying on specialist maintenance to achieve it.

Direct support for situation customisation did not feature in any of the presented approaches. To create a customisation required altering a situation description directly, which demands skill in application programming, the particular description approach, correlating the context information into situations, as well as intimate knowledge of the context information available and the situations that occur in the environment. This is much too specialised and unsuitable for use by typical end-users.

Empowering the end-user to customise situations themselves, in terms meaningful to them, as is required to support pervasive situation determination, is not possible with current approaches.

**Rich situation models** - Models of a situation must be rich in detail and structure, allowing the specific roles within the situation to be captured and the

system's and applications' reactions more precisely defined, so as to more closely match the user's needs.

In those approaches that modelled the concept of a situation explicitly, it would typically be presented to applications merely as a string label, and not a rich structure detailing the situation's characteristic elements drawn from the group of people and devices involved within it.

The approach of Crowley et al. [35, 36, 37] offered the greatest level of support in this aspect, as it defined a basic, general structure for a situation which included explicit representation of different, abstract roles within a situation and the relations that hold between them.

**Alternative descriptions** - To be pervasive, recognising a situation cannot rely upon particular sensing infrastructure being available. The model of a situation must include multiple, alternative descriptions that specify how the situation can be recognised from a variety of different forms of sensing infrastructure.

There was no approach that offered explicit support for defining and utilising alternative situation descriptions. In some, it was possible to implicitly define alternative descriptions by resolving to the same string label, or deducing the same situation predicate as discussed for the logic-based approaches. However, this prevents effective management of the multiple descriptions that are used, such as fusing their results to achieve a greater confidence that the situation is occurring, or properly integrating and limiting the potential proliferation of descriptions when combining the multiple descriptions with end-users' customisations.

**Multiple viewpoints** - To be fully pervasive, a situation determination system must be capable of reporting a situation from the viewpoint of any person, device or location to any user in the system.

Existing approaches however, are almost exclusively grounded in the single viewpoint of the current user in the local environment. No approach gives direct support to modelling multiple viewpoints of a situation. Whilst in some approaches it may be possible to define multiple viewpoints implicitly, this demands that several additional descriptions must be created. Furthermore, in doing so, it may not be possible to tell which of these alternative viewpoints are related to the same instance of a situation. Supporting multiple viewpoints in a situation shifts its recognition into a more complex class of problem, and therefore requires a different type of recognition process than those that back the existing approaches.

And also, the architectural requirements:

**Adaptable recognition** - A pervasive situation determination system must provide adaptable recognition for the situation recognition process as a whole, incorporating new descriptions and sensing infrastructure as they appear in the environment, as well as adapting to the loss of existing sources, to obtain a greater availability of both the system and users' situation-aware applications.

In both the infrastructure-based and infrastructure-free architectures reviewed, none provided direct support for adaptable recognition of situations. Whilst in some approaches it would have been possible for a situation-aware application to subscribe to multiple, alternative sources of context information to detect a situation, the onus is then completely on the application developer to implement the recognition of the situation, as well as its adaptation, by him/herself for potentially each situation and application that they wish to develop.

The approaches reviewed that focussed on adaptation were concerned exclusively with atomic pieces of context information. None addressed adaptable recognition of complex, structured compositions of context information that comprise a situation.

**Resource management** - As recognising a situation can be a computationally expensive task, and especially as it must often be performed by resource-constrained devices, a pervasive situation determination system must strive to manage the resources within it, shifting expensive tasks to the devices that can best afford them.

The infrastructure- and agent-based approaches reviewed offered good support for resource management, providing suitable mechanisms for the dissemination of context information by the system and its acquisition by applications. What was lacking however, was the inclusion of the concept of a situation. As without directly modelling a situation, it is difficult to achieve precise control over how, where and what situations are recognised and how the results are shared.

**Inter-environment operation** - To be pervasive, the reach and scope of situation-aware applications must be able to extend beyond the local environment, to whichever external or remote environment that hosts the situations a user is interested in.

Few existing project have addressed inter-environment operation. While some architectures, such as the agent-based approaches, may provide inter-environment communication protocols, none of the approaches offer direct support for recognising situations in external environments.

In the two projects that specifically addressed large-scale operation, sufficient

support was still lacking. Hearsay, despite having global scope, only supported processing context information and reacting to it locally, while Super Spaces focussed not on inter-environment situation determination, but on sending messages and commands to several smart environments at once.

**Situation discovery** - When the situations a user is interested in extend beyond the local environment, a pervasive situation determination system must have means of discovering situations that are occurring elsewhere within the network of environments.

This aspect of pervasive situation determination received no attention in the existing approaches.

In summary, it is not currently possible to achieve pervasive situation determination using existing approaches. Although partial support is given for some of the requirements by the existing approaches, to fully meet all the requirements would demand significant additional work to be performed on the part of the application developer. The requirements address common, inter-related modelling and architectural elements that pertain to pervasive situation determination systems. These can be more appropriately realised in a middleware layer, which can then be exploited by a wide range of situation-aware applications, as well as free the situation-aware application developer from the intricacies of developing pervasive situation determination support.

From the support that is offered by the existing approaches however, there are a number of ideas and techniques that can be built upon, by extending them and introducing novel elements where necessary, that can aid in achieving pervasive situation determination.

It is possible to extend a number of the modelling aspects that have been presented in this chapter. For example, the idea of supporting annotation of symbolic locations can be extended to form 'location types', to offer greater convenience and generality in writing rich situation descriptions. Furthermore, a logic-based approach, similar to those presented earlier, could be extended with additional modelling constructs to provide greater expressiveness in describing situations. It is also possible to derive a fuzzy logic scheme which can be used to manage the uncertainty associated with different sources of context information as well as the situations based upon them, but that is simpler and lighter than those presented here. Drawing from the graphical and role-based representation approaches, a number of techniques can be developed for the model that simplify working with complex situations and the large amount of context information from which they

are recognised, and help partition them into smaller, more manageable and readily understandable sets.

There a number of ideas from the architectural aspects presented here that can also be built upon. For example, the resource management capabilities that were covered can be extended to include situations explicitly, allowing control over where and which situations are recognised, helping to preserve the battery life of mobile, resource-constrained devices. Moreover, an agent-based substrate can be used to autonomously manage the discovery and interaction of system and application components in collaboratively recognising the situations occurring within the environment. The idea of using an open communication style, enabling arbitrary, perhaps previously unknown components of the system to communicate with each other, can be effectively exploited in achieving an open, collaborative situation recognition process. It is also possible to extend the idea of using multiple sources of atomic pieces of context information to increase the confidence of the context information delivered to the system or an application, to apply to situations and complex compositions of context information. Furthermore, the use of a peer-to-peer network may be innovatively applied to provide dynamic, inter-environment situation recognition and real-time situation discovery.

In the following chapters, it is shown how this thesis builds upon these good ideas from the literature, and also introduces its own novel elements, in addressing the problem of designing a model and a supporting infrastructure that can fully provide the necessary support to realise the requirements and benefits of pervasive situation determination.

# Chapter 3

# A Model for Pervasive Situation Determination

## 3.1 Introduction

This chapter presents a novel situation modelling approach developed to meet the requirements of pervasive situation determination. This includes support for customised situations, which allow individual users to have situations that are unique to them and their environment to be recognised, and be created by the end-users themselves. Also, rich situation models are supported, which capture a wealth of details about the features and relationships of a situation, allowing the way in which the system and situation-aware applications react to situations to more closely match the user's needs. In addition, alternative situation descriptions are supported, making recognition of a situation robust against changes in environment and supporting sensing infrastructure. Moreover, multiple situation viewpoints are supported, making the possibility of recognising a situation of any person, device, or location, pervade throughout the network of environments and situation-aware applications.

To begin, an overview of the situation model is presented in Section 3.2. Following this, each of the different parts that comprise a model of a situation are looked at in turn in Sections 3.3 through 3.5. Section 3.6 describes how uncertainty associated with sensor data is naturally incorporated into the model of a situation, before a summary of the model is given in Section 3.8 highlighting the ways in which the particular requirements of pervasive situation determination

are fulfilled.

## 3.2 Overview of the situation model

Similar to other contemporary definitions of a situation in a pervasive computing environment [95, 96, 37], this approach considers a situation to be an external, semantic interpretation of context information that represents the high-level activity of an individual or a group of people and the devices they use.

Recognising a situation involves three main types of information - sensor data, context information and situations themselves. Sensor data represents the raw data output from a sensor, which may commonly be in a sensor or vendor specific format. For example, consider a stream of raw signal strength readings produced by a device with wireless network capability. Context information is a structured representation of sensor data. In this approach it is a typed relation that represents some property of an entity. It may be derived from sensor data and/or other context information. For example, from the stream of signal strength readings it may be possible to infer the location of device, and if the device represents a person, then the location of that person too. Such individual pieces of context information may be interpreted along with several others to deduce that the person or device is currently involved in a particular situation.

A description of a situation has three parts. First is the situation itself, which is an abstract description of the features a situation contains. A feature is an externally observable property that characterises some aspect of the situation. Then, there are customisations, which detail simple constraints on the features of a situation, which allow environment administrators and end-users to tailor the situation to their own environment and preferences. Finally, there are situation specifications, which are more complex documents describing the entities, properties, locations and constraints that are used to detect when its associated situation is occurring. It is a situation specification that provides a mapping from context information, derived from low-level sensor data available in the environment, to the high-level features of the situation that is sought to be recognised. A person or entity is considered to be involved in a situation if they appear or are included in one or more of the features of a situation, when the situation is occurring. A situation is considered to be occurring when all of the relations and constraints included in an associated specification hold.

Below is presented a brief overview of each of these parts. Following this are

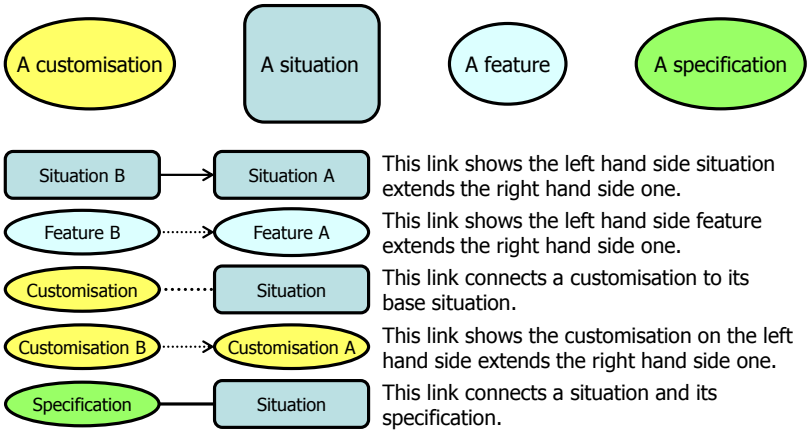



Figure 3.1: A key for the figures included in this section.

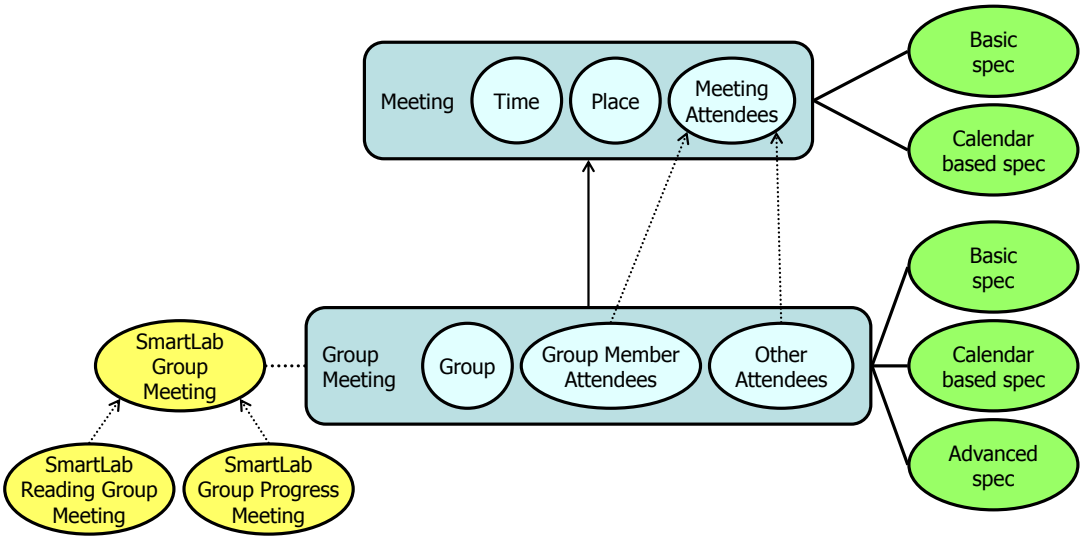



Figure 3.2: An illustration of the structure of the 'Meeting' and 'Group Meeting' situations, as well as their specifications and customisations.

sections describing each part in detail.

Examples of each of these three parts are illustrated in Figures 3.2 and 3.3. Figure 3.1 provides a key for these figures. These examples are taken from an actual University deployment discussed in Chapter 5.

Let's first look at Figure 3.2. Presented here are two situations, a 'Meeting' situation and a more specific 'Group Meeting' situation. It can be seen that each situation consists of a set of features. For example, a meeting may be characterised by the time at which it occurs, where it occurs, and the people

who are attending the meeting. In Figure 3.2, these are represented by the features 'Time', 'Place' and 'Meeting Attendees' respectively. Each light blue oval illustrates an individual feature, and these are contained within the darker blue rectangle, showing that this set of features make up the 'Meeting' situation.

Figure 3.2 also includes a 'Group Meeting' situation which has the additional features 'Group', 'Group Member Attendees' and 'Other Attendees'. The arrow between the 'Group Meeting' and 'Meeting' situation indicates that a group meeting is a more specific type of meeting situation, and that it includes all of the meeting features. The 'Group' feature represents the name of the group. The 'Group Meeting Attendees' and 'Other Attendees' features represent the people who are attending the meeting, but more specifically, those who are group members and those who are not. The dotted arrows from these features to 'Meeting Attendees' show that they are a more specific type of that base feature.

In order to adapt a situation to a particular environment or to an individual's preferences, an end-user can create a customisation. A number of customisations are shown in Figure 3.2 as yellow ovals.

A customisation allows an end-user to place simple constraints upon the features of a situation to define a bespoke variation of the situation that is particular to them or their environment. For example, the 'SmartLab Group Meeting' customisation can be created by simply adding the constraint that the featured group of the 'Group Meeting' situation should be the SmartLab group. The dotted line connecting the 'SmartLab Group Meeting' customisation to the 'Group Meeting' situation indicates that it is a customisation of that situation. Furthermore, the 'SmartLab Reading Group Meeting' and 'SmartLab Group Progress Meeting' customisations can be created by further adding the constraints that the 'Time' and 'Place' features must match the particular times and places at which these meetings happen. The dotted arrows from these customisations to the 'SmartLab Group Meeting' customisation show that they are more specific types of that base customisation. Customisation like this provides the end-user with a simple and flexible means to tailor the situations that are recognised to include the ever-evolving set of bespoke situations within their environment.

Figure 3.2 demonstrates that three different types of inheritance are employed within the model: where a feature inherits from another, where a situation inherits from another, and when a customisation inherits from a situation or from another customisation. Each of these different types of inheritance can be identified from Meyer's inheritance taxonomy [97]. When a feature of a situation

extends another, such as 'Group Member Attendees' or 'Other Attendees' in the 'Group Meeting' situation extending 'Meeting Attendees', it is an example of subtype inheritance. This indicates that the people who are group member attendees form a subset of all the people who are meeting attendees, and that the set of group member attendees is disjoint from the set of any other subtype heir, such as the 'Other Attendees'. When a situation extends a base situation, it is an example of extension inheritance, as the new situation introduces features that are not present in the base situation, nor are applicable to direct instances of the base situation. Whereas, when a customisation extends a situation or another customisation, it is an example of restriction inheritance, as instances of the new customisation are those instances of the base situation or customisation that satisfy additional constraints. It is in exhibiting these separate types of inheritance, in addition to the different forms of content that each of them addresses, that make features, situations and customisations distinct elements of the model.

A specification describes how a particular situation can be recognised from context information derived from low-level sensor data. In Figure 3.2, the green ovals represent specifications of a situation. The line connecting a specification to a situation shows that the specification recognises that particular situation.

As can be seen from the figure, a situation may be described by more than one specification, such that each can use its own set of sensor data and context information to recognise the situation in a different way. Figure 3.2 shows that the 'Group Meeting' situation has three different specifications: 'Basic spec', 'Calendar based spec' and 'Advanced spec'. The basic specification simply uses the time and the location of the attendees to detect the situation. The calendar-based specification uses calendar or personal organiser information in addition to the location of the attendees to detect the situation. The advanced specification more closely examines the structure of the group of attendees to determine who are the group members and who are the other attendees.

Having multiple specifications allows recognition to dynamically adapt to the context information that is currently available in the environment, enabling situations to be detected in the greatest detail possible. Furthermore, different specifications can be created to suit different modes of operation. For example, lightweight specifications can be used when resource-constrained mobile devices are being used to recognise the situation, and more complex, richer specifications can be used when dedicated infrastructure is available to recognise the situation. In addition, as will be presented in Section 3.6, it allows the fusion of each of the

specifications to produce a higher confidence result for the situation overall.

The following sections present each of the three parts of a situation description in detail, and further discuss their support for pervasive situation determination.

## 3.3 Situations

As noted in the overview, a situation description comprises a set of features, where a feature is an externally observable property that characterises some aspect of the situation.

A situation is an abstract description, as it declares *what* is recognised for a given situation, but says nothing about *how* it is recognised. While a specification is necessary to recognise a situation and its features from low-level sensor data, a situation description is separate from any particular specification.

The features of a situation description serve as a common interface to the situation for customisations, specifications and situation-aware applications. This allows customisations to be defined solely upon the features of a situation, and can therefore be applied independently of which particular specification is used to recognise a situation, and vice-versa. Similarly, behaviours that may be defined by situation-aware applications are also based solely upon the features of a situation, and so too can operate without being aware of, or tied to, any particular specification. This offers a great level of flexibility in dynamically adapting the recognition and reporting of situations to the continual changes in the available sensing infrastructure that will occur in a pervasive computing environment.

Part of this flexibility lies in being able to reason about situations abstractly and to create hierarchies of situations. For example, in Figure 3.2 the inheritance link connecting the 'Group Meeting' to the 'Meeting' situation shows that 'Group Meeting' is a more specific type of 'Meeting' situation. When a situation inherits another, its feature set will contain the features of the parent situation, as well as its own features. So the feature set of a 'Group Meeting' situation will contain the 'Time', 'Place' and 'Meeting Attendees' features from the 'Meeting' situation, as well as its own 'Group', 'Group Member Attendees' and 'Other Attendees' features.

It is inheriting from a base situation like this that allows a descendent situation and its specification to be reasoned about abstractly. For example, if a situation-aware application requested notification of the meeting situations a particular person was involved in, it would receive notification of instances of 'Group

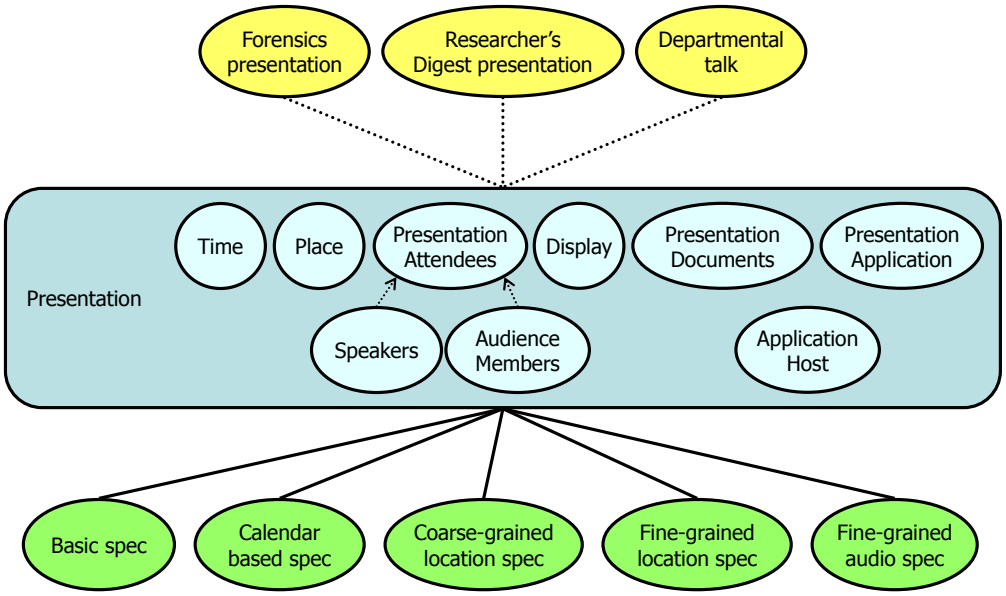



Figure 3.3: An illustration of the structure of the 'Presentation' situation, as well as its specifications and customisations.

Meeting' situations, as well as the customised situations derived from this, in addition to Meeting situations. Therefore, even in cases where no direct 'Meeting' specifications are available, the request can still be satisfied by using 'Group Meeting' specifications instead.

It is also possible for a feature to subclass another. To illustrate this, consider the 'Presentation' situation presented in Figure 3.3. This situation represents the time and location the presentation takes place through the 'Time' and 'Place' features. The people involved in the presentation are represented by the 'Presentation Attendees', 'Speakers' and 'Audience Members' features. The materials being presented, and the display, application and host machine used to present them, are represented by the 'Presentation Documents', 'Display', 'Presentation Application' and 'Application Host' features, respectively. There are three customisations of the 'Presentation' situation defined, which include the presentations of the Digital Forensics course at the University named 'Forensics presentation' in the figure, the post-graduate weekly research seminar presentations 'Researcher's Digest presentation', as well as the more general 'Departmental talk'. Five different specifications exist for the 'Presentation' situation, ranging from a basic specification at the left of the figure, to a fine-grained, audio-based specification at the right.

In Figure 3.3, the 'Speakers' and 'Audience Members' features both subclass the 'Presentation Attendees' feature, showing that they are a more specific type of the base feature. An entity that appears in a subclassed feature will also appear in the base feature. For example, as 'Audience Members' are more specific 'Presentation Attendees', any person who features in the situation as an audience member will also feature as a presentation attendee.

Subclassing features like this allows rich situation descriptions to be created. It enables the situation author to represent not only whether a person or device is involved in a situation or not, but also which role they are playing within the situation, and further still which specific type of role. For example, it is possible to tell that a particular person is involved in a presentation, that they are an attendee, and specifically whether they are a speaker or an audience member.

The set of features declared for a situation is intended to be a comprehensive set for that situation. However, it is not required that a specification recognise all of the features of a situation. Because the resources that will be available in an environment will vary and be unpredictable, it cannot be guaranteed that sufficient information will be available to recognise all of the possible features of a situation. However, rather than fail to recognise the situation, it may be possible to recognise a partial version of the situation. That is, a specification may recognise some, but not all of the features. Any of a situation's features may or may not be available in a reported instance of a situation at application runtime.

Consider again the Presentation situation in Figure 3.3. This situation has the features 'Time', 'Place', 'Presentation Attendees', 'Speakers', 'Audience Members', 'Presentation Documents', 'Display', 'Presentation Application' and 'Application Host', where 'Speakers' and 'Audience Members' subclass 'Presentation Attendees'. To detect if a person is a speaker may be more difficult, or demand more specialised or high-performance sensing capability, than to detect if a person is an attendee. For example, it may be possible to detect that a person is an attendee in a presentation simply because they are in the room in which the presentation is occurring. However, to detect if a person is a speaker may require fine-grained location information, such that it can be detected that they are standing in the speaker area, or may require advanced audio processing, such that it can be detected that they have been the only person speaking for most of a period of time. Where such information is available, it can be exploited, and the speaker role can be detected. In cases where such information is not available,

the system is still able to recognise the less specific attendee role.

It is this flexibility that allows the range of specifications of varying granularity shown in Figure 3.3 to be created for the 'Presentation' situation. In addition to basic and calendar-based specifications, which are computed in a similar fashion to those for the 'Group Meeting' situation, the 'Presentation' situation has a course-grained location-based specification and two fine-grained specifications. The course-grained location-based specification can detect all of the roles of the situation, but only to the level of granularity of Presentation attendee. Detecting whether an attendee is an audience member or a speaker can be achieved using the fine-grained location-based specification. Alternatively (or additionally), the audience member and speaker roles can be detected using the fine-grained audio-based specification.

Declaring a comprehensive set of features for a situation has the advantage of providing a consistent interface to situation-aware applications, and that the situation determination system may utilise the finest-grained reports of a situation that the current environment is capable of providing.

## 3.4 Customisations

Customisations are used to specify simple constraints on the features of a situation, in order to customise them to the requirements of a particular user or to a particular environment. As they are based on the features of a situation description, they can be applied on top of any situation specification that describes these features.

As customisations are intended to be created by end-users and environment administrators, the customisation process is deliberately kept simple, to make it easy to write customisations. It permits constraints to be defined for features, where a feature will either be a single entity (such as a person), or a collection of entities (such as a set of people). For a single entity feature, a customisation will be a set of simple Boolean expressions based on the properties of the entity, such as that the time is between 15:00 and 16:00. For a feature that is a collection of entities, a customisation will either be a set of simple Boolean expressions based on properties of the collection itself, such as the size of the collection, or on the members of the collection, such as if the collection contains a person whose ID property matches a certain value. A complete customisation may define expressions over one or several features.

As an example, the 'Researcher's Digest presentation' customisation shown in Figure 3.3 adds the constraints that the place of the presentation must be a particular room within the Computer Science Department of the University, that the time is between 13:00 and 14:00, and that all the audience members are post-graduate students.

Customisations, like situations and features, may be constructed in hierarchies and be reasoned about abstractly. For example, in Figure 3.2, the 'SmartLab Reading Group Meeting' is a more specific type of customised situation than 'SmartLab Group Meeting', which is a more specific type of customised situation than the 'Group Meeting' situation, which is a more specific type of situation than 'Meeting'. Note however that a customisation does not introduce any additional features, but simply further refines or adds additional constraints to the existing features of its base situation.

## 3.5 Specifications

The final type of situation description presented here is a situation specification. A specification provides a mapping from context information, derived from low-level sensor data available in the environment, to the high-level features of a situation and is used to recognise that situation.

### 3.5.1 Specification structure

Figure 3.4 depicts the fine-grained location-based specification for the 'Presentation' situation, previously featured in Figure 3.3. There are two types of specification used to recognise a situation - a role specification and a situation specification. A role specification describes the individual role an entity plays in the situation. That is, it describes a set of constraints and relations that must hold for a particular entity to be playing the role. For example, in Figure 3.4, the Speaker role identifies that a particular individual may potentially be a speaker, and the Audience Member role identifies whether a particular individual is potentially an audience member or not. The situation specification then describes the situation as a whole. From the point of view of a situation specification, a situation can be seen as a particular composition of roles. That is, it describes a set of roles and a set of constraints and relations upon them, which must hold in order for the situation to be occurring in the environment.

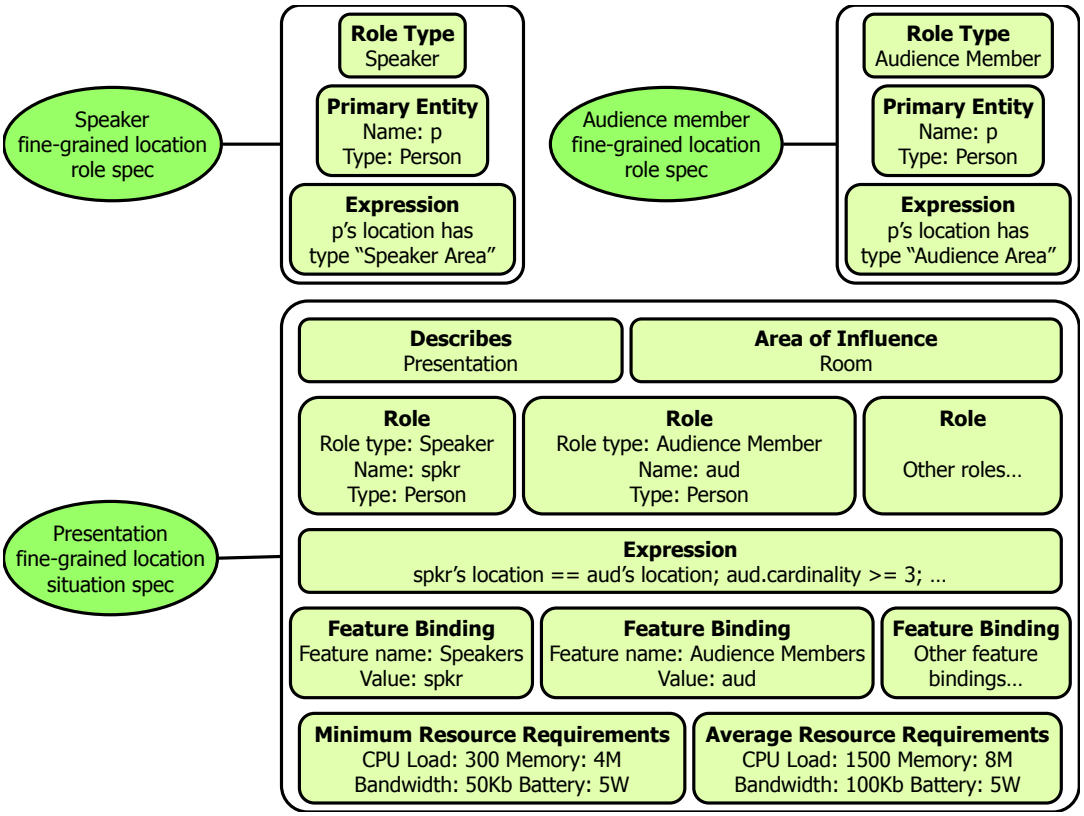



Figure 3.4: An illustration of the structure of the fine-grained location-based Presentation specification.

This two-tiered structure allows for a simple and natural description of a situation. In particular, it is easy to express constraints over the set of all of the entities playing a role and on the properties of the set itself, such as that there must be at least three audience members in a presentation, that all group member attendees of a group meeting must belong to the same group, or that not more than 25% of the total number of attendees of a group meeting are not members of the group. Such constraints are difficult to express using the strict logic-based approaches presented in Chapter 2, though they were used extensively throughout all of the specifications developed in Chapter 5.

### 3.5.2 Role specifications

Let's further examine the role specifications depicted in the top half of Figure 3.4. A role specification may consist of four parts - a role type, a primary entity, auxiliary entities and an expression. The role type gives a name to the type of role the

specification describes. This type information is used to connect compatible role specifications with situation specifications. The primary entity states the type of the person or device that will play the role, as well as providing a name for the primary entity, allowing it to be referenced in the role specification's expression. A role specification may define one or more auxiliary entities, providing a type and name for each. Auxiliary entities may be required when the role of the primary entity is recognised in relation to them. The expression includes all of the constraints that must be satisfied by the primary and any auxiliary entities for the primary entity to be playing the role. Expressions within a role may refer to the properties of the primary and auxiliary entities. The current date and time may also be referred to. The expressions can include the standard comparison operators, the Boolean operators $\neg$, $\wedge$, $\vee$, and $\Rightarrow$, as well as other type specific operators and relations.

The role specification depicted in the top left of Figure 3.4 recognises a speaker of a presentation using fine-grained location information. It declares the role type to be 'Speaker', the primary entity to be of type 'Person' and named 'p', and the expression to contain a check that the primary entity is currently located within a speaker area. This role specification does not define any auxiliary entities as the role can be recognised based solely upon the properties of the primary entity. The audience member role depicted in the top right of Figure 3.4 is defined similarly, though the expression in this case checks that the primary entity is within an audience area.

### 3.5.3 Location types

The checks on the primary entity's location made by the role specifications in Figure 3.4 warrant a little more explanation. Location information is commonly regarded as essential for describing situations [18]. The modelling approach presented here assumes that a location property is defined for each physical entity, including people and devices. This requires that an underlying location infrastructure is available. The infrastructure is expected to at least be able to provide the symbolic co-ordinates of the location of an object, for example 'Room L10.01' or 'Ric's Desk'. Symbolic co-ordinates are commonly supported by location systems [40].

The model presented here extends this common notion of a symbolic location co-ordinate to include a set of location types, similar to those employed by Look

et al. [46]. These types indicate the category or function of the location. For example, the symbolic co-ordinate 'Room L10.01' may have the types 'Meeting Area' and 'Room', while 'Ric's Desk' would have the type 'Desk Area'. A full description of the location types and the prototype location detection system that were developed can be found in Appendix A.

Therefore, in the expressions of the 'Speaker' and 'Audience Member' role specifications, the location type 'Speaker Area' refers to the space within the room in which the speaker would typically stand, and the location type 'Audience Area' refers to the space in which the audience would typically sit.

### 3.5.4 Situation specifications

Now, let us look at the situation specification, which is illustrated in the bottom half of Figure 3.4. A situation specification has several parts - a 'Describes' property, an 'Area of Influence', a number of role specifications, an expression, feature bindings, and two sets of resource requirements metrics.

A situation specification declares the type of the situation that it describes using the 'Describes' property. The specification in Figure 3.4 describes the 'Presentation' situation from Figure 3.3.

A situation specification also includes one or more roles. The example in Figure 3.4 shows only the 'Speaker' and 'Audience Member' roles, though all of the roles defined in the 'Presentation' situation, as shown in Figure 3.3, are given in the full specification listing in Section B.2.2 of the appendices. It is the sets of entities that play these roles that are referenced in the situation specification's expression. Role specifications are matched to a situation specification based on their role type, and the same role description can be shared among several situation specifications.

An expression property is also defined for a situation specification, and is declared similarly to the role expression of the role specification. The expression may include the same set of operators, relations and patterns as a role specification. The situation expression may refer to the properties of any of the primary entities of the roles it contains. When a constraint is made on the properties of a primary entity of a role, it applies to all of the set of entities that are currently playing that role. For example, when the presentation specification states that the person playing the 'Audience Member' role must be in the same room as the display, it applies to all instances of people playing the 'Audience Member' role.

Furthermore, the situation specification's expression may refer to the size of the set of entities playing each of the roles it contains. For example, within the presentation specification's expression, it is possible to state that there must be at least three audience members. All of the parts of the expression in the situation specification hold when the situation is occurring in the environment.

A feature binding simply binds an expression or property referenced in the specification to a feature of the situation. These can be thought of as the outputs of the specification. In Figure 3.4, the feature bindings connecting the 'Speaker' role specification to the situation's 'Speakers' feature and the 'Audience Member' role specification to the 'Audience Members' feature are shown. Note that a feature binding respects the structure of the hierarchy between features. For example, there is no need to bind the 'Speaker' or 'Audience Member' roles specifications to the situation's 'Presentation Attendees' feature. This is done automatically, as both the 'Speakers' and 'Audience Members' features are subclasses of the 'Presentation Attendees' feature.

## 3.6 Incorporating uncertainty

So far, only expressions that result in crisp Boolean values have been considered for specifying situations. In a pervasive environment, many properties will be captured using sensors that may be limited in their accuracy and reliability. This will affect the level of confidence that can be had that the value of the property is correct, and whether a situation based on these properties is really occurring. Even for properties that are not sensed, factors such as the passing of time may alter the confidence that their value is correct. To effectively incorporate such properties into situation specifications, their level of confidence must be interpreted appropriately.

For properties such as these, an associated confidence value is defined. This is a real number ranging from 0 to 1, indicating no confidence to complete confidence that the value is correct. The confidence of a property may be fixed and known a-priori, for example it may be specified in the manual of the sensing equipment, or it may be estimated dynamically, which may be based on other factors about the property such as its freshness or source.

There are two main methods of incorporating uncertainty - probabilistic techniques such as Bayesian networks [22, 71, 77] and fuzzy logic [23]. This work employs fuzzy logic as an appropriate framework to incorporate and combine

the confidence values of properties in a situation specification. This method was chosen in preference to probabilistic techniques for the following practical reasons:

**Ease of translation** As will be demonstrated later in this section, situation specifications can be easily translated into a set of fuzzy rules. The resulting rule set is of equivalent size to the specification - approximately one rule per role and one antecedent per expression. The process of translating a specification into a Bayesian network is difficult as representing ad hoc groups of people and devices and the relations between them cannot be accommodated naturally by the fixed topology of a Bayesian network. Attempts to create a probabilistic representation of situation specifications result in Bayesian networks whose topology must be restructured at runtime to incorporate the varying number of people and devices in the environment, and in which the number of nodes increases exponentially with the number of people, devices and properties.

**Efficient reasoning** Situation determination attempts to match descriptions of several situations against the properties of an arbitrary-sized group of people and devices. This is an instance of the many pattern/many object pattern match problem, for which the Rete algorithm is a very efficient solution [66]. Rete-based fuzzy rule engines have mature tool support and are freely available [98].

**Improved scalability** The size of a situation can be measured in terms of both the structure of its specification as it grows larger and more complex, and in the number of entities that are involved in the situation. As the number of elements that feature in a specification grows, the amount of data required by a probabilistic approach to calculate its full joint probability distribution becomes impractically large. While it is possible to reduce this by establishing independence between elements of the specification, this in itself is a complex process which may require extensive example data and experimentation to identify correctly [38]. Given these difficulties, practical approaches have used naive Bayesian classification [59, 22, 69, 62], in which all of the elements used to recognise a situation are assumed to be independent. In a situation determination system, the uncertainty associated with several properties may share the same source. For example, all objects within an environment may have their location determined by the

same location system. Assume that a party situation is defined as twenty or more people gathered in a specific room. Furthermore, assume that the location of each person is reported by the same location system at a confidence of 0.9, which is the maximum confidence supported by the system. Treating the locations as independent, under a probabilistic scheme the confidence value is interpreted as a probability and these are combined for each person's location by multiplying the probabilities together. Therefore, the maximum probability of a party situation occurring is $0.9^{20} \approx 0.12$. If the party was defined with thirty or more people the maximum probability would be $\approx 0.04$. As the number of people increases, the maximum probability that can be had in the situation occurring decreases, despite the fact the uncertainty stems from a single source. Under a fuzzy logic model, as shall be demonstrated later in this section, the maximum confidence that the party situation is occurring would be the minimum confidence of a person's location, which would be 0.9 independently of the number of people at the party. Furthermore, the same, simple set of combination operators can be used to recursively calculate the confidence of specifications of arbitrary size.

To illustrate how fuzzy logic can be incorporated, consider again the presentation example in Figure 3.4. A person can be playing the role of a speaker when their location contains the type 'Speaker Area'. This role can be represented as a fuzzy if / then rule where the expression forms the antecedent or predicate (if part) and whether the role is occurring or not forms the consequent (then part). The antecedent and consequent are represented as fuzzy sets. These fuzzy sets are defined by monotonic functions that map the level of confidence to the degree of membership in the fuzzy set, and take the form $\mu_{FS} \leftarrow (c \in C)$ for the fuzzy set $FS$ and the confidence value $c$ over the domain $C[0, 1]$. Each role and each expression a role contains is represented by its own fuzzy set. For every fuzzy set, the domain will be confidence and the membership function will be $\mu_{FS}[c] \leftarrow c$, that is, the level of confidence is equivalent to the degree of membership.

As all fuzzy sets are defined by monotonic functions, simple monotonic reasoning can be used. This has the advantage that a role's expected confidence value can be estimated directly from the confidence of its expressions without employing complex composition and defuzzification methods [23].

If $ec$ is taken to be the confidence that person p's location contains the type

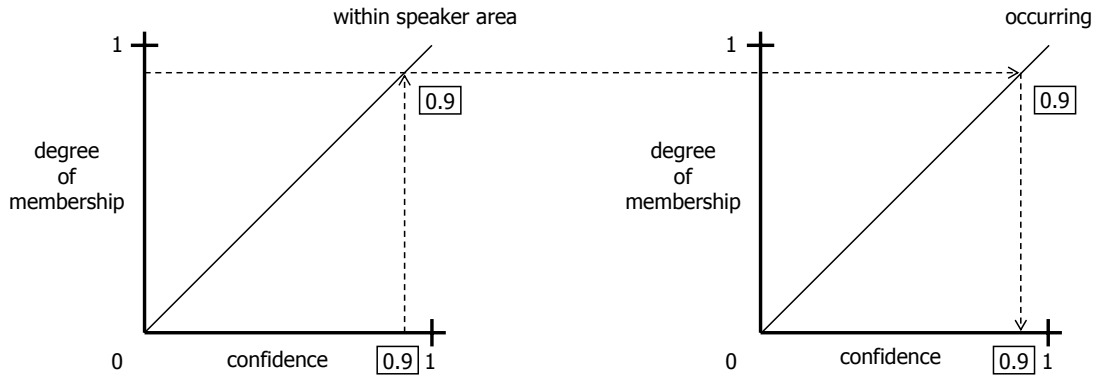



Figure 3.5: An example of determining the confidence value of a consequent fuzzy set using monotonic selection.

'Speaker Area' and $rc$ to be the confidence that person p is playing the speaker role, and have the expression represented by the fuzzy set 'within speaker area' and the role presented by the fuzzy set 'occurring', the following fuzzy rule can be constructed:

`if` $ec$ `is within speaker area then` $rc$ `is occurring`

Given this fuzzy rule and the confidence value of $ec$, the confidence value of $rc$ can be inferred by using a method of implication known as monotonic selection. Under this scheme, $\mu_{FS}[c]$ is calculated for the antecedent and the consequent has the confidence value that has the equivalent degree of membership as $\mu_{FS}[c]$, that is, $c_c \leftarrow \mu_{FS_c}[\mu_{FS_a}[c_a]]$, where $c_c$ and $c_a$ are the confidence values and $FS_c$ and $FS_a$ are the fuzzy sets of the consequent and antecedent respectively.

An example for the speaker role is shown in Figure 3.5. The confidence that person p's location contains the type 'Speaker Area' is 0.9. The result is $\mu[0.9] = 0.9$. This 'carries over' to the occurring fuzzy set and is translated to a confidence value of 0.9.

More complex roles such as the presentation specification in Figure 3.4 that have several expressions will be represented by a fuzzy rule that has several antecedents. Therefore, it is necessary to have a way to combine multiple degree of membership values. Antecedents may be combined using the fuzzy intersection operator ($\wedge$) where the minimum degree of membership is selected, or the fuzzy union operator ($\vee$) where the maximum degree of membership is selected. The confidence values of each of the expressions in the presentation specification will

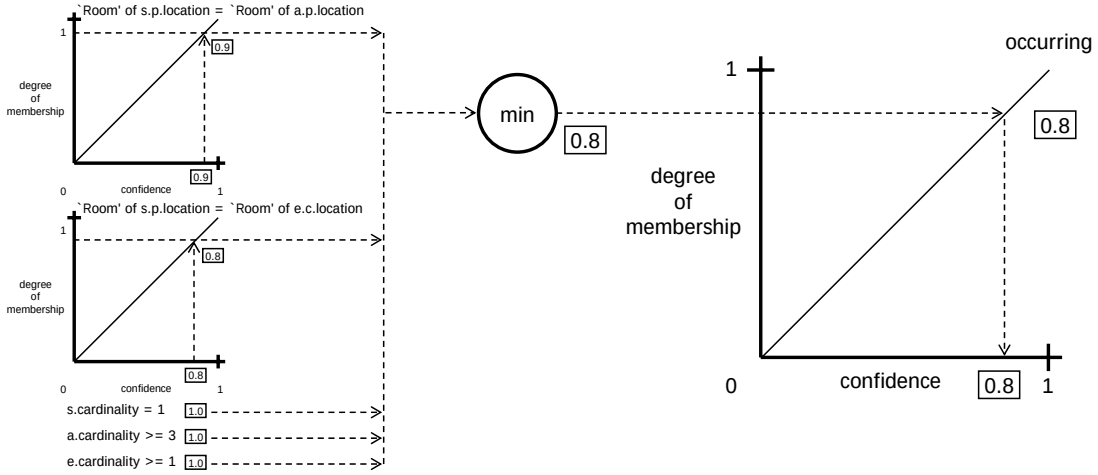



Figure 3.6: An example of combining confidence values.

be combined by fuzzy intersection. Expressions that do not involve uncertainty, such as the cardinality expressions, are treated as having a confidence value of 1 when true and 0 when false. Figure 3.6 illustrates how the confidence values of the presentation specification are combined.

Note that the meaning of $\neg$, $\Rightarrow$ and role cardinality also changes when considering confidence. The $\neg$ operator returns the complement of a confidence value, $1-c$. For a minimum confidence threshold $t$, $a \Rightarrow b$ is interpreted as "if a has confidence $\geq t$, then $b$", and $role.cardinality \geq X$ is interpreted as "role is occurring X or more times simultaneously, each with a confidence $\geq t$".

The confidence of the whole specification is the confidence assigned to an instance of the situation as a whole. It is tempting to think that as the properties or roles that may be mapped to individual features of a situation can have a higher confidence than the situation as a whole, it would be useful to expose the individual confidences of each of the features to permit more fine-grained decision making in situation-aware applications. However, giving this issue a little more consideration reveals that it does not make sense.

The issue is related to independently detecting occurrences of a role. Take for example, a situation-aware application that wishes to be notified when a particular person is the speaker of a presentation with a confidence of 0.9 or higher. That particular person may be detected as a speaker as the person's current properties match the speaker role's expression, say with a confidence of 0.92. However, at this point, the person is only a speaker in a provisional sense. If the actual situation is a party, and perhaps many people are standing in a

speaking area, or have been talking for most of the time, then at this point, there are several people that are provisionally a speaker. The particular person only becomes a speaker 'for real' in the context of the rest of the situation, when all of the other expressions of the situation specification also hold. Therefore, when the expression of the speaker role holds and a presentation is actually occurring, it is when the speaker role expression is a smaller part of the single, bigger, combined expression of the full situation, and it is this bigger expression that currently holds. Assume that the confidence of the overall situation is 0.86. Therefore, the confidence of the role expression when it holds 'for real' is the confidence of the full situation expression. So in this example, it is 0.86, and not 0.92. This is true for all roles and features. Therefore, it does not make sense, and is misleading, to separate out individual feature's confidences.

Note however, that it is possible to fuse the confidences of separate situation reports. Suppose that a presentation is recognised using the fine-grained location-based specification, and it reports a confidence of 0.84. However, the presentation can simultaneously be recognised using another specification. Suppose that the presentation is also recognised using the coarse-grained location-based specification, and it reports a confidence of 0.96. For the speaker in the presentation, if an application requires to know that they are the speaker, then the maximum confidence it can have is 0.84. However, if the application only requires to know whether the person is an attendee or not, then the maximum confidence it can have is 0.96. It is by recognising the same situation in different ways using separate specifications simultaneously, and fusing the results, that the overall confidence for the situation can be increased.

Just as situations and context information form layers, where a situation is a higher-level interpretation of the lower-level context information, the uncertainty associated with situations and context information also form layers, where the uncertainty associated with situations is a higher-level interpretation of uncertainty associated with context information.

At the context layer, there may be several context quality parameters that influence the overall confidence for a particular type of context information [99, 100]. Different quality parameters will be appropriate for different types of context information. For example, a report of the location of a particular entity may have accuracy and precision quality parameters associated with it. These would indicate the frequency and range within which the report can be expected to be correct. However, these parameters may not be appropriate for context informa-

tion such as whether a particular application is running on a computer, which can be detected much more reliably. Here, freshness may be a more appropriate parameter. Note however that the means to reliably calculate a freshness parameter may again depend on the particular type of context information. In each case, the suite of parameters used ultimately pertains to the overall confidence in the individual piece of context information at a particular point in time [99].

The situation layer focuses on how uncertainty demonstrates itself at this higher level, which is in the overall confidence an application can have that a report of a particular instance of a situation is occurring, and for how long that report can be considered valid. Therefore, when calculating the overall confidence and validity of a situation, it is based upon the confidence and validity period of each of the individual pieces of context information that comprise the situation. This creates an upward connection between the two layers. A suite of context quality parameters may be used to determine the overall confidence and validity period of individual pieces of context information. These in turn are used to determine the overall confidence and validity period of a situation.

This section has shown that in this approach it is straightforward to incorporate confidence measures into a situation specification, which requires that only minimum confidence thresholds be additionally stated. It was also shown that confidence calculation is efficient via monotonic selection, and that the approach scales easily to large and complex situations through recursive application of the same, small set of combination operators. In conclusion, the approach provides a simple, suitable means of incorporating and combining uncertainty measures of the composition of context information that comprises a situation.

## 3.7 Pragmatic aspects

There are a number of pragmatic aspects that are important for recognising situations. These include bounding the amount of context information that is considered relevant to a situation, locating the situations that are sought to be recognised and specifying the resource requirements necessary to recognise a situation. Each of these aspects is explored in turn.

### 3.7.1 Area of Influence

A situation specification states an 'Area of Influence' (AoI). The AoI describes a physical boundary around the location of the entities that may have an influence on the situation. For example, the AoI of the presentation specification shown in Figure 3.4 is set to 'Room', as when a presentation occurs, the entities upon which the specification will be based–the attendees, the projection screen, a laptop hosting the slides, etc.–are likely to all be in the same room. The AoI will commonly be a location type such as 'Room', though it may also be set to a specific location for specifications tailored to a particular environment.

The AoI addresses a practical concern of bounding the amount of context information that is considered relevant to a situation. If a specification did not specify an AoI, the recognition process would have to consider every entity within the environment when attempting to determine the situation. Clearly, this would be very expensive, and will almost always be unnecessary, as situations are commonly confined to much smaller areas. By specifying the AoI, the recognition process knows that it only need consider entities that are within the AoI, reducing the computational effort required to recognise a situation.

### 3.7.2 Situation Index

As the approach to pervasive situation determination presented here supports inter-environment operation, it is important to be able to locate situations that are sought to be recognised, whether they may be occurring in the local environment or in an external environment.

Discovering a situation within a network of environments is a difficult task. Situations may have complex specifications and the set of situations that are occurring within the network will be large and constantly evolving. This creates a lot of information that the discovery process must cope with, and it may quickly become stale. These difficulties are addressed by transforming the problem into a much simpler one.

A situation can be considered from a set of explicit viewpoints. For example, a situation request may be concerned with the situations a particular person is involved in, that a particular device is involved in or the situations that are occurring at a particular location. In each of these cases, the viewpoint of the situation changes. The first takes the viewpoint of a person, such as for a request to determine if a particular person is currently involved in a meeting or not. The

second takes the viewpoint of a device, such as for a request that checks if a particular projector is currently free or if it is being used in a presentation or other situation. The third takes the viewpoint of a location, which would be used for requests such as to check whether a particular meeting room is currently available, or whether it is already occupied by another meeting.

The viewpoint for a situation request is set by selecting a particular person, device or location as the situation index for the request. Then, rather than search for an occurrence of a situation, the system has only to search for the person, device or location that takes the desired view of the situation. For example, when a user is interested in the situations of a colleague, the system can first discover the location of the colleague and then request their situations from the local environment.

The situation index is related to the Area of Influence (AoI), as the AoI of the specifications used to fulfil a request will be set to the appropriate type of the current location of the index. For example, if a specification's AoI had the location type 'Room' and the index were a person or a device, then the AoI would be set to the particular room that the index currently occupies. If the index were a location, then the AoI would be set to that location itself.

As the situation index is part of a situation request, rather than part of the situation, the multiple viewpoints can be supported dynamically and simultaneously without requiring any changes to be made to situations, their specifications or customisations. Furthermore, it is possible to share the results of recognising a situation request that is made from differing viewpoints, but that share the same AoI.

Details and examples of how to specify the situation index in a request and the situation discovery process are covered in Chapter 4.

### 3.7.3 Resource requirements metrics

In Figure 3.4, two sets of resource requirements metrics are shown, 'Minimum Resource Requirements' and 'Average Resource Requirements'. These metrics allow a variety of resource requirements measures to be expressed for the situation specification. They are used by the situation determination middleware at runtime to determine if and where a particular specification can be recognised. The 'Minimum Resource Requirements' set denotes the limits of the minimum amount of resources required for a specification and is used to judge if that particular speci-

fication is suitable for processing on a particular device. For example, a complex specification that consumes a lot of resources may not be suitable for use when it is a resource-constrained mobile device that is attempting to use it to recognise a situation. The 'Average Resource Requirements' set denotes average resource consumption estimates, and these are used to help distribute the recognition load within larger deployments that have dedicated infrastructure available to process situation recognition. For example, by using the typical resource requirements metrics to identify the average expected resources the specification will consume, the situation determination system can implement a best-fit assignment strategy with the machines that are used to perform recognition. Further details of these metrics and mechanisms are given in Chapter 4.

## 3.8 Summary

The situation modelling approach presented here was developed to fulfil the requirements of pervasive situation determination. Specifically, that the modelling approach supports:

**Customised situations** - It was shown how a situation could be customised to fit the needs of a particular individual or their environment by specifying simple constraints upon the features of the situation. This provides a straightforward means to enable end-user customisation, where the customisations are based on conceptual terms that are meaningful to the user.

Furthermore, as a customisation is based solely on the features of a situation, they can be applied by any situation-aware application in any environment without concern for which particular specifications are being used to recognise the base situation.

**Rich situation models** - It was demonstrated that the situation modelling constructs support the description and specification of high-level situations featuring ad hoc groups of people and devices. Rich descriptions of a situation can be created that capture a multitude of roles and features within a situation, at varying depths of granularity. Having this scope and flexibility allows the reactions to situations by the system and applications to more closely match the end-users needs through the greater precision it affords.

Also shown was how a specification author can exploit the expressiveness offered by the situation specification constructs, to naturally describe how to recognise high-level situations given various sets of low-level sensor data, and

that it was straightforward to incorporate confidence measures and reasoning about uncertainty into a specification.

**Alternative descriptions** - It was shown how alternative situation specifications are supported, and make recognition of a situation robust against changes in environment as well as the available sensing infrastructure.

The critical idea of separating the description of the features of a situation from that of how to recognise it, is well illustrated in this respect. Not only may the number and choice of specifications used to recognise a situation be dynamically adapted, but also the customisations applied to the situation may also change freely. It is this independence that allows the model to provide simultaneous support for the two dimensions of variability that will be encountered in a pervasive computing environment.

**Multiple viewpoints** - Also illustrated was that the model of a situation is not anchored to any particular viewpoint. A request for a situation may be from the perspective of any person, device or location featured within a situation, without changing the representation of the situation, customisation or its specifications.

Furthermore, it was demonstrated that situations are described independently of any particular situation-aware application, and so can be reused between many, serving as a common model and vocabulary for a situation. Presented later in Chapter 5 is how this was exploited to build up both general and domain specific libraries of situations and specifications and how it eases situation-aware application development, by allowing developers to simply reuse situations rather than having to re-implement them by themselves.

Each of these aspects combine to produce a situation modelling approach that is able to support the distinctive requirements of pervasive situation determination.

# Chapter 4

# A Pervasive Situation Determination Architecture

## 4.1 Introduction

This chapter presents the architecture of a middleware for pervasive situation determination. As well as providing architectural support for the situation modelling approach presented in Chapter 3, the architecture meets the distinct adaptable recognition, resource management, inter-environment operation and situation discovery requirements necessary to realise pervasive situation determination.

The chapter begins by describing a scenario in Section 4.2 that is used to illustrate various demands upon and features of the architecture throughout. A brief overview is then given in Section 4.3 of each of the separate parts that make up the situation determination architecture. Details of how each of the parts interact and collaborate to recognise situations are given, as well as how situation-aware applications make use of the architecture. Following this, focus is given to how each of the individual parts operate in Sections 4.5 through 4.9. Finally, Section 4.10 presents details of how the architecture supports continuous situation recognition over changes in location, network and type of environment.

## 4.2 Scenario

The scenario presented below illustrates some example uses of the pervasive situation determination architecture:

As a computer science researcher, John frequently presents technical talks. He has been embarrassed in the past by his mobile phone ringing in the middle of a presentation, as well as his screen saver appearing on the projection board and even his laptop computer shutting down. Still, John often forgets to switch these devices to the appropriate mode in the hive of preparation activity just before a talk.

Wishing to avoid such embarrassment in the future, John installed a situation-aware mode manager application to recognise when he is the speaker in a presentation and to automatically switch his mobile phone to a silent mode, and disable the screen-saver and power-saving modes of his laptop during this time.

On this particular occasion, John had arrived late to give a presentation. Flustered, John could not remember where on his laptop computer he had stored the copy of the slides that he was to present. He remembered that they were the same slides that he had presented to a partner research group at Glasgow University at the start of the week.

In order to quickly find the slides, John used a situation-enhanced file search tool to search for the slides. This tool records the situations that John is involved in and the files he accesses during these situations. John entered the specific type of situation, location and approximate time of occurrence as search criteria. The tool quickly displayed its record of the event, which pointed John to the location of his slides. Relieved that being unable to remember where his slides were stored did not further delay the proceedings, John began his presentation.

Meanwhile, John's wife Angela was at home and received a call from a friend, inviting her and John to the theatre that night. Wishing to call John to see if he would like to go, Angela used the situation-aware availability checker application on her mobile phone to see what situations John was currently involved in, and if he was currently busy or not. The application reported that he was taking part in a presentation, so Angela set the application to notify her when the presentation was finished, so that she could call him then without interrupting him.

Not only does this scenario illustrate uses of the pervasive situation determination middleware, but it also exemplifies the distinct architectural requirements of pervasive situation determination.

The scenario highlights both general situations such as a presentation, and also bespoke, user-specific situations such as the research group presentations at Glasgow University. Chapter 3 presented situation specifications and described how general specifications may be provided in libraries, targeting wide deployment across many environments. Situation customisations were also presented as a means for end-users to create bespoke situations, by setting constraints on the properties of the features of an existing situation.

In a given environment, many different people and devices may come and go. Mobile devices such as laptops, music players and mobile phones may carry their own custom situations and specifications with them, as well as several customisations that many different people have created. The situation determination middleware should allow this diverse collection of situations, specifications and customisations to be efficiently integrated and recognised within the environment.

Different devices that enter an environment potentially introduce new data sources and additional sensing technologies. The situation determination middleware should be able to dynamically incorporate the new information that these data sources and sensors provide, in order to make the broadest possible range of specifications available. Where alternative specifications have been provided that vary in the situation's granularity, the situation determination middleware should exploit this new information in an attempt to recognise the highest possible granularity specification. This should happen not only for new situations that occur, but also for situations that are already taking place.

The middleware must also handle any uncertainty associated with the data and inference mechanisms that appear in specifications. That is, it must incorporate varying levels of confidence of the data it uses to recognise a situation, and also in reporting situations to applications. Given that several alternative specifications may exist for the same situation, as well as several alternative means to compute the same property of a person or a device, the middleware should be able to capitalise on this, and fuse together the results of the specifications and multiple sensing technologies, in order to attain the highest confidence possible for a situation.

The combination of these factors clearly demonstrates the requirement for a pervasive situation determination system to support adaptable recognition.

The scenario above illustrates the use of the situation determination middleware in a work environment and in the home environment. When Angela checked on John's situations he was giving a presentation, but he could have equally been in a more public environment such as a train station or a café.

This simple use of the middleware demonstrates the need for a number of architectural requirements. The first is the need to support situation discovery, as situation-aware applications must be able to discover situations that are occurring at potentially unknown places. It also highlights the need for the middleware to support use in private, administered environments that may have dedicated infrastructure to run the middleware, and also in public, non-administered environments where no dedicated infrastructure is available, as well as supporting the transitions between the two. The mode-manager application in the scenario highlights a situation-aware application that is present in the local environment, and reacts to situations that are occurring there. The situation-aware availability checker demonstrates an application that involves situations that are happening elsewhere in the world. The situation determination middleware must be able to support applications that exploit situations occurring both in the local environment and in remote environments. That is, it must support inter-environment operation.

At a more technical level, the design of the middleware must consider that it will run on resource-constrained, wireless, mobile devices. This impacts the design in a number of ways. As the devices communicate over wireless networks, the middleware must handle intermittent connectivity, as well as devices disappearing from the current network. The middleware must also be able to cope with device failure, where failure may simply mean that the device's battery has run dead.

It is well known that wireless communication is the biggest source of battery drain for mobile devices [101, 102, 103]. Studies have shown that mobile devices can consume as much as 70% of their battery while connected in idle mode to an 802.11 wireless network [104]. Furthermore, under realistic power consumption scenarios, the power consumed by wireless transmission can be anywhere from 6 to 50 times that consumed by the CPU or memory [103, 105, 106, 107].

In order to preserve battery power, it is desirable to have the middleware support *resource foraging* [108, 109]. That is, to allow devices to offload processing to other, possibly mains-powered devices. Furthermore, it is also advantageous that the middleware support placing the recognition of particular situations on specific devices, in order to reduce the level of wireless communication required.

These considerations clearly show the requirement for resource management in a pervasive situation determination system.

The scenario illustrates not only the individual requirements of adaptable recognition, resource management, inter-environment operation and situation discovery, but also how these requirements intertwine. This chapter presents an architecture for pervasive situation determination that is able to provide support for these requirements in combination.

The architecture is designed to be extensible. It is not intended that it represents a single, exclusive system used in an environment. It is a pervasive architecture, intended to be used in many and in many different types of environments, but as noted in Section 1.1, a "one-size fits all" approach is not desirable.

In each of the three layers of a situation determination system – the sensor data layer, the context layer and the situation layer – an environment may already have existing systems in place. For example, it may be fitted with particular sensor infrastructure, it may feature one or more location sensing systems, there could be an existing context information dissemination system that supports actively used context-aware applications, there may even be specialised situation recognition components. The goal would not be to replace these, but to make it possible to overlay the architecture and integrate these components, in order to provide a pervasive situation determination system.

## 4.3 Overview of the architecture

This section presents a general overview the of the pervasive situation determination architecture by walking through the scenario involving John and Angela presented above. It introduces the high-level aspects of the architecture that are involved in each stage.

Figure 4.1 presents the scenario pictorially. Its shows Angela at home and John at the University. Both of these environments are connected to a network of pervasive computing environments that support pervasive situation determination. Both environments have dedicated infrastructure that hosts the pervasive situation determination middleware and manages this connection.

Let's first look at the case where Angela uses the situation-aware availability checker application from her mobile phone to see what situations John was currently involved in, and if he was currently busy or not. This will provide an overview of a single, complete round of the pervasive situation determination

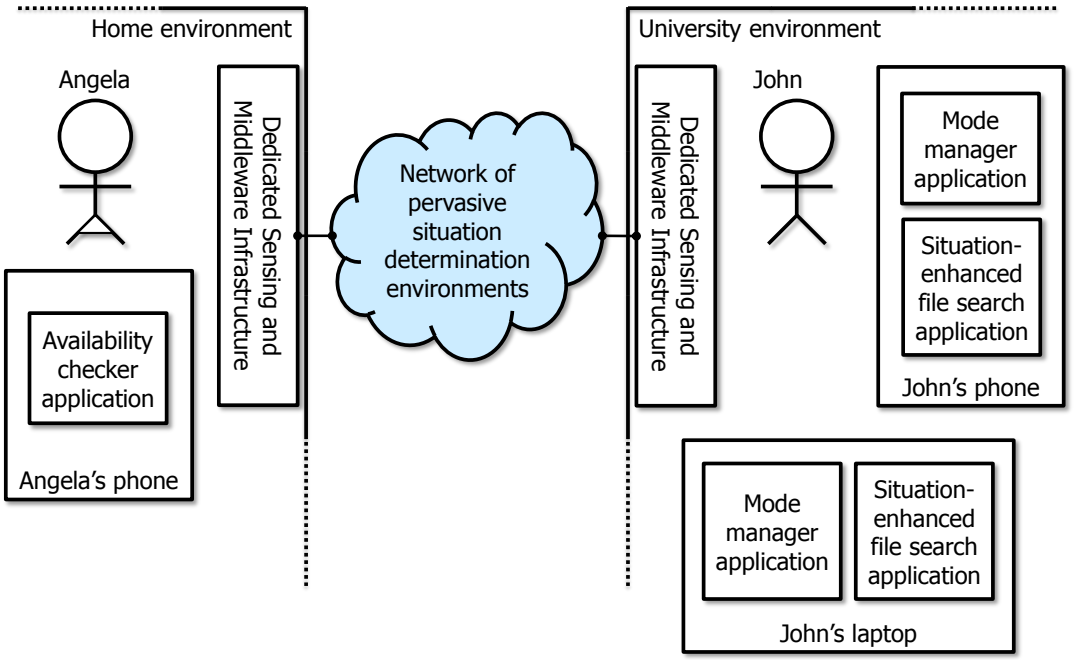



Figure 4.1: This diagram represents the scenario where Angela uses the availability checker application on her mobile phone to check which situations John is currently involved in.

process.

The first step in this process is that Angela uses the application and selects to check the current situations that John is involved in. The application connects to the host infrastructure in the home, which then searches the network of environments to locate John. Specifically, it is John's mobile phone that will be located, as this device currently acts as a representative for John. The availability checker application on Angela's phone then sends a message to John's phone, requesting that it report the situations that John is currently involved in.

Having checked the request, John's phone will forward the request to the host infrastructure in the University environment to begin recognising the situations John is involved in. When a situation is recognised, the host infrastructure sends the report to John's phone which in turn sends the report back to Angela's application. First sending the report to John's phone, rather than to Angela's application directly, allows John the opportunity to configure the phone to obfuscate or otherwise transform the report before sending it back to the application that issued the request.

It will not be just a single report that is sent to the application, but a stream of

reports, for as long as the request is maintained by the application. Recognition of a situation is based on context information, which in turn is derived from the sensor data currently available in the environment. The sensor data, context information and therefore the confidence with which the system can detect which situations are occurring, will be continually changing. These changes are captured in the stream of reports.

This highlights the query-based nature of the architecture. That is, the middleware does not attempt to recognise a situation until an application makes a request for it. Furthermore, the middleware will cease attempting to recognise a situation when there is no longer an active request for it.

It is in this respect that it is important that a situation can be recognised within a short temporal extent. This allows, as in this case, for a request to be issued in one environment, the index of the situation to be located, recognition of the situations referenced in the request to be started and performed in the index's local environment, and a report sent back to the issuing application, all within a suitably short time-frame to support fluid, interactive use in applications.

Not all situation requests will span different environments. The mode manager and situation-enhanced file search applications running on John's mobile phone and laptop provide examples of situation requests that are processed locally. The requests from these applications will be processed in a similar way to the request made by Angela's availability checker application, with the exception that the middleware will not search the network of environments as the middleware will already be aware of John's presence locally.

The situation requests issued by the mode manager and situation-enhanced file search applications John is running will be longer-lived that the request issued by Angela. These applications continually monitor the situations John is involved in for as long as John is running the applications. This requires that these requests persist as John changes location and environment.

The results of recognising situations can be shared between multiple requests for them. For example, in the scenario, when Angela's request arrives, the applications John is running have already requested the situations John is currently involved in. So when Angela's request arrives, the middleware does not need to recognise the situations again, but simply add Angela's availability checker application to the list of recipients of the reports for the situations John is currently involved in.

As noted above, when Angela checked on John's situations he was giving

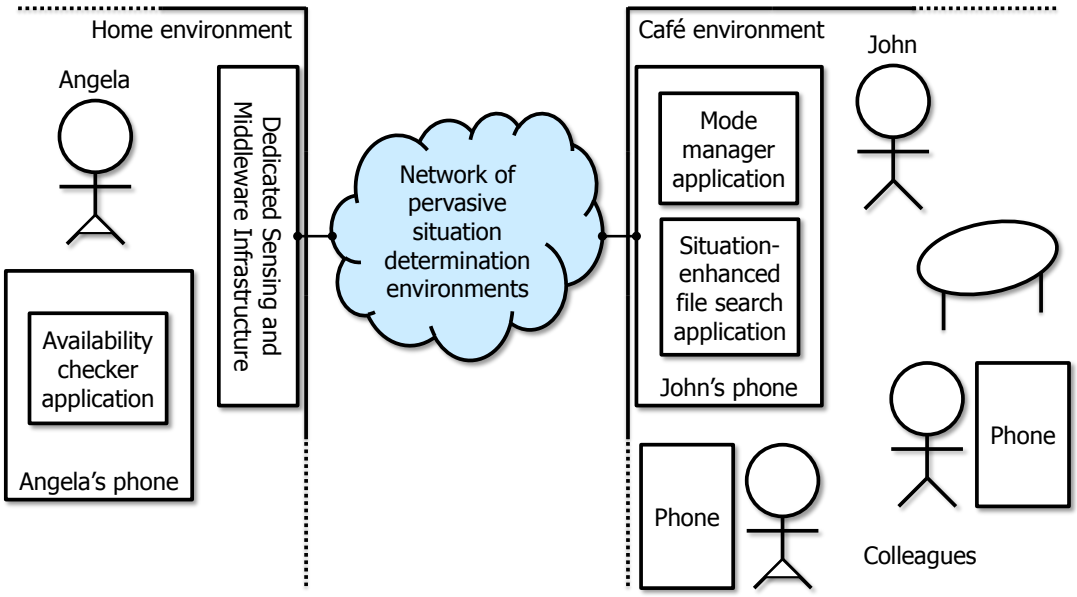



Figure 4.2: This diagram represents the scenario where Angela uses the availability checker application on her mobile phone to check which situations John is currently involved in, but this time John is located in a café where no dedicated infrastructure is available.

a presentation at the University which had dedicated infrastructure available to host the middleware, but he could have equally been in a more public environment where such infrastructure is not available.

Imagine a slightly different scenario where rather than being at the University, John was enjoying some discussion with colleagues at a nearby café. This alternative scenario is illustrated in Figure 4.2.

In the café there is no dedicated infrastructure available to host the pervasive situation determination middleware. Therefore, situation recognition must be performed by the ad hoc collection of devices themselves. To do this, the devices must form a network to allow the context information they can detect to be shared amongst each other. Using this context information, the devices would then recognise situations themselves. Note that in Figure 4.2, John's phone connects directly to the network of environments, allowing John to be located from external environments, despite his phone operating in this ad hoc mode.

When operating in ad hoc mode, situation requests are processed as before - John's phone would receive the request from Angela's application, and a stream of reports would be sent back as the situations referenced in the request are

recognised. Furthermore, it is still possible to process local situation requests in an ad hoc network, as can be seen from Figure 4.2, where John's phone is still running the mode manager and situation-enhanced file search applications.

This concludes the brief overview of the high-level pervasive situation determination architecture and the situation request, recognition and response process, shown for both infrastructure-based and ad hoc modes of operation. The following sections present further detail about how this architecture is realised in the pervasive situation determination middleware.

## 4.4 Agent architecture

The situation determination middleware that has been described so far has several distinct characteristics that must be supported by its architecture. It is an open system, as it must incorporate a variety of people and heterogeneous devices, the number and identity of which may not be known in advance and will change over time. The data describing the properties of people and devices, as well as new and customised situation specifications, are inherently distributed. Recognition of situations is a responsive process, as it must continually monitor changes in the environment and report the situations occurring. Situation-aware applications are often adaptive, tailoring their behaviour to the current situation. Both recognition of situations and adaptation of application behaviour must be performed autonomously. Given these characteristics, an agent-based architecture is appropriate [110].

This section presents a high-level overview of the agent architecture of the pervasive situation determination middleware. It covers typical set-ups for use in private, administered environments and briefly introduces the main features of the architecture (set-ups for public, non-administered environments are covered later in Section 4.10).

First, let us look at the main function of each of the types of agent used by the situation determination middleware, before going on to look at how they work together to recognise situations. As a key and summary, Table 4.1 provides a list of each of the agents mentioned in this chapter, as well as their acronym and a brief description of their function.

The situation determination agent (SDA) is the agent that actually performs situation recognition. It continually attempts to match situation specifications against the current state of the people and devices in the environment, and report

| Acronym | Name | Function |
|---|---|---|
| **SDA** | Situation Determination Agent | Performs core recognition of situation specifications |
| **CEA** | Context Entity Agent | Provides context information about people/artefacts in the environment |
| **DCEA** | Data Context Entity Agent | A CEA which reports context data |
| **CCEA** | Compute Context Entity Agent | A CEA which dynamically computes context information |
| **CCEAM** | Compute Context Entity Manager | Manages requests to CCEAs |
| **SRA** | Specification Repository Agent | Stores descriptions of situations, specifications and customisations |
| **SAA** | Situation-aware Application Agent | Base agent from which situation-aware applications are derived |
| **ISA** | Index Server Agent | Manages situation requests for a single person/artefact |
| **ILA** | Index Locator Agent | Locates a particular person/artefact in the local/remote environment |
| **SDAM** | Situation Determination Agent Manager | Manages recognition process between multiple SDAs |
| **EMA** | Environment Manager Agent | Manages transitions between environment and ad hoc operation |
| **PSA** | Publish/Subscribe Agent | Manages subscriptions/notifications of a publish/subscribe network |

Table 4.1: A key and summary of the agents used in the situation determination architecture.

instances of situations when they occur. In an administered environment, one or more SDAs may appear and these would typically be hosted on dedicated infrastructure for providing situation determination functionality. However, as is shown later in Section 4.10, mobile devices may also host SDAs allowing situation recognition to be performed in a more ad hoc setting.

The current state of the people and devices in the environment are reflected by the current values of their properties, which are detected and reported by context entity agents (CEAs). There are two flavours of CEAs - data CEAs (DCEAs) and compute CEAs (CCEAs). DCEAs detect and report individual properties. CCEAs calculate relations between properties, detect patterns occurring in the values of one or more properties over time, or perform conversion of the value of a property to a different type. DCEAs and CCEAs can be thought of as signals

and signal processors, respectively. CEAs could be hosted on any device. For example, location, audio, application, and diagnostic sensor CEAs may be hosted on a mobile phone, and ambient light, temperature, audio, and motion sensors may form part of the fixtures in a smart environment, with their corresponding CEAs hosted by the situation determination infrastructure. If there exists several similar or equivalent CCEAs in the environment, the CCEA Manager (CCEAM) is used by the SDA to select the best choice of possible CCEAs based on the current runtime state of the CCEA host device.

Specification repository agents (SRAs) store collections of the specifications that are used by an SDA to recognise situations. An SRA may store customisations as well. SRAs would typically be hosted on an environment's infrastructure to store the situations, specifications and customisations particular to the environment, and also on an individual's personal devices, storing the situations, specifications and customisations that are particular to them.

A situation-aware application agent (SAA) is the main interface to the situation determination middleware for situation-aware applications. An SAA manages an application's situation requests and reports occurrences of the requested situations back to the application. Any device that runs a situation-aware application will host a local instance of its SAA.

An index server agent (ISA) maintains a record of which SAAs currently have requested situations that involve a particular person, device or location. That particular person, device or location forms the index of a situation request, as it is the situations involving that particular person, device or location that are sought to be reported. For example, if a user requests the situations of a particular person, then that person forms the index of that request. Likewise, if an application were to request the situations occurring at a particular location, then it is that location that forms the index of the request. Each individual person, device and location will be represented by their own ISA. In the scenario, when John's wife requests that she be notified when John is no longer giving the presentation, the SAA on her phone will request notification of presentation situations with John as the index. John's ISA will maintain this request. The ISA allows subscription requests to persist as the index changes environment or network. Furthermore, the ISA can be configured to amend the situation report depending on the particular SAA instance that has requested it. ISAs that represent locations within a particular environment will typically be hosted on the environment's infrastructure. ISAs that represent devices will either be

hosted on the device itself, or on the environment's infrastructure where the device lacks sufficient capacity. An ISA that represents an individual will typically be hosted on a personal device belonging to that person, such as a mobile phone.

An index locator agent (ILA) performs the discovery of the index entity of a situation, whether the index is located within the local environment or a remote, external environment. As an SAA requires an ILA to issue situation requests, an ILA instance will be hosted alongside each SAA.

The environment may contain many SDAs and many SRAs. The function of the SDA Manager (SDAM) is to take situation requests, passed to it from ISAs, and first gather all specifications that describe the situation from the collection of SRAs. Then, the SDAM selects the best choice of SDA for each specification, based on the performance requirements of the specification, and the current runtime state of each SDA host machine. Similar to an SDA, an SDAM will typically be hosted on dedicated infrastructure within an administered environment, though as is shown later in Section 4.10, mobile devices may also host an SDAM for ad hoc operation.

The steps involved in how these different types of agents interact to perform situation recognition are illustrated in Figure 4.3 and represented in a UML sequence diagram in Figure 4.4.

First, an SAA will receive a situation request from a situation-aware application that states a particular index for the situation. For example, in the scenario, this is when John's wife requests which situations John is currently involved in. This is labelled as step 1 in Figures 4.3 and 4.4.

Next, using the ILA to find the ISA of the index entity in question, the SAA sends the ISA the situation request. At this point, John's ISA would have received the situation request from his wife's application's SAA. These steps are labelled 2 and 3 in Figures 4.3 and 4.4.

The ISA then adds the request to the set of active situation requests that it maintains, and then forwards the request to the SDAM. The SDAM then analyses the request to determine which situations are being requested, and then contacts each SRA, asking that it be sent any specifications the SRA has that describe the particular situations. As the SDAM receives each specification, it passes it to each of the SDAs in the environment (first contacting dedicated SDAs if they exist), and requests a resource requirements estimate from the SDA that indicates its current capacity for performing the recognition of the specification. The SDAM then selects the SDA that provides the best estimate. These steps are labelled 4

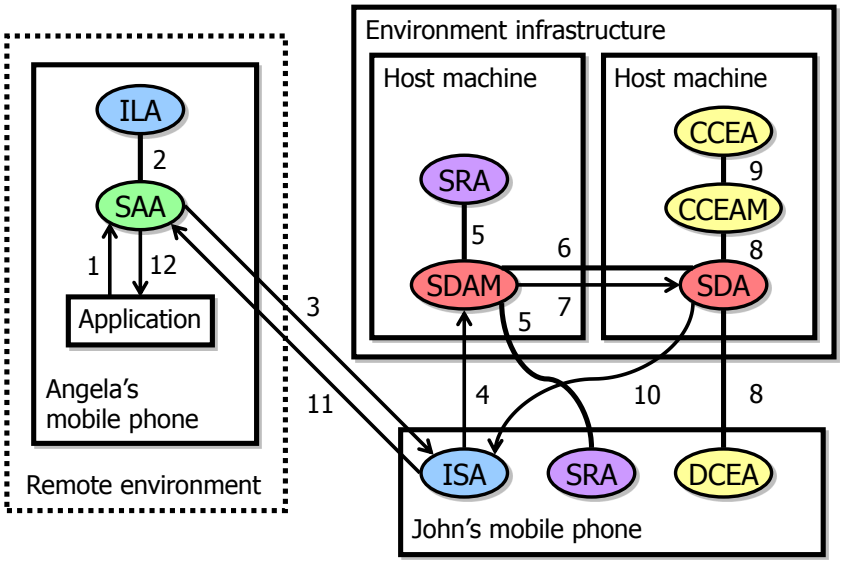



Figure 4.3: This figure illustrates the different types of agents involved in the situation determination middleware, as well as the steps involved in the basic situation recognition process. For clarity, not all agents that would be hosted on a device are shown, only those that help illustrate the process.

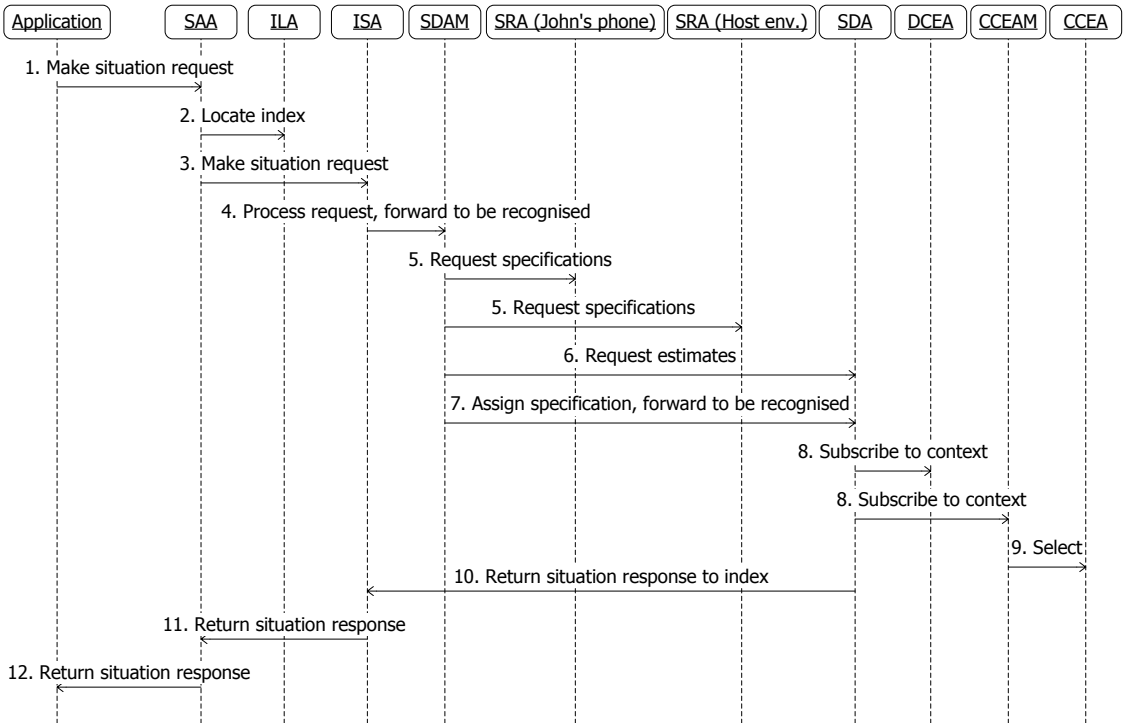



Figure 4.4: This figure shows the steps performed in Figure 4.3 represented as a UML sequence diagram.

through 7 in Figures 4.3 and 4.4.

Having been assigned a specification from the SDAM, an SDA analyses the specification and determines which properties, relations, patterns and conversions it requires to recognise the situation. DCEAs use a publish / subscribe style of communication [111] to transmit properties. As such, the SDA sets up subscriptions to the properties it requires. The SDA will also use the CCEAM to employ the CCEAs it requires. When the SDA detects that a situation is occurring, such as John's presentation, it sends a report to the index's ISA, which then passes the report back to the SAA, which in turns passes the report to the application. These steps are labelled 8 through 12 in Figures 4.3 and 4.4.

This completes a single round of the situation determination process. The description here has been intentionally brief, giving an overall flavour of the complete process. Though many details have been omitted here, each type of agent is presented in full in the following sections.

## 4.5 The Situation-Aware Application Agent

The situation-aware application agent (SAA) is the main interface for situation-aware applications to the pervasive situation determination middleware. Using the SAA, situation-aware applications can request and receive notifications about occurring situations. The details of the interface provided by the SAA and the format of situation requests and responses are given later in Section 4.11.

## 4.6 The Index Server Agent

The index server agent (ISA) continually maintains a set of active situation requests for a particular person, device or location, which have been made with the particular entity as the index.

It is assumed that any person who wishes to have the situations they are involved in recognised by the situation determination middleware, carries with them a personal device that is capable of hosting an ISA. Most commonly, this device would be the person's mobile phone.

An ISA for a particular device will usually be hosted on the device itself. Devices that have insufficient capacity to host an ISA can be supported by proxy ISAs that can communicate with the device. The proxy ISA will be hosted

on another device that has sufficient capacity. A proxy ISA would typically be installed in an environment along with the device.

ISAs for locations will also exist within administered environments. When the middleware is operating in ad hoc mode, the location of detectable situations is centred around an individual person or a device, and so the index for such situations is the person or device themselves. Within an administered environment however, there will exist distinct, absolute locations that can be selected as an index. An ISA exists for each symbolic location within an environment. A location model for the environment will already exist, detailing each of the locations within the environment. Each ISA can then be created automatically using the location model.

As the ISA is hosted on a device that is representative of the device itself, the set of active situation requests can persist throughout intermittent wireless network connectivity, changes in location, network, and environment, as well as power and other failures of the ISA host device. The ISA will always have an up-to-date record of which situations are to be recognised for the particular index, and which SAAs are to receive notification about them, regardless of their location.

All situation requests made by a situation-aware application are first delivered to the index's ISA. The ISA records the request, and then forwards the request to the SDAM. Regardless of whether the middleware is operating in an administered environment, or in ad hoc mode, there will be a single SDAM in the environment that the ISA discovers through the agent substrate. Details of SDAM discovery are covered in Section 4.10.

## 4.7 The Situation Determination Agent Manager

The situation determination agent manager (SDAM) receives situation requests from any ISA in the local environment. Its function is to select an available SDA to recognise a particular specification for a situation request. The selection is based on a number of resource requirements parameters that reflect the current capacity of each SDA host and the suitability to the particular specification. Selecting an SDA in this manner allows the SDAM to control the load on each SDA host, dynamically adapting each to best meet the demands of the current

set of situation requests. Additionally, the recognition of some specifications may be *grounded* to a particular SDA. Each of these mechanisms is discussed below.

### 4.7.1 Grounded specifications

Before going on to look at gathering and preparing specifications in the next section, the concept of a grounded specification must first be introduced.

Recall from the introduction to this chapter that wireless communication is the biggest source of battery drain for mobile devices, consuming up to 50 times as much power than that of the CPU or memory.

Grounded specifications help preserve a mobile device's battery power by reducing wireless communication for certain specifications. There are situations for which their specification will be based entirely or largely on properties available on a single mobile device. Consider the simple situations 'Making a phone call' and 'Editing a document'. For both of these situations, all of the properties required to recognise the situation may be provided by CEAs hosted on a mobile phone or a laptop, respectively. As these situations are localised to a single device, it would be unnecessary and wasteful to have to transmit all of the properties required to recognise the situation to an SDA hosted on an external device. Instead, it is more appropriate to have these specifications recognised by an SDA hosted locally. That is, the specifications are *grounded* to the local device.

An SDA can be configured to recognise a grounded specification by indicating which specifications are to be grounded and for which indexes they apply. For example, the 'Making a phone call' situation could be grounded to John's mobile phone by registering the situation's specification and John as the index with the SDA hosted on the phone.

Configuring an SDA like this lets the user select which specifications are registered for a particular index and on which particular device it is to be recognised. This allows the user to tune the placement of the specification to their own particular habits. However, there may be times where the groundings the user has selected will not be optimal. If all of the pieces of context information required by the grounded specification are already subscribed to from external sinks, then recognising the situation locally may not create savings in battery power.

A middleware-controlled scheme could potentially manage such cases automatically. However, detecting whether the best placement of the specification is local or external is difficult, and may itself be expensive. It is possible that

the mobile host could analyse all subscriptions the local PSA currently maintains and determine if all context information required for a specification is already subscribed to, and therefore the specification may be best recognised externally. However, this creates significant extra work for the mobile host to perform, as all subscriptions and a potentially large set of specifications must be analysed continuously. Furthermore, the cost of the analysis and performing the recognition locally must be balanced against the additional cost of transmitting the specification.

In favouring simplicity, this approach allows the user to control which specifications are grounded to which device (which could be recommended by the specification authors), which may frequently allow the mobile host to make power savings, and eliminates the need to perform continuous subscription and specification analysis.

### 4.7.2 Gathering and preparing specifications

In order to select an appropriate SDA, the SDAM must first gather all specifications and customisations that relate to a situation request. To do so, the SDAM indicates interest in a particular target by registering the target's identifier with each specification repository agent (SRA) in the environment. An SRA will then send any appropriate specifications and customisations it contains to the SDAM. An SRA will also send any other appropriate specifications and customisations that are subsequently added while the SDAM is registered for the particular target. The SDAM indicates that it is no longer interested in a particular target by unregistering the target's identifier.

The SDAM must also handle the special case of grounded specifications. Here, a specification is tied to a particular SDA instance, and therefore grounded specifications are not forwarded to the SDAM. In this case, grounded specifications are registered with the SDAM explicitly by the SDA they are grounded to. The SDAM then selects this SDA directly when one of its grounded specifications match the target and index of a request.

Once the SDAM has gathered the specifications, the area of influence (AoI) for each specification must be checked, and transformed if necessary. First, the SDAM subscribes to the location property of the specification's index. The SDAM then processes each specification in turn. If the specification has an AoI defined by an absolute location, then the SDAM checks whether the index is within the

absolute AoI. If it is not within the absolute AoI, the specification is removed from consideration. If the AoI is defined by a location type, then the SDAM transforms the AoI into the symbolic location of the index's location that matches the location type. Note that the transformation is dynamic - as the index changes location, so too will the transformed AoI.

There is a special case for transforming an AoI when the index is a location. It may be the case that the index location contains more than one instance of the AoI's location type. For example, if the AoI is a room and the index location is a floor of a building, then the floor may contain many rooms. In this case, the AoI is transformed several times, once for each location of the AoI's type that is contained within the index location.

Now that the specifications have been prepared, they are ready to be forwarded to the available SDAs.

### 4.7.3 Resource requirements estimation

The SDAM requests resource requirements estimates from each situation determination agent (SDA) for an individual specification. The SDA that returns the best estimate is assigned the specification. In this way, the SDAM can manage the distribution of load between the available SDAs.

When selecting which resource requirements parameters to use, their number must balance the level of detail required with the increased level of associated overheads. Dijk et al. demonstrate that a small number of parameters, usually less than or equal to 5, is sufficient to capture the dominant properties of a system [112]. Furthermore, Aurrecoechea et al. suggest that parameters should be declarative in nature. That is, they should specify only what is requested, and not how the requirements should be implemented by a provider [113].

Influenced by this work, the framework proposed in [114] is extended in this chapter to include the following resource requirements parameters:

*Distance* - this parameter indicates the distance from the situation to the SDA. This parameter allows administrators to assign an SDA to particular areas. For example, if a particular room plays host to a large number of situations at once, or to a situation that frequently features a large number of people, then an administrator may wish to assign an SDA primarily for recognising situations in this room. The value of this parameter will range from 0.0 (best) to 1.0 (worst). The administrator can configure *coverage areas* for an SDA. A coverage area

states the locations that an SDA prioritises. A specification's AoI is within the coverage area if the AoI's location is contained completely within the locations of the coverage area. An administrator will specify a coverage area and a related distance value. For example, to give a particular room priority for an SDA, the coverage area would be defined as the room and the distance value of 0.0. Any other location would receive a distance value of 1.0. A specification whose AoI's was contained within the room would have a distance value of 0.0, and 1.0 otherwise. An administrator may define more than one level. For example, distance values could be configured such that the room has a value of 0.0, the floor that the room is in has a value of 0.4, and any other location 1.0. Distance values may be configured on a per-SDA basis.

*CPU Load* - Choosing a suitable metric to express CPU load is difficult, as it must provide an estimate of the CPU load of a specification across a range of processor architectures. The approaches presented in [114, 115] address this issue by employing measures based on the amount of time consumed by previous executions of a service on a particular processor. Using such a measure is not suitable for situation specifications as it ties the estimate to a single processor, and requires that the execution time of a single service can be isolated. Though it is possible to learn execution times for pairs of processors and services, and use predicted execution times as is done in both [114] and [115], it is not feasible for situation specifications as they may be recognised by a scheme, such as a Rete network [66], which extracts commonalties from many specifications in order to optimise their recognition, which makes it difficult to isolate the execution time of a given individual specification. Instead, an approach similar to that proposed in [116] is preferred, using a more approximate measure of average number of clock cycles per second. The advantages that this offers are that individual specifications can be tested offline in isolation by specification authors to obtain consumption estimates, which then can be used across a variety of processors online in an environment.

Customisations will be written by end-users rather than specialised specification authors, so it is impractical to assume that they could empirically establish resource requirements measures for their customisations. Furthermore, as customisations are simple constraints on the values within a feature set, it is unlikely that the additional overhead incurred by processing a customisation significantly alters the resource requirements of the base specification. Therefore, customisations do not state resource requirements, and customisations are not considered

when calculating the resource requirements estimate for the base specification.

A situation specification will state its CPU Load requirement in average number of clock cycles per second. The resource requirements estimate for CPU Load given by an SDA is a percentage, representing the percent of the total available CPU utilization that the specification would consume, where the available CPU utilization is measured as maximum clock cycles per second multiplied by the percent of the current CPU utilization. For example, if the average number of clock cycles per second consumed by a specification is 200 million cycles, and the SDA host's processor has a maximum clock cycle per second rate of 3,000 million cycles and the processor currently has 50% utilisation, then the resulting estimate for CPU Load for the host is 200 million / (3,000 million * 0.50) $\approx$ 13%.

*Memory* - A situation specification states this requirement as the average maximum amount of physical memory consumed by the specification. An SDA host's estimate is calculated as the percent of the remaining available physical memory of the host that the specification would consume.

*Bandwidth* - A situation specification states its bandwidth requirement as the average amount of bandwidth consumed by the specification in bits per second. An SDA host's estimate is calculated as the percent of the remaining available bandwidth of the host that the specification would consume. Note that this single measurement acts as a summary of the available bandwidth across all active network types of the host. For example, if the host were a laptop, it may on some occasions be communicating exclusively via an 802.11 wireless network, while on other occasions, it may be utilising its Ethernet and Bluetooth connections. It is necessary to provide an approximate summary as it is impossible to predict ahead of time exactly which devices will be involved in the recognition of a specification and on which type of network they will be communicating with the SDA host.

*Battery* - This requirement is stated in a situation specification as the average amount of battery consumed by the specification in watts. An SDA host's estimate is calculated as the percent of the remaining available battery of the host that the specification would consume. If the host is plugged into the mains, the remaining amount of battery is taken to be infinite.

The CPU Load, Memory, Bandwidth and Battery parameters were established in the original framework. The Distance parameter and the modification of the CPU Load parameter are introduced here.

In a similar approach to [114], each resource requirements parameter is combined to give a single overall value to allow easy comparison with other resource

requirements estimates. This is extended to allow the relative weighted importance of each parameter that the SDA host device places on each to be incorporated in this calculation. In this way, individual hosts such as a mobile phone can stress that using the least amount of battery and network is more important for that particular host than distance or CPU Load, for example. The overall resource requirements metric is calculated as:

$$\text{Overall resource requirements} = \sum_{i=1}^{n}(q_i * w_i)$$

where each $q_i$ represents a resource requirements parameter, and $w_i$ is the associated host's weight for that parameter.

A specification contains two sets of resource requirements parameters. The first is the set that has just been presented, which is used to estimate the average consumption over the lifetime of the recognition of the specification on a particular SDA host. The second set describes minimum resource requirements. This gives the SDAM a means to ignore specifications for which an SDA host cannot offer the minimum capacity required. In periods of heavy load the SDAM can relocate a specification to another, less utilised host. Furthermore, the SDAM is prevented from running a specification on a device that has an inappropriate capacity. For example, a high consumption specification that is intended to be run on a dedicated server will be ignored when considering an SDA hosted on a mobile phone. Similarly, if the middleware were to reach a saturation point where all available SDAs are fully loaded, no new situation requests would be started.

There may be a large number of SDAs available in the environment, and it is unnecessary to contact every SDA for a resource requirements estimate for every specification. For example, if an environment's dedicated SDA hosts are lightly loaded, it may not be necessary to contact SDAs hosted on a PDA. The middleware allows sets of SDAs to be staged, in order to reduce the number of SDAs that must be contacted for each specification. Each stage specifies a set of hosts to be contacted, described either by their identifier or by their properties, and a set of weights and threshold values for resource requirements parameters. For example, an administrator may configure the middleware such that only the environment's dedicated SDA hosts are contacted until the average resource requirements estimates from each host, weighted only on CPU Load, memory and bandwidth, exceed the threshold. Then, all mains-powered, fixed-network SDA hosts are contacted, until the threshold for this stage is exceeded, then any SDA may be contacted. The SDAM can then distribute load across many hosts in the

environment, calling upon additional hosts as necessary.

### 4.7.4 Situation Determination Agent selection

After a short time interval has passed, the SDAM will compare all of the resource requirements estimates it has received. The SDA that provided the best estimate is selected to recognise the specification. The specification is then forwarded to the SDA, along with any customisations that are required to be recognised in addition to the specification.

The resource requirements estimates for a specification may change throughout the time an SDA is attempting to recognise a situation. If the index of the situation is mobile, then the AoI of the specification will change as the index changes location. The load on the SDA effecting parameters such as CPU Load, Memory and Bandwidth will constantly be in flux. If the host is battery powered, then the remaining power will always be steadily decreasing.

In order to take these changes into consideration, the SDAM periodically re-evaluates the resource requirements estimates for each specification. If the estimate from the SDA a specification is hosted on is no longer the best, then the SDAM may relocate the specification to the host that provided the new best estimate. However, there will be a cost involved in performing the relocation of the specification. In addition, thrashing must be avoided - the case when two or more SDA hosts provide almost equal estimates, and the SDAM continually transfers the specification from one host to another as their estimates gently fluctuate. The SDAM addresses these aspects by using a delta threshold. That is, the difference of the specification's resource requirements estimate from the current host and the new potential host must differ by more than the delta threshold for the relocation to occur.

Note that in large environments, a single SDAM may become a bottleneck. To alleviate this, an environment administrator may deploy an SDAM group. That is, a group of SDAMs, hosted on different machines, that all share the same configuration and common group identity. When an ISA connects and there is more than one member of the SDAM group, it picks an SDAM at random. This provides a simple means of balancing the load between each of the SDAMs in the group.

## 4.8 The Situation Determination Agent

The situation determination agent (SDA) is the main reasoning agent in the situation determination middleware. It is the SDA that takes a specification and tries to match the data available within the environment against the expressions in the specification in order to recognise situation occurrences.

The overall operation of the SDA is straightforward. First it receives a specification and optionally one or more customisations from the SDAM. The SDA then analyses the specification and updates its set of subscriptions to properties reported by DCEAs and results computed by CCEAs. It then enters a loop of receiving notifications, matching this new data against the specification, and again updating its subscriptions. If a match occurs against a specification, the SDA then prepares the response and notifies the appropriate ISA. If there are customisations that are associated with the specification, then the SDA first checks the result against the constraints of these customisations and then sends a response when appropriate. If instructed by the SDAM, the SDA may also drop a specification and the associated CEA subscriptions. This will occur as a result of a situation-aware application cancelling a situation request.

The details of the operation of the SDA are more complex however, and these are presented below.

### 4.8.1 SDA and CEA communication

The SDA will communicate with a variety of CEAs in order to establish the properties, relations, patterns and conversions it requires to recognise specifications. It must continually interact with new CEAs that have arrived in the environment, as well as CEAs that are newly required to meet the current demands of the recognition process.

When the SDA requires a particular property or other information, there may be more than one CEA that can provide it. For example, a DCEA that represents a particular person will be centred around providing a number of properties about that person. Such a DCEA may provide a large amount of personal details including the person's name, age, home and work addresses, their location, and so on. However, other CEAs may be centred around providing a particular property, relation or patterns about a number of people or other entities. For example, a meeting room may be fitted with a video-based DCEA that provides high accuracy room-level location and speaker identification properties for each person

in the room. An SDA may require notification of all of these properties, regardless of the style of CEA that produces it.

Within an environment, the population of available CEAs may be large and dynamic. CEAs hosted on mobile devices may enter and leave the environment at unpredictable times. Furthermore, CEAs may fail at any time.

An SDA could discover the available CEAs in the environment using yellow pages-style lookup [117, 118], and manage registering and unregistering interest in particular properties and other information with each CEA individually. However, doing so would not only increase the complexity of the SDA and CEA, but it may also incur a significant runtime overhead.

Given these characteristics, the use of a more loosely coupled style of communication is desirable, and make publish / subscribe more appropriate for communication between SDAs and CEAs [111, 119, 120].

Ideally, a CEA could simply transmit a message, which would then be received by any other interested agent. Similarly, if an agent is interested in receiving a particular kind of message, they could simply advertise this, and receive all messages of that kind. Such runtime flexibility, achieved through loose coupling of message senders and receivers, is the key benefit of publish / subscribe style messaging systems.

Using this, SDAs and CEAs communicate based on the content of a message, rather than the particular destination address of the message's receiver. A CEA can publish notifications by connecting to the publish / subscribe substrate and issuing an advertisement that describes the attributes of the events it publishes. Details of how an agent discovers and connects to the substrate are given below. An SDA can receive notifications by attaching to the substrate and issuing a subscription that describes constraints on the attributes of the events it wishes to receive. When a publisher generates a new event, it will be delivered to each subscriber that has at least one matching subscription.

Advertisements, subscriptions and published events all share a similar structure. Each describes a tuple of the following form:

```
(<Subject>, <Subject Type>, <Predicate>, <Object>, <Object Type>)
```

By issuing a series of subscriptions, an SDA can traverse the graph structure of an entity's information, and can obtain the value of any desired property. Furthermore, the subject and object types provide the SDA with a convenient means of subscribing to a property of all entities of a particular type. For example,

it can subscribe to the identifiers of all entities of type Person, whose location is within a particular room.

While the tuple structure is sufficient to investigate an entity's properties, several metadata attributes are defined for a tuple to support further functions. These attributes include confidence, location and validity period.

The confidence attribute specifies the level of confidence the publisher CEA has that the information provided by the tuple holds.

The location attribute indicates the area in which the reported property is supported. For example, for the video-based DCEA above, it may only be able to report the 'isSpeaking' property for people that are within the same meeting room in which it is installed. As an SDA is only interested in receiving properties about entities that are within the AoI, the location attribute allows the SDA to exclude notifications that originate from outside that area.

The validity period attribute states the length of time the publisher CEA believes the information provided by the tuple will hold. This provides the SDA with a simple and convenient means to identify and remove stale information from its internal model of the environment. CEAs may be mobile or battery powered and so may disappear or fail at any given moment. If this were to happen just after a CEA had published an event for a particular property, the CEA has no way to publish further, more up-to-date events for that property or otherwise indicate that it is no longer valid. An SDA may rely on this property to recognise a situation, but will never receive external notification that it is no longer valid. The SDA could use its own models of a property's validity, but this would considerably increase the complexity of the SDA as it must store such a model for every property type, and may not have a model for newly introduced types. Instead, the publisher CEA provides a validity period estimate for a published event. This way, accurate, type-specific validity models can be built into the DCEAs themselves, and the SDA can reliably purge its internal model of the environment so that it only contains up-to-date information.

A Publish / Subscribe Agent (PSA) was developed to provide the publish / subscribe substrate. A PSA acts as a single node in a publish / subscribe network. Client agents connect to a PSA that they can then send subscription requests to and receive event notifications from. The PSA essentially provides an interface that allows direct access to the context information layer with an environment. Section 3.2 noted that three layers of information existed in a pervasive situation determination environment, where each built on top of the layer below it - sensor

data, context information and situations. By sending a situation request, and application has access to the situation layer. By sending a subscription to the PSA, an agent or application has access to the context information layer. Unlike situations and context information, which have a general, structured form, accessing sensor data may depend upon device- or sensor-specific details. Therefore, access to the sensor data layer is achieved through constructing a specific CEA that is aware of these details.

In larger environments, a network of PSAs may be deployed to avoid a communication bottleneck. Event-producing and event-consuming agents are then connected to one of several leaf PSAs throughout the network.

### 4.8.2 Recognising situations

An SDA recognises a situation by matching the expressions of its specifications against the information it has gathered about the state of the environment. This chapter assumes that matching is performed by an SDA using a Rete network [66], enhanced to support fuzzy-logic reasoning [23, 98]. Recognising a specification is an instance of the many pattern/many object problem [66] as the environment plays host to a number of instances of different types of entity that must be matched against many different expressions within a number of specifications. A Rete network provides an efficient solution to this problem by extracting and reusing the commonality in a set of patterns. The architecture is not restricted by this choice however, and another appropriate matching framework could be used in practice.

Each of the steps an SDA performs in recognising a situation is illustrated in Figure 4.5. In this figure, red parallelograms denote data, the yellow rectangles represent individual steps carried out within the process, the light blue diamonds show decision points within the process and the rounded green rectangle indicates the terminal state of a request.

Figure 4.5 shows that the situation recognition process starts with a specification. The first step to be carried out is to translate this into a form that can be manipulated by the SDA, which in this case is a Rete network.

An SDA is the most computationally intense agent in the middleware, and it is desirable to minimise the load on this agent. However, this conflicts with the intention to recognise as broad a range of alternative specifications as possible, in order to increase the overall confidence of the situation being recognised, as each

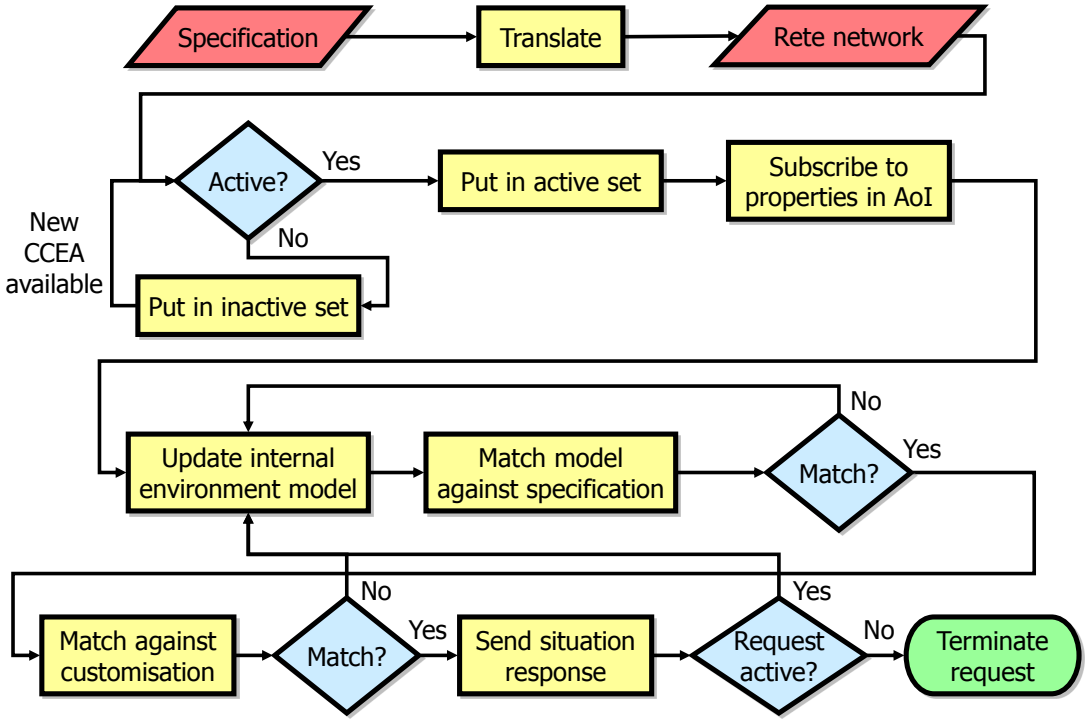



Figure 4.5: This flowchart shows the steps involved in processing a single situation specification, from an SDA receiving the specification to sending the situation response.

specification that is added to the Rete network increases the computational load of the SDA.

In order to balance these contrary goals, the SDA maintains two sets of specifications, the active set and the inactive set. When the SDA receives a specification from the SDAM and it has been translated, it checks with the CCEAM that for each function referenced in the specification, that there is at least one CCEA available that can compute the required function. If there are sufficient CCEAs available, then the specification is added to the active set. However, if not all of the required CCEAs are available, then it is not currently possible to recognise the specification. In this case, the specification is placed in the inactive set. By using these two sets of specifications, the SDA maintains and up-to-date record of which specifications it is currently expected to recognise, and does not waste computational effort by attempting to recognise specifications that can only be partially matched. Furthermore, when the outstanding CCEAs of a specification in the inactive set do become available, the specification can quickly be trans-

ferred to the active set. This is the reasoning behind the "Active?" decision and the "Put in active / inactive set" actions, which are the next steps shown in Figure 4.5.

When a specification is put in the active set, it is loaded into the Rete network. The SDA has already discovered the CCEAs it requires to compute the functions referenced in the specification. The next step the SDA performs is to identify which types of entities are involved in the specification, and discover all instances that are within the Area of Influence (AoI). To do this, the SDA issues a number of subscriptions, each matching one type of entity identified and whose current location is within the AoI. Based on the resulting notifications for this subscription, the SDA can use the entity identifiers to issue further subscriptions to obtain the properties required to match the specification. As each piece of information is delivered through notifications, the SDA updates its internal model of the state of the AoI. These two steps are shown in Figure 4.5 as "Subscribe to properties in AoI" and "Update internal environment model" respectively.

Meanwhile, the Rete network attempts to match the expressions of the role and situation specifications against the SDA's evolving internal model. First, the Rete network will attempt to match various combinations of entity instances against the expressions of a role. When a match is made, the network updates its internal model by entering the type of role matched into the appropriate slot of the model's representation of the primary entity of the role. For instance, when the example presentation specification is being matched, the model for John may look like:

```
id: JohnID
type: Person
...
roles: { (Speaker, 0.89, 13:32:44-13:33:44),
         (Speaker, 0.71, 13:32:46-13:33:56),
         (Audience Member, 0.12, 13:32:40-13:32:48) }
```

Note that there are two entries for the Speaker role. This is because the Rete network may match more than one specification for this role. Note also that each entry contains a time period. These describe the validity period of the match and are based on the SDA's internal clock. Each property that was used to match the role expression will also have a validity period stored in its metadata. The overall validity period of the match as a whole spans from the point in time when

the match occurred to the nearest expiry time of any of the properties used in the match. The confidence of a match for a particular entity is then the highest confidence from all matches for the same type of role, for the length of its validity period. For example, the confidence that the entity with identifier JohnID is playing the Speaker role will be 0.89 until 13:33:44, after which it will drop to 0.71.

When an entity has been identified as playing a role, the Rete network will then use this new information to try to match different combinations of role-playing entities against the expressions of the situation specification. If the situation specification is matched, it is checked against any currently requested customisations. By checking the customisations after the specification has been recognised, the specification need only be processed once, and the result can be shared among several customisations, and as is shown later in Chapter 5, many customisations may exist within the environment. These matching steps are shown next in Figure 4.5 as "Match model against specification" and "Match against customisation".

Recall that a situation-aware application's request may query if an index entity appears in a particular specification, a situation or a situation with a set of customisations applied. Also, that the request may state the reply should be of short, feature set or full form. The values of these parameters affect what is sent in reply to the request when a match occurs. When a short reply is requested, the application wishes to know only whether or not the stated index appears in a particular specification, a situation or a situation with a set of customisations applied, and with what confidence. That is, the application does not care about the contents of the feature set. So when a situation, or a situation with customisations is requested, there is an opportunity for the SDA to fuse the results of the several specification matches to increase the confidence of the short form reply. When a situation is matched by two or more specifications, the SDA picks the match that has the highest confidence, and uses this confidence value for the reply.

When a feature set form reply is requested, a reply is sent for each specification that matches the situation and any customisations. This is because for a feature set form reply, the application does care about the contents of the feature set. As each specification may support different features, a reply is sent for each match with its particular confidence value. This is also the case for full form replies.

A reply is sent as soon as possible. For example, if a request is for a particular

specification, the reply is sent as soon as that particular specification is matched. If a request is for a situation and a set of customisations, the reply is sent as soon as the last of the customisations is matched. In the case of a short form match for a situation or customisations, the SDA must wait until all specification matches have been made, before sending the highest confidence reply.

To summarise, if a particular specification is requested, then the SDA sends a reply immediately after that the specification is matched, for short, feature set and full form requests. If a situation or customisation is requested in a short form request, then the results of all matching specifications are fused by the SDA, and then the fused report is sent in reply. Finally, if a situation or customisation is requested in a feature set or full form request, then the SDA sends a reply after each of the associated specifications are matched, though the receiving SAA may fuse these reports where appropriate.

Note that a notification is not sent after a role is matched. That is because, as discussed at the end of Section 3.6, at that point the entity only has the *potential* to be playing the role. Only once the entity has been recognised to be playing the role within the context of the rest of the situation, is the entity then *actually* playing the role. Only when all the roles of a situation specification are being played and all of conditions in its expression are met is a situation occurring.

The final decision step in Figure 4.5, "Request active?", checks that the request has not yet expired, nor been explicitly terminated by the issuing application. If the request is still active then the recognition process cycles back to updating its model of the environment and continues the recognition, otherwise the request is terminated.

### 4.8.3 Aspects of uncertainty

This section looks at a number of aspects relating to uncertainty that concern the architecture and the situation recognition process. These include trusting the confidence values reported by the system, situation boundaries and the benefits of enabling situation-aware applications themselves to interpret the confidence of a situation.

#### 4.8.3.1 Trusting confidence values

The Rete-based reasoning component presented in the previous section implicitly trusts the confidence values reported for the context information it uses to

recognise situations. That is, it treats individual pieces of context information as facts, where values that are likely to be incorrect are reported at a low confidence, and those likely to be correct at a high confidence. Therefore, by including all reports, the recognition component can trust that selecting those with the highest confidence will lead to the most accurate results being generated.

However, if for some environments or for some particular sensing infrastructure this level of trust cannot be assumed, such that values reported with high confidence may be suspected to be potentially incorrect anyway, it is also possible that uncertainty itself could be reasoned about. That is, the context information used to recognise a situation is taken to represent only evidence of facts, and not the facts themselves.

For example, consider a case where the location system reports with high confidence that a user has suddenly shifted to another room, but at the same time, a speech CEA reports that that user has been talking continuously. If it is known that the location of the microphone used to detect that the user is talking has stayed the same, for example it may be fitted at a fixed position, the system may be able to infer that the user's location may not in fact have changed.

This introduces an independent system of reasoning about context information, requiring its own specifications and knowledge. This is a complementary aspect to recognising a situation, in that it is a separate component that could identify, reason about, and add or remove particular pieces of context from consideration for recognising a situation. It essentially forms an initial, dynamic, context sanitation phase.

Such a system is outwith the scope of the work presented here. It would form a specialised, sophisticated element of a more general context-processing system. However, it would be possible to integrate such a system into both the context information layer, by constructing appropriate CCEAs, and also the situation layer, by employing an appropriate variant of recognition algorithm in the SDA. Applying Dempster-Shafer Theory provides a means of reasoning about the uncertainty of context information, and has been used to good effect by McKeever et al. [121].

#### 4.8.3.2 Situation boundaries

A situation specification creates a boundary within the space of the context information that is used to recognise the situation. When a specification has a short

temporal extent, the SDA can support immediate recognition of its situation. Furthermore, should any of the expressions or constraints within the specification no longer hold, the SDA will immediately cease to recognise the situation.

This quick switching in recognising situations offers two key benefits in both inter-environment and local usage. The architecture is query-based, and in inter-environment operation, it allows a situation request to be made, the index to be located in a remote environment, the recognition to be performed, and the report sent back to the application, all within a sufficiently short temporal extent to give responsive feedback to the user and allow interactive use of the application. For local use, imagine a 'Confidential presentation' situation in which confidential information is displayed in a room to a select set of attendees only. Should someone else unexpectedly enter the room, the information must be removed from the display immediately. Being able to quickly detect that the intended situation is no longer occurring is critical in creating reactive situation-aware applications such as these.

However, there may be other cases where retrospective, rather than interactive, use is desired. For example, for applications that record occurrences of situations for later use, such as the situation-enhanced file search application mentioned in Section 4.2, employing a longer temporal extent may help to clarify which situations occurred over time.

Continuing with the confidential presentation example, imagine the presentation is taking place, it is interrupted by an unauthorised person entering the room, and upon them leaving the confidential presentation is resumed. This creates an uncertainty as to whether two instances of confidential presentation occurred, separated by another or an unknown situation, or whether a single, interrupted instance occurred. Once the presentation has finished, it may be possible for the system to look back over the full period of time and identify that it was in fact the latter case.

While such retrospective reasoning can help reduce uncertainty in recognising situations, it comes at the cost of impeding interactive and inter-environment use, which are critical for pervasive situation-aware applications. It could be possible that the system would be able to support both styles of reasoning where each is appropriate. Other possibilities afforded by a longer temporal extent are discussed in Section 6.2.1.

#### 4.8.3.3 Interpreting situation confidence

Interpreting the confidence value associated with a situation report can be an intricate task. It can incorporate several factors, including the current intent and context of use, whether it is part or all of the situation that is important, what actions are intended to be taken as well as the risk involved if these actions are mis-applied. The combination of factors may be specific to a particular type or even unique to a particular instance of an application.

To assist in this, the architecture allows a situation-aware application full control over the interpretation of the confidence of a situation. That is, the middleware does not attempt to interpret the confidence on the application's behalf.

This allows the application designer to fully capture the intricacies of correctly interpreting the confidence and to obtain the appropriate precision for the particular application at hand. Their scope would not be limited due to unanticipated uses or other restrictions that a standard implementation in the middleware may unintentionally impose.

## 4.9 The Index Locator Agent

In the opening scenario, Angela used a situation-aware availability checker application to see if John was currently busy or not. A situation is recognised in the local environment of the situation's index. When Angela ran this application, she was not aware of the location of John, who was the index of the situation request. He may have been in the local environment, in the same city or perhaps in a different country. The purpose of the index locator agent (ILA) is to discover the ISA of the index of an application's situation request, wherever in the world it may be located.

Locating a mobile entity within a network is a problem common to many systems. A common solution, used by the Mobile IP protocol [122], is to locate a mobile entity via a home server.

A home server is a host machine that can be reached from a fixed address. As the mobile entity moves throughout different networks, and as such may change address, it notifies its home server of the address it currently has, known as its 'care-of' address. Any messages that are to be sent to the mobile entity are sent to the fixed address of the home server, which forwards the message to the mobile

entity at its current care-of address.

While the home server approach has been used successfully in other systems, it is less well suited for use for pervasive situation determination. In particular, it is undesirable to have to set-up and maintain a home server for every person, device and location that may feature as an index entity of a situation subscription, as the total number of such home servers would be large. Furthermore, the home server introduces an additional component that provides a single point of failure for determining situations that involve the particular index entity.

Instead, a distributed home server approach is proposed in which each entity can manage their own home server individually and automatically, that builds on top of the existing situation determination infrastructure, and does not rely on a single host machine being operational.

Several distributed hash table (DHT) algorithms have appeared in the literature in recent years. These include Pastry [123], Chord [124], CAN [125] and Tapestry [126]. Distributed hash tables provide an appropriate means to implement a distributed home server, as they are resilient to host machine failures, can automatically adapt to host machines frequently joining and leaving, can scale to very large network and hash table sizes, and can locate entries stored in them very quickly.

A DHT provides a key-based lookup and storage facility (just as a regular hash table does), distributed across many nodes in a network of such nodes. The architecture implements a distributed home server on top of a DHT. Each index entity has a unique, fixed identifier (specifically a UUID [127]). It uses this identifier as the key to write its current care-of address into the DHT. Conversely, to locate the entity, an agent can lookup the current care-of address in the DHT using the entity's fixed identifier.

A situation-aware application makes a situation request to an SAA in exactly the same way, regardless of whether the situation it requests is occurring in the local environment, or in an as yet unknown external environment. The SAA uses the ILA to manage the detail of locating the index entity and provide transparent operation to situation-aware applications.

When an SAA receives a situation request from an application, it passes the identifier of the index to an ILA. The ILA then attempts to discover the index's ISA in the local environment, based on the index's identifier. If the index's ISA is found, then the SAA registers with it, and will receive situation notifications as normal.

If the ISA is not found, the ILA looks up the current care-of address of the index entity in the DHT. If a care-of address is found, the ILA uses this to forward the situation request to the ISA of the index entity. The remote index entity's ISA will issue the request to the SDAM in the remote environment. Similarly, any situation notification received by the remote index entity's ISA is sent back to the application's SAA in the local environment, as if the notification had been generated in the local environment.

If no match is found in the DHT, it implies that the index entity cannot be found on the network, and so the situation request is dropped.

The nodes that make up a DHT form a connected network. In order for agents to read from or write to the DHT, they must connect to one of its nodes. A deployment in an administered environment would typically host a DHT node, which can be used by any ILA within it. As is shown in the next section, the situation determination middleware also supports an ad hoc mode of operation that does not rely on fixed infrastructure being available. In this case, an ILA hosted on a mobile device would connect to a pre-configured DHT node, such as one hosted in the owner's home environment, or perhaps even one hosted by the owner's Internet Service Provider.

By using a transparent combination of large-scale DHT-based lookup and localised agent discovery, the situation determination middleware seamlessly supports inter-environment operation.

## 4.10 Environment and Ad hoc modes

One of the aims of the design of the situation determination middleware is to support recognition of situations wherever the situation's index may be. That could be in a private, administered environment such as the workplace, where there may be dedicated infrastructure available for use by the middleware. It could also be in a more public, non-administered environment such as an open-air café or a train station, where it is only the collection of devices that a single person or a group of people carry with them that is available to the middleware.

In order to support continuous recognition of situations, the middleware provides two modes of operation. One is the environment mode, designed for use in private, administered environments with dedicated infrastructure available. The other is the ad hoc mode, designed to be used in public, non-administered environments with only an ad hoc collection of (most likely) mobile devices available.

Previous sections have presented the environment mode, but this section focuses on the ad hoc mode and transitions between the two. When operating in the environment mode, the middleware attempts to incorporate as many devices and as much information as it can. However, this approach could prove disastrous when operating in the ad hoc mode. In the ad hoc mode, the devices being used are most likely to be portable, resource-constrained, battery-powered devices such as a mobile phone. If a phone were to attempt to discover and connect with all other devices in range, there may be so many that it would be choked just trying to connect to the sea of devices within network range. Consider for example, the case where a user is in a busy train station, and there may be thousands of other mobile phones. To avoid this problem, when operating in ad hoc mode, devices will only try to connect to other known devices that have been explicitly identified to be incorporated. A mobile device will operate in ad hoc mode by default.

A user may configure a mobile device to either manually or automatically join a new environment when one is detected. In both cases, it is the responsibility of the Environment Manager Agent (EMA) to co-ordinate the transition.

There are several details that must be configured when a mobile device enters a new environment. These include, where appropriate, setting which SDAM the device's ISA should use, and likewise which CCEAM the device's SDA should use, setting which DHT node the device's ILA is connected to, and linking together nodes of the publish / subscribe network.

Each environment advertises itself on the network using an ad hoc discovery protocol. As with an ISA, an EMA is assigned a UUID. Using the ad hoc discovery protocol, an environment advertises the UUID of its EMA agent, as well as the type of environment (administered or ad hoc) that the EMA represents.

When a mobile device enters a new network, its EMA agent can discover all other EMAs in the network using ad hoc discovery. Through interacting with the other EMA, the mobile device can join an administered or ad hoc environment. If the user has the mobile device configured to manually join a network, then the device may alert the user that new environments are available, and the user can select which environment they wish to join. The user can also configure the device to automatically join new environments.

When a mobile device connects to an administered environment, the process is relatively straightforward. The mobile device's EMA discovers other environment EMAs of administered type. The device's EMA uses the address of the

environment's EMA from the advertisement to send it a request for the addresses of the environment's SDAM, CCEAM, DHT node and publish / subscribe node.

The device's EMA then sets its ISA to use the environment's SDAM, and the ISA forwards its current set of situation requests to the environment's SDAM. Any active requests that use the device's local or a previous environment's SDA are cancelled. Similarly, the EMA instructs each of the DCEAs hosted on the mobile device to connect to the environment's publish / subscribe node, and cancel the previous connection. The EMA also sets the device's SDA to use the environment's CCEAM, and the device's ILA to use the environment's DHT node, if one is available. When the EMA has completed these steps, the mobile device will now operate within the new environment, and any active situation requests will have been transparently transferred over into the new environment.

When a mobile device automatically joins an ad hoc environment, the process is a little more complicated. This is because of the extra restrictions in place for ad hoc operation. As noted earlier in the chapter, the set of environments a device will attempt to join must be restricted, in order to prevent the device from being flooded with join requests when many ad hoc environments are available, such as in busy public places like a train station. Figure 4.6 gives an example of how a number of mobile devices may be connected when operating in ad hoc mode.

Mobile devices are configured with a user-maintained white-list of ad hoc environments, which are the only ad hoc environments the device will try to connect to. The white-list is made up of a ranked set of EMA UUIDs. In the case where more than one white-listed environment is available, the EMA will select the one with the highest rank. The user can add and remove new and known environments to and from the white-list as required.

When joining an ad hoc environment, it may not always be appropriate to switch to the SDAM and SDAs of the new environment. For example, if the ad hoc environment consists of a single mobile phone, and then three other mobile phones join the ad hoc environment, it could turn out that it is the unlucky original phone's SDA that would now have to try to recognise all the situations of all four phones, choking its resources. However, in another case, the ad hoc environment may first include a single laptop that can happily support the recognition of its own and the other three phones' situation requests.

In order to control which devices are used in the foraging of recognition load within ad hoc environments, mobile devices also employ a user-maintained

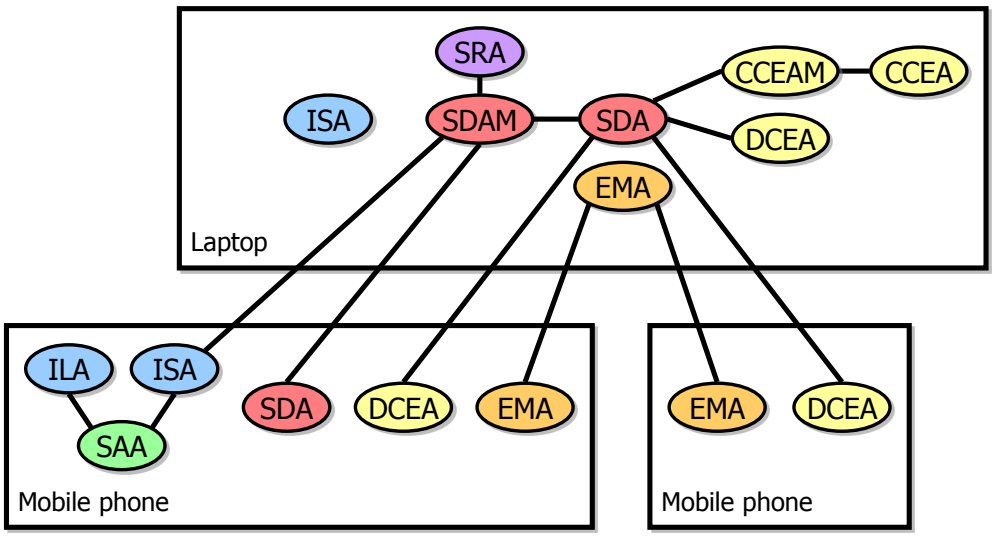



Figure 4.6: An example network of mobile devices operating in ad hoc mode. Here, both mobile phones have the laptop listed on their white-list and offload-list, and so both use the laptop to perform situation recognition. Lines indicate communication between two agents. For clarity, not all agents that would be hosted on a device are shown, only those that help illustrate the devices' interaction.

offload-list. The offload-list is also made up of a ranked set of EMA UUIDs. The environments featured in the offload-list will be a subset of the environments included in the white-list.

A key advantage of this user-configured scheme is that it puts the user in control. While a middleware-managed scheme could relieve the user of having to maintain an off-load list, the current scheme appeals to the user's self-interest. In ad hoc mode, devices will be competing to preserve their own battery. The user-configured scheme allows the user, rather than the middleware, to dictate which devices are favoured, and so control resource foraging.

When the mobile device discovers an ad hoc environment, it first checks the UUID of the environment against its white-list and offload-list. If the UUID features only on the device's white list, then the EMA will simply connect the device to the new environment's publish / subscribe network. If the UUID features on the device's off-load list, then the EMA will perform the same changes for joining an administered environment, which are to switch to using the off-load host's SDAM, CCEAM, DHT and publish / subscribe network. If more than one of the

devices present features in the offload-list, the EMA will select the device that has the highest rank. As the set of devices a user owns and encounters changes over time, he/she can alter the offload-list to best suit.

By supporting seamless operation for the transition between environment mode and ad hoc mode operation, the situation determination middleware allows situation requests for a particular index to transparently persist over changes of location, network, environment and operational mode.

## 4.11 Application interfaces

This section describes the application interfaces offered by the pervasive situation determination middleware to situation-aware applications.

The interfaces are agent-based interfaces in that they build upon the message passing interface of the underlying agent substrate. Two agents provide an interface to situation-aware applications. The first is the situation-aware application agent (SAA), which provides an interface to make and receive situation requests and responses. The second is the Publish/Subscribe Agent (PSA), which provides an interface that gives situation-aware applications direct access to the context information layer. Each of these interfaces is presented in turn in the next two sections.

### 4.11.1 The SAA interface

The situation-aware application agent (SAA) presents a simple situation query interface to situation-aware applications. The interface supports three basic operations. The first operation allows situation-aware applications to discover which situations, customisations and specifications are supported in the local environment. This is particularly useful for situation-aware applications that require to discover situations at runtime, rather than have the situation or specification identifiers the application requires hardwired at design time. For example, a situation-aware navigation aid application may present to a user a list of discovered situations that are supported in the environment, and having recognised the type of situation they are looking for, they can then search for occurrences of that particular situation. The query is named 'getSituations' and can be parameterised to return the set of supported situations, customisations, specifications or any combination of these.

In the description of the interface below, the term target is used to refer to situations, customisations and specifications collectively.

The second operation allows situation-aware applications to request notifications of occurring situations. This query is named 'requestSituations' and takes the following parameters:

- index - this parameter indicates which situation index is to be used. It reflects the principal entity whose situation(s) an SAA wishes to know about. An index may be a device, a person or a location. When the index is a person or a device, a SAA is asking either "is person/device X involved in situation Y?" or "which situations is person/device X involved in?". When the index is a location, a SAA may ask "is situation Y occurring at location X?" or "which situations are occurring at location X?".

- target - this parameter indicates what is to be recognised. It may specify the identifier of a situation, a customisation or a particular specification. When used with an identifier, the SAA is asking "is *index* involved in target *identifier*?". This parameter may also have the special wildcard value '*' which indicates that the SAA wants to receive notification of all of the situations or customisations that the index is currently involved in. When used with the wildcard value, the SAA is asking "which situations is *index* involved in?". This parameter may also specify combinations of targets. For example, the request could specify a customisation and a number of specifications, meaning "is *index* involved in the stated customisation, as recognised by any of the given specifications?"

- confidence - this parameter specifies a constraint on the confidence of the situation occurrence. For example, a situation-aware application may set this to be greater than 0.6, meaning that it only wishes to receive notification about situations that have been detected with a confidence greater than 0.6.

- mode - this parameter specifies the form of the response the situation-aware application requires. The parameter can be set to the 'short', 'features' or 'full' form. Each of these forms is described below.

- expiry - this parameter specifies the amount of time the request should be maintained. It indicates to the middleware that after this amount of time

has passed, the situation-aware application no longer requires the results and the request may be safely terminated.

In some cases, an application will not be interested in the features of a situation, requiring only notification of whether a situation is occurring or not for a particular index. For example, in the scenario, Angela wished initially to be notified only of which situations John was currently involved in, and not the details of those situations. The response to this style of query reports only the situation index, the target identifier and the confidence of the occurrence. To receive this style of notification, a situation-aware application can specify the 'short' form for the mode parameter.

Some applications will require the full feature set of a situation to be returned. For example, the situation-enhanced file search tool John used in the scenario requires this in order to mark-up each file access with as many search criteria as possible. That is, not only the identifier of the target, but all of the details of the features as well. The response to this style of query contains the target identifier, the confidence and the results of each of the features of the situation. For example, the results of the 'Audience Members' feature of the Presentation situation would contain all of the identifiers representing each of the people who were detected to be in the audience. To receive this style of notification, a situation-aware application can specify the 'features' form for the mode parameter.

Environment administrators and specification developers may wish to receive even more information about the matched specifications, to assist them in debugging a deployment or authoring specifications. The mode parameter may also be set to the 'full' form. The response in this case includes the identifier of the specification that the SDA matched, the identifiers of each entity that was involved in the match, as well as the values of all of the properties that are referenced in the specification at the point when the match occurred.

Consider an example query drawn from the scenario presented earlier, where the situation-aware availability checker Angela used first requested to be notified of which situations John was involved in. Here, the query would have the following form:

```
requestSituations(JohnID, *, > 0.6, short, 60 mins)
```

indicating that the situation index is to be John, that any situations that John is involved in are to be included, the confidence must be greater than 0.6, the short

form of notification is to be used, and that the request should be maintained for 60 minutes.

Next, Angela sets the availability checker to notify her when John is no longer taking part in the presentation. Note however, that before issuing this request, the application must first inform the SDA that it is no longer interested in the results of the first request. Although the request will be automatically terminated once its expiry time has passed, an application may also explicitly terminate a request by calling the third operation supported by the SAA, 'unregister'. When the 'requestSituations' operation is called, an identifier for that request is returned to the application. To end a request, the application can pass the request identifier to the 'unregister' operation.

The SAA uses an index locator agent (ILA) to detect if the index of a request is located in the local or a remote environment. While the details of the ILA are covered later in Section 4.9, note here that special attention must be given to the distribution of customisations when the request is for an external environment. As customisations may be bespoke to an individual situation-aware application, the SDA that will eventually recognise the customisation must have access to it. When the index is located in the same environment as the application, the customisation will be available from the application's local situation repository agent (SRA). In the case where the index is located in a remote environment, the customisation is embedded in the request, in order to make it available to the local environment of the index.

### 4.11.2 The PSA interface

The Publish / Subscribe Agent (PSA) interface allows situation-aware applications to directly query context information that is available in the environment. For example, this interface may be used to query further information about an entity that appeared in a feature of a situation report, but the desired additional information is not relevant to the situation or did not appear in a related feature in the situation report.

The interface is based on the operation provided by the publish / subscribe substrate to which the PSA provides access. Applications can register interest in receiving particular types of context information by sending a subscribe message to the PSA. Conversely, an application can indicate to the PSA that they are no longer interested in particular types of context information by sending an

unsubscribe message. The PSA publishes context information to all current, interested subscribers by sending notification messages that contain the desired context information.

This interface is not only available to situation-aware applications, but also any other agent in the system. For example, it may be common for a Compute Context Entity Agent (CCEA) to subscribe to one or more Data Context Entity Agents (DCEAs) to aggregate or further transform multiple context information sources.

Subscribe and unsubscribe messages are specified using a template based on the typed tuple form, shown earlier in Section 4.8.1, to describe the type of context information they wish to receive. The template may contain matches on exact values, functions indicating a range of values that could be matched, as well as wild card or "don't care" matches where any value will be matched.

As an example, assume an agent or application wanted to determine all of the people that were currently located within a particular location, room "L10.01". This could be achieved by issuing a subscription with the following template:

```
(*, Person, location, within(L10.01), Location)
```

This template states that the subscription would match any entity with type 'Person' that has a property 'location' whose value satisfies the predicate 'within(L10.01)' and is of the type 'Location'.

Notifications deliver context information also in the typed tuple form. However, rather than being a template which specifies values and ranges to be matched, the notification contains the actual data that was matched against the subscription template. For example, a notification matching the subscription above with a person represented by the identifier 'JohnID' could take the following form:

```
(JohnID, Person, location, L10.01-speaker-area, Location)
```

An agent or application may unsubscribe from a previously issued subscription by sending an unsubscribe message to the PSA with the same template given in the subscription.

## 4.12 Summary

This chapter has presented an architecture for pervasive situation determination. In particular, it was shown how the architecture provides support for the following requirements:

**Adaptable recognition** - It was shown how the architecture can automatically exploit any new data sources or sensing technologies introduced into the environment on mobile devices in the form of data context entity agents (DCEAs), as well as any new computational, pattern recognition or conversion capabilities in the form of compute context entity agents (CCEAs), and how the flow of information from these was co-ordinated and fused by the compute context entity manager (CCEAM). Also shown was how specification repository agents (SRAs) act as dynamic sources of new situations, specifications and customisations within an environment. The flexibility of the situation recognition process used by a situation determination agent (SDA) was demonstrated, including how each of these sources can be dynamically incorporated to provide recognition of the broadest range of situations at the highest granularity possible. The autonomous co-ordination of these agents actions provide an extremely flexible, extensible and adaptable situation recognition process that can continually adapt to an environment as it evolves.

**Resource management** - The architecture employs several techniques to evenly distribute the load between agents of the system and to strive to preserve the battery life of mobile devices. These included resource requirements based assignment of situation specifications to SDAs by the situation determination agent manager (SDAM), as well as a resource foraging approach to allow the computationally expensive operation of recognising situation specifications to be off-loaded from resource-constrained to resource-rich hosts. Also included was the specification grounding technique that can be used to minimise communication costs for mobile SDA hosts.

**Inter-environment operation** - This requirement was supported by the architecture in two ways. The first was the ability of the index server agent (ISA) of an individual person, device or location to receive situation requests from and send situation reports to a situation-aware application (SAA) anywhere within the network. The second was by supporting both environment and ad hoc modes of operation and having an environment manager agent (EMA) autonomously manage transitions between the two, allowing the architecture to support continuous recognition of a person or device's situations over changes in location, network or environment, as well as changes in the architecture's operational mode.

**Situation discovery** - It was shown how within the architecture, an index server agent (ISA) and an index locator agent (ILA) co-operate to provide large-scale discovery of the index of a situation and how this enables situation-aware

applications to base their behaviour not only upon situations that are occurring in the local environment, but also upon those in external, potentially remote environments elsewhere in the world.

The architecture presented in this chapter provides a comprehensive approach by both supporting the unique features of the modelling approach presented in Chapter 3, and also fulfilling the distinct architectural requirements necessary to achieve pervasive situation determination.

# Chapter 5

# Evaluation

## 5.1 Introduction

This chapter presents an evaluation of the pervasive situation determination middleware prototype described in the previous two chapters.

The work presented in this thesis has focussed on providing support for the distinct modelling and architectural requirements necessary to realise pervasive situation determination. These include the requirements for end-user customisation, rich situation models, alternative situation descriptions and multiple viewpoints in the model, as well as the particular adaptable recognition, resource management, inter-environment operation and situation discovery requirements of the architecture.

This chapter presents an evaluation of how well the situation determination middleware supports these themes. In particular, the evaluation addresses the following criteria:

**Sufficient expressivity** - Establish that the model and architecture are sufficiently expressive to conveniently describe several situations, specifications and customisations spanning a range of domains and environments, and support a variety of styles of pervasive situation-aware applications, respectively.

**Straightforward instantiation** - Demonstrate that situation models, the architecture and pervasive situation-aware applications afford straightforward instantiation.

**Practical performance** - Determine that the middleware provides a sufficient level of performance for the number and size of situations that would be

encountered in real-world settings.

These criteria were chosen as together they broadly cover a number of important, high-level qualities of the middleware in use.

First, the effort required to initially instantiate the model and architecture in a prototype implementation is discussed in Section 5.2. It demonstrates the rich tool set available to assist in creating situation descriptions and details existing frameworks that simplify the development of the middleware.

Section 5.3 goes on to explore the ease of instantiation of situation-aware applications and the level of expressivity offered to them by the middeware. It shows how simple it is to incorporate the situation determination middleware into an application, to set up the sensing infrastructure required by an application, and to create situations, specifications and their customisations.

In exploring expressivity, the aim is to determine if the prototype middleware has the capability to support the description and recognition of a range of situations and customisations across a number of domains and different environments. Four different domains are explored, including the use of devices and applications, a domestic setting, a public setting and a University setting. While the University setting is slightly specialised, it suggests the kind of situations that could be recognised in an office environment too. These domains reflect settings that would commonly be encountered by a large number of users, and the situations developed for each cover a broad range of different styles of situation that would commonly be encountered within that domain, demonstrating that the middleware would be widely applicable.

Furthermore, this section presents a case study of the development of not only this collection of situations, specifications, customisations, and their required context sensing and processing agents, but also a suite of applications, each drawn from a different class of situation-aware application and distinct in purpose. This suite includes an availability checker application, which is an example of an application that performs ad hoc situation queries about a person, device or location in real time, whether the target may be in the local environment or in an unknown remote environment; a mode manager application, which is an example of an application that continuously monitors and autonomously reacts to particular local situations in real time, constantly adapting to changes in sensing infrastructure and environment as the user travels from place to place throughout their day; and also, a situation-enhanced file search application, which is an example of an archival application that detects and records all of the situations the user

is involved in and allows later searching and analysis of those situations for information and artefacts associated with them. These particular applications were chosen as together, the suite of applications exercise the full functionality of the middleware, as well as being fun and appealing applications in themselves.

This part aims not only to show that the situation determination middleware satisfies the main goals set out in this thesis, but also to demonstrate its utility in a realistic deployment. Through developing a range of styles of application, this part attempts to illustrate that the middleware supports a sufficient level of coverage of the situation-aware application domain, in addition to making it easy to create powerful situation-aware applications.

The next section looks at practical performance. It first describes the experimental set up in a University environment that was used to conduct the performance assessments in Section 5.4.1. The University environment was chosen as it played host to the largest and most complex situations, and so represents the system under the greatest load.

The purpose of Section 5.4.2 is to show that the pervasive situation determination middleware is suitable for use by interactive situation-aware applications as it scales to realistically-sized deployments, for both infrastructure-based and ad hoc environments. Therefore, two measurement criteria have been chosen that are important for interactive use: round trip time and join time. The load incurred in recognising situations can scale in two ways. The first is in the size of the situation itself, that is, the number of people, devices and other artefacts that are involved in a single instance of a situation. The second is the number of types and instances of situations that are recognised at the same time. This section reports a series of measurements that capture both of these cases. Moreover, these measurements are repeated for both infrastructure-based and ad hoc modes of operation. This section also examines the efficiency of the additional technique of sharing the recognition load between multiple hosts in further addressing scale in infrastructure-based environments. This section closes by providing relative resource consumption rates, to give an impression of how the different parts of the system contribute to the overall system load.

Section 5.4.3 presents an algorithmic analysis of the Rete-based recognition core to indicate how the overall complexity grows in relation to the different elements that comprise a situation specification. An analysis of grounded specifications is also given, providing an example quantification of the savings afforded by this technique. This section closes by summarising the performance charac-

teristics of the agent substrate and the distributed hash table implementation employed by the middleware.

The focus of the evaluation is on the novel aspects of the pervasive situation determination middleware presented in this thesis. In this respect, assessing the accuracy of the recognition algorithm is outwith the scope of this evaluation. As noted in Section 4.2, the middleware does not represent a single, stand-alone system that is to be used exclusively within an environment. Is it an adaptable and extensible middleware that can be overlaid and integrated with existing systems within the environment at the sensing infrastructure, context and/or situation layers, allowing specialised components to be used where necessary. Recognition accuracy is a property of the recognition algorithm itself. The Rete algorithm has been used in this approach to provide a default implementation. As the Rete algorithm is not part of the contribution of this work, assessing its accuracy lies outwith this evaluation. (It is possible to note anecdotally however, that as location gave the greatest source of uncertainty in most of the specifications developed in this chapter, that recognition accuracies were similar to that for the location system itself, which are reported in Section A.2.4.)

However, performance measures of the default Rete-based implementation are included here to show that the novel aspects of the approach can be fully supported, and that in doing so it is possible to achieve sufficient performance.

Similarly, as the middleware employs the agent substrate and distributed hash table implementations in their standard, intended ways, including the authors' own assessment of these frameworks is appropriate.

## 5.2 Instantiating the model and architecture

To begin, the instantiation of the model and architecture is presented. This is to assess, under the complexity criterion, if it is possible to instantiate and use the model and architecture easily. The specific technologies that were used to represent situation descriptions and realise the model, as well as the specific technologies that were used to instantiate the architecture are discussed.

Each of the three flavours of situation description–situations, customisations and specifications–is represented by an OWL document. The Web Ontology Language (OWL) is a language for creating ontologies [128] and allows for the rich description of semantic models. The entities within a situation, their properties and relations, the environment, and the situations, specifications and customisa-

tions themselves can all be semantically described.

There are several factors that make OWL an appropriate choice of language for situation descriptions. First, it allows rich, semantic models to be constructed to represent the environment, entities and situations. Elements of the model can be reasoned about abstractly, allowing greater flexibility and generality in writing situation specifications. The collection of situation descriptions within an environment may come from several different sources, and so make use of their own ontology or extensions. Tool support exists to automatically infer equality between the concepts and relations of two different OWL ontologies [129, 130, 131]. Furthermore, tool support exists to automatically check the validity and consistency of situation descriptions [132, 129]. Additionally, there are tools available for graphically creating and manipulating situation descriptions [133]. As situation descriptions are formalised in an OWL ontology, it is also possible to define translations between OWL and other ontology languages, increasing the situation descriptions' interoperability and availability. OWL is an open standard, and is freely available [128]. Also, the language is XML-based, making it accessible to, and interoperable between, many platforms and programming languages.

The situation ontology defines the structures of a situation, a customisation, as well as a specification. In addition, more general concepts and relations are also defined which model the entities that may appear in the environment. These include people, devices, software artefacts, as well as some domain-specific concepts such as classes, lectures, labs, etc.

Defining a suitable and complete ontology for situation-aware computing is not the focus of this work. Works that do strive to provide this include COBRA-ONT [134], CONON [26] and Ontonym [27].

Unfortunately, there were several factors that prevented one of these ontologies being used in the implementation of the situation determination middleware. One of which was availability, only the CORBA-ONT ontology was publicly available at the time of development. Another factor was coverage, despite aiming to be general, COBRA-ONT did not include key concepts and relations required for the applications developed. Conversely, the ontology included several concepts that were not required by the middleware or applications, and so would have introduced extra overhead to support. A further factor was novelty, the location model introduces the notion of location types, which were not supported by existing ontologies.

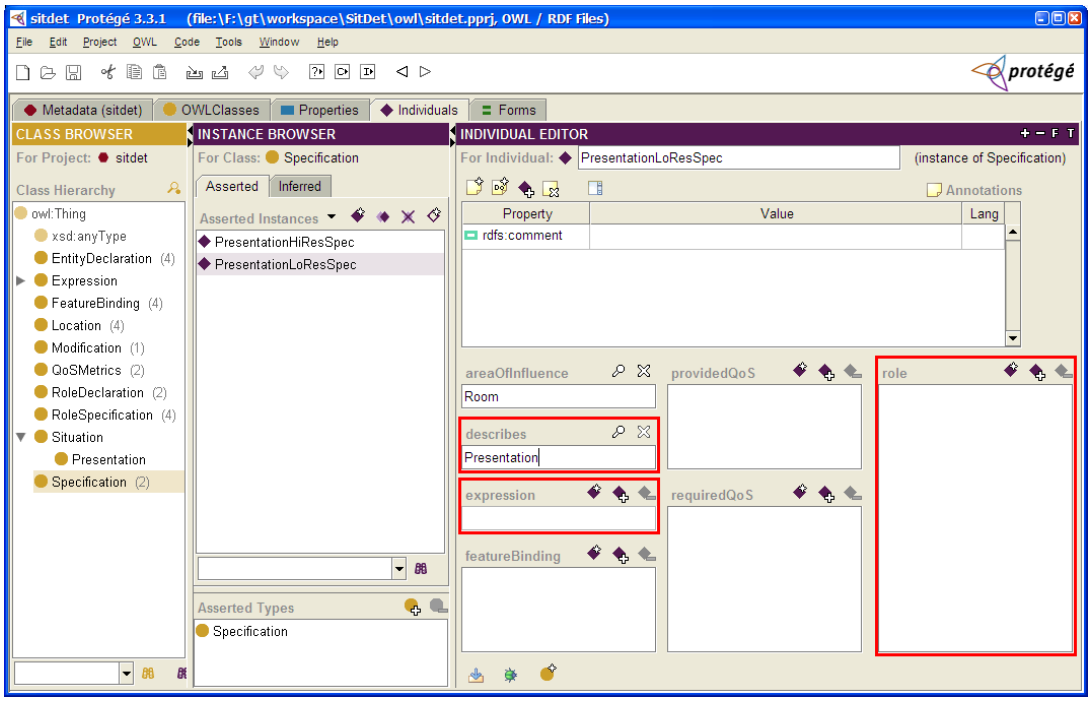



Figure 5.1: Using the Protégé ontology editor to graphically create new specifications.

It was therefore necessary to develop a custom ontology for use by the situation determination middleware and the situation-aware applications that were developed. In addition, using a concise, tailored ontology helped in quickly producing a prototype system. The full listing of the situation ontology can be found in the appendices in Section B.2.

Writing a situation specification can involve detailed work. While it is possible for a developer to write specifications in OWL XML directly, this is a tedious and error-prone process. Fortunately, tool support exists for editing OWL documents graphically, such as the Protégé Ontology Editor [133], which reduces the effort required to produce, and also checks the validity of, the documents created. Figure 5.1 shows a snapshot of an alternative, coarse-grained Presentation specification being created using Protégé.

Similarly, it is desirable to be able to create customisations quickly and simply, perhaps even for situations in which the user is currently involved in. It is possible to create a customisation with a graphical ontology editor such as Protégé [133], but this is not the most convenient way. For example, it may only be run from a laptop or PC, and not a PDA or mobile phone.

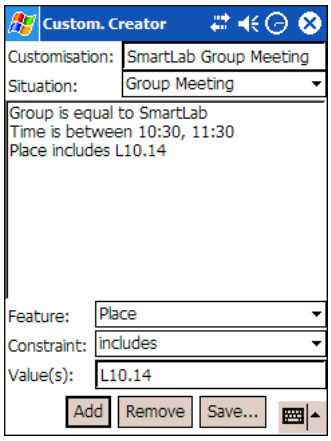



Figure 5.2: Using the customisation creator tool to create a new customisation on a Pocket PC.

To facilitate such quick and simple creation of customisations, a dedicated tool was developed. A snapshot of the tool is shown in Figure 5.2. By analysing the structure of a situation description, it dynamically provides drop-down boxes for each of the features of a situation and possible constraints that may be placed upon them, assisting the user at each stage when creating a customisation's constraints. Once created, the customisations will be stored by the user's situation specification repository agent and accessed and applied when appropriate when situation recognition is performed.

Instantiation of the architecture was made swift by the availability of several appropriate frameworks that could be used to construct the non-exclusive parts of the prototype implementation. For example, the agent implementation for the pervasive situation determination middleware prototype was based upon the Java Agent DEvelopment Framework (JADE) [118]. This framework was chosen as it was freely available, open-source, mature and in active development, and well documented. It provided many of the features required by the pervasive situation determination architecture, including suitable abstractions for agents and agent behaviours, agent discovery, agent messaging including inter-environment agent messaging, support for wired and wireless networks, was able to handle intermittent connectivity through automated store and forward techniques, as well as support for multiple device platforms, from server-class machines to mobile phones.

Similarly, the Situation Determination Agent (SDA) implementation is based on the Jess Rule Engine realisation of the Rete network algorithm [135]. The Pub-

lish / Subscribe Agent (PSA) implementation employs Sienna-like attribute-based publish / subscribe communication which provides a sufficient level of expressivity while offering good performance [119, 136]. The Environment Manager Agent (EMA) implementation uses the Service Location Protocol (SLP) for ad hoc agent discovery [137]. SLP was chosen as it is a simple protocol that can be used in either small or large networks, and is supported by mature, freely available implementations in several programming languages [138]. The particular SLP implementation that was used was jSLP, a lightweight Java SLP implementation that is in active development [139]. While the implementation of the situation discovery parts of the pervasive situation determination middleware made use of FreePastry for DHT support [123, 140].

This allowed the effort involved in the instantiation to focus on the original parts of the architecture. In addition to the development of the inter-agent protocols, these also included the mechanism to translate situation specifications into a form that can be processed by the Rete network (the full details of this translation process are included in the appendices in Section C.1). Furthermore, there were the adaptable recognition techniques, the resource management capabilities, as well as inter-environment operation and situation discovery facilities.

## 5.3 Developing situation-aware applications

This section presents a number of applications that were developed using the pervasive situation determination middleware. First, the agents that were created to supply information about different entities in the environment to the middleware are presented. Following this, a survey of the different situations, specifications and customisations that were created is given, before detailing each application individually.

The expressivity of the approach in this regard is assessed by presenting the wide range of situations, customisations and specifications that were created across a number of domains and for a variety of sensing infrastructure and environments, as well as their use in a suite of situation-aware applications, where each exploits the middleware in a different way and for its own distinct purpose.

The complexity of the approach is assessed by illustrating the effort involved across the full extent of the development lifecycle, following the development from scratch of the necessary ontology and CEAs, through building the situation descriptions, to constructing the situation-aware application code.

### 5.3.1 Developing CEAs

A total of thirteen context entity agents (CEA) were developed. These included eight Data-CEAs (DCEA) that produced data about the properties of entities in the environment, and five Compute-CEAs (CCEA) that provided functions for that data.

Two of the eight DCEAs provided fixed properties about a person and a device respectively. The person DCEA provided properties such as the person's name, date of birth, the groups they belong to, which office they work in, etc. While the device DCEA provided properties about a particular device such as its identity, the type of device it is, who owns the device, its operating specifications, etc. Both of these DCEAs can be easily configured to report new, additional properties.

The other six DCEAs were built to detect more dynamic properties. These included the location DCEA which embedded an instance of the location detector described in Appendix A, a calendar DCEA that reported entries from a person's calendar or planner, an application monitor DCEA which reported properties about the applications that were running on a particular device, a mouse monitor DCEA and a keyboard monitor DCEA that detected keyboard and mouse activity on a host device, and a DCEA that reported whether a particular person was speaking or not.

The calendar DCEA was based on the Google Calendar Data API using the provided what, when and where tags, to allow real calendars to be used without special extension. For details on the Google Calendar Data API, see [141].

The application monitor DCEA reported the window ID of an application, the host machine ID (these combined give a unique ID for the application instance), the text of the application's title bar, the name of the executable of the application, whether the application was the active application on the host machine (the application that currently has input focus), and where possible, the active file (extracted from the title bar text). If an application has multiple windows, an event is emitted for each. The design here assumes that an application will have at least one window, which is reasonable given that it is this element of a user's interaction with the application that the DCEA is trying to detect. Further details for this DCEA are provided in the appendices in Section C.2.

The keyboard and mouse DCEAs detected whether a key had been pressed, and whether the mouse had been moved or a button had been pressed, respectively. Again, further details of these DCEAs can be found in Section C.2.

The DCEA to detect whether a person was speaking or not was implemented as a small application that could simulate Boolean values, reported at a variable confidence, via graphical control. The DCEA was implemented this way to allow a convenient means to test situation specifications that incorporated this property as it did not require the actual speakers to be present. Given the availability of a suitable speaker recognition component, a version that reacted to a speaker's voice could be implemented by embedding an instance of the component in the DCEA.

The five Compute-CEAs (CCEA) all acted as history functions over a given property. Each of them reported whether a particular property held for a certain amount of time over a given period. For example, one CCEA could report for how long a particular application had been the active application on its host machine over a given period, while another could report how long a person had (or had not) been talking for, a location history reported how long an entity had been at a particular location, and a keyboard input CCEA and a mouse input CCEA could report how much input activity had occurred over a given period, based on counts of the number of keyboard and mouse events that occurred.

This small set of CEAs was enough to provide all of the properties required by the large set of situations developed for each of the situation-aware applications.

### 5.3.2 Developing situations, specifications and customisations

A total of thirty-two different situations were developed, with many having several alternative specifications and customisations. These included situations based on the use of devices and applications, as well as situations from a domestic setting, a public setting and a University work environment. Customisations were created using the specially developed graphical tool described in Section 5.2.

The first set of situations focussed on a particular individual's use of devices and applications. These situations could be applied anywhere the devices and applications could be used. In this category, it was sought to capture situations such as 'Viewing a document' and 'Editing a spreadsheet'. In fact, this style of situation, where a user is primarily working with a particular type of media, appeared several times. For each type of media, three situations were created - a general 'Working with media', and inheriting from this the more specific 'Viewing media' and 'Editing media'.

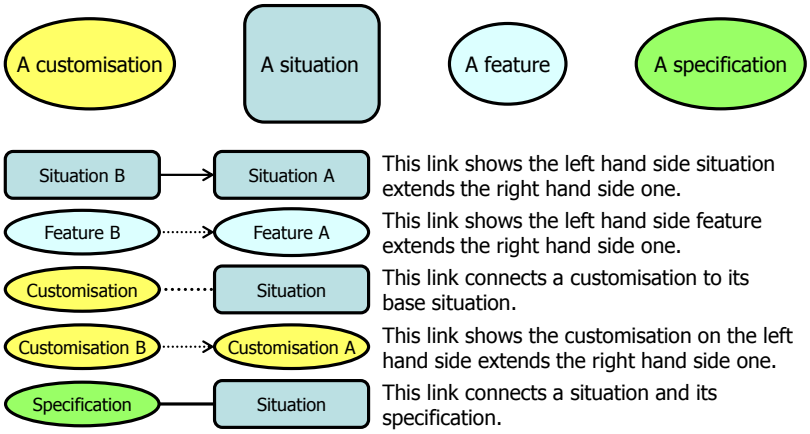



Figure 5.3: A key for the figures included in this section.

Customisations based on this set of situations were created for a variety of media types. For example, taking the word-processor document type specifically, three customisations were created to represent the 'Working with a document', 'Viewing a document', and 'Editing a document' situations. The other media types included spreadsheets, slides, web pages, e-mail, photographs, movies, Go game records, code and ontologies. These specific media types were chosen as they are frequently used by the author, but this list could easily be extended to include almost any media type.

The specifications written for these situations followed a general form. The 'Working with media' situation would be recognised by detecting that a particular application was running on the machine currently in use by the person who is the situation index, and also that it is the active application on the host machine, and has a particular document open. The 'Viewing media' specifications add to this the constraints that keyboard and mouse activity is low, while the 'Editing media' specifications add that the keyboard and mouse activity is high.

Other situations in this category include listening to music, making a (VoIP) phone call, making a video call and chatting (using a chat program). Each of these situation's specifications were implemented in a similar way. Figure 5.4 illustrates each of these situations, specifications, customisations and the relations between them. A key explaining the different shapes, colours and links used in the figures in this section is given in Figure 5.3. The omissions that have been made in the interests of space in the figures in this section, are included in full in the appendices in Section B.1.

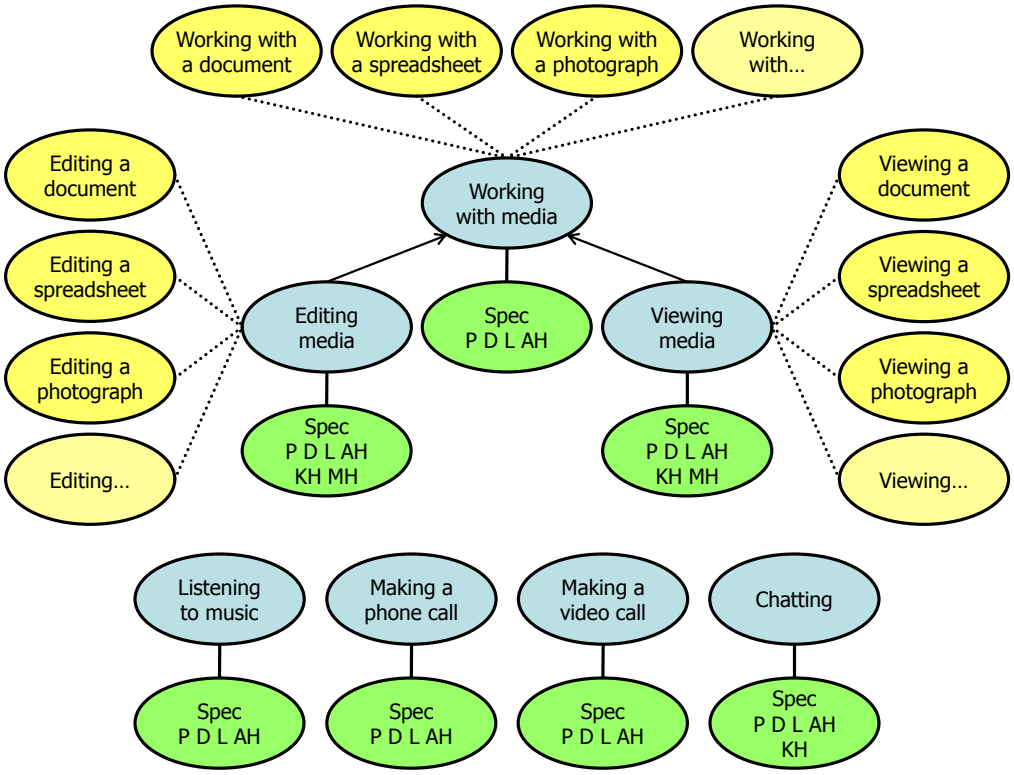



Figure 5.4: Situations that focus on an individual's use of devices and applications. In the interests of space, only a selection of the customisations for each of the different media types is shown.

The domestic setting situations that were implemented included a small number of simple home environment situations, such as a 'Sleeping' situation, whose specification checked that it was after a certain time at night and the user was in their bedroom. Also, an 'Entertaining' situation that detects when the user is entertaining guests by recognising the presence of a number of people in addition to the house's inhabitants. This situation was extended with the more specific variations 'Family visit' and 'Entertaining friends'. A 'Studying' situation was also implemented that recognised when one of the inhabitants was working on a computer while in the study room of the house. 'Watching a movie' and 'Making a video call' situations were also implemented, that are similar to their counterparts described above for a single user, though this time are based within a particular room in the house and include all of the people that are watching the movie or are involved in the video call. These situations are illustrated in Figure 5.5.

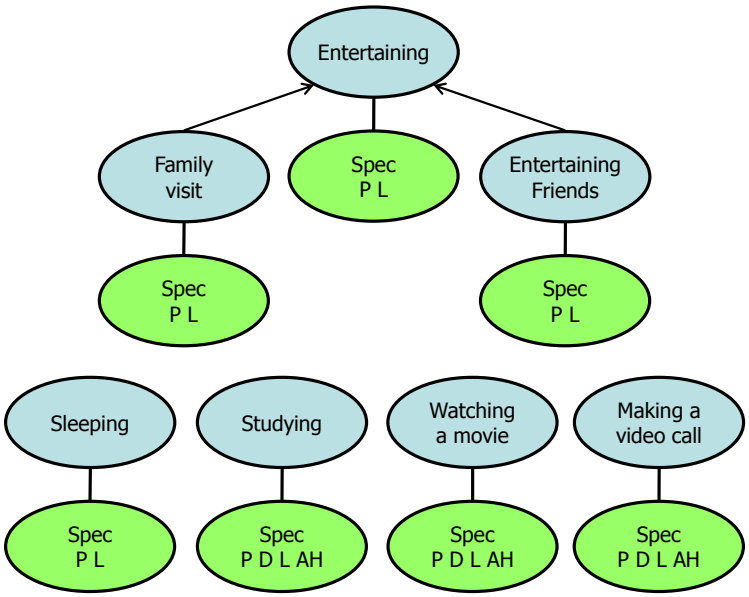



Figure 5.5: Domestic situations.

The situations that were implemented for a public setting were deliberately kept simple, as they targeted resource-constrained devices, such as mobile phones and PDAs, operating in an ad hoc environment. Included in these were a 'Dining' situation, that reported at which café or restaurant and with whom the user was dining, a 'Shopping' situation that reported where and with whom the user was shopping, similarly a 'Swimming' situation, and situations that detected when the user was waiting for a train or for a bus. Each of these situations was recognised by detecting the user's location and its type, as well as the presence of the other people in the situation and their relation to the user, using the person and location history CEAs. Figure 5.6 illustrates these.

It was for the University work environment that the most complex situation specifications were created. Situations in this category included general work environment situations, a variety of meeting situations, presentations, as well as a number of University teaching situations.

The general work situations included 'Working at desk' and 'Working in office' situations, as well as a number of situations that detect different types of breaks, including lunch, coffee breaks and bathroom breaks. The specifications for these were mainly based on the location and person DCEAs and time. Diagrams of these situations are included in the appendices in Section B.1.3.

A general 'Meeting' situation was implemented that was extended by several

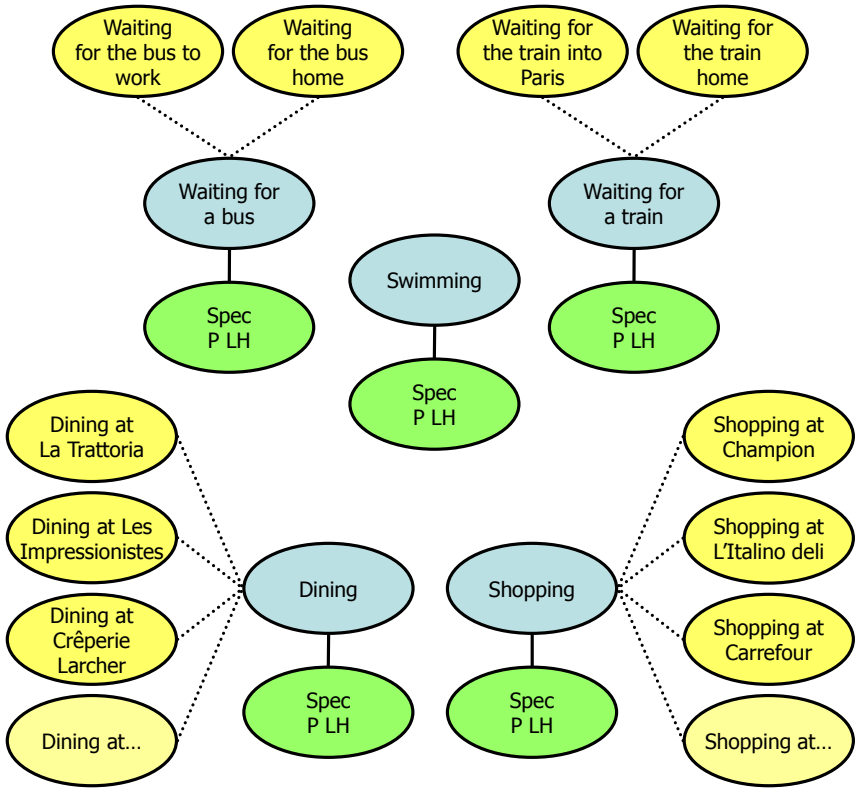



Figure 5.6: Public situations. In the interests of space, only a selection of the customisations for dining and shopping situations are shown.

more specific variations, including a 'Group meeting' situation that recognised meetings of a particular research group or administrative group, a 'PhD meeting' situation that detected a meeting between a PhD student and one or more of his/her supervisors, as well as a 'Demonstrator meeting' situation that recognised a meeting held with the lecturers and demonstrators of a particular course. The meeting situations are depicted in Figure 5.7.

Each of the specialised meeting situations had at least three alternative specifications. One specification was based on recognising the situation as occurring at a fixed time and place and the location of the attendees, representing a simple specification that could be authored by a local administrator for convenience. Another specification was based on the location of the attendees, matched against a calendar entry stating that the particular type of meeting should be taking place at that time and place. A third specification was tailored to the particular type of meeting, for example part of this specification for the group meeting required that more than 75% of the attendees were members of the same group, and similarly

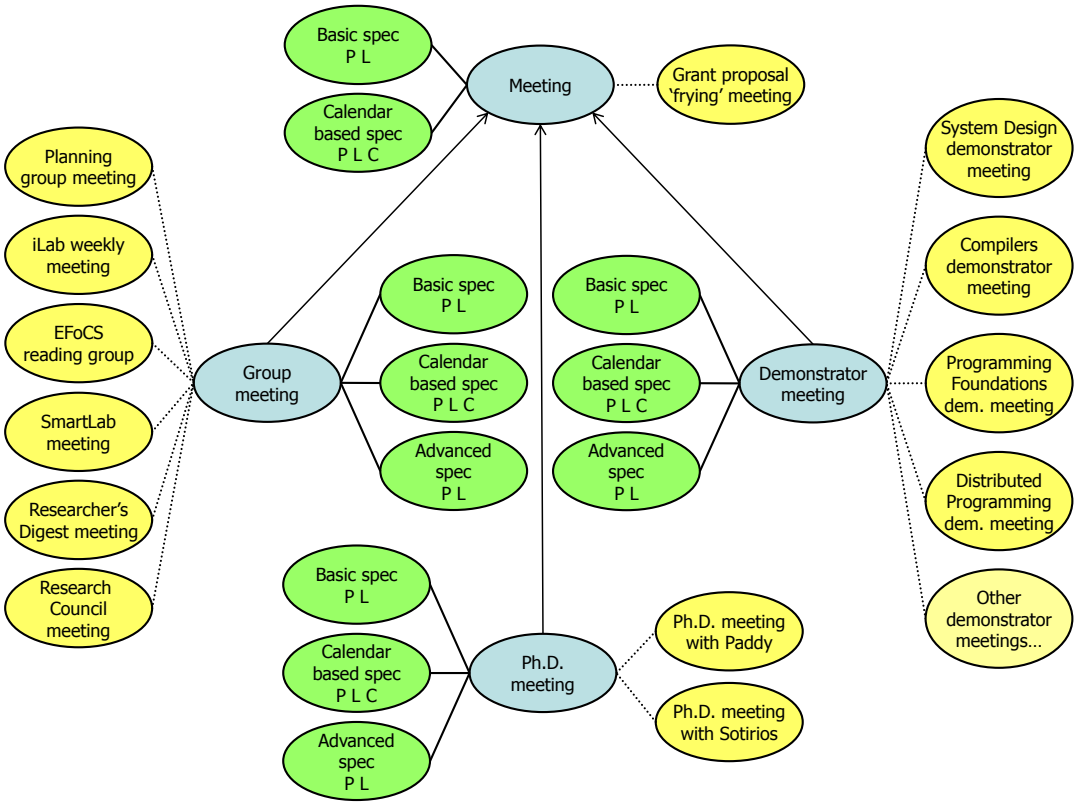



Figure 5.7: University meeting situations. In the interests of space, only a selection of the customisations for each of the different demonstrator meetings are shown.

the part of this specification for the PhD meeting detected the PhD supervisor and PhD student relations between the two attendees. This offered a large degree of versatility in detecting these situations, and allowed the overall confidence to be increased by fusing the results of each of these specifications.

The presentation situation that has served as the running example in this thesis was implemented with five different specifications. These included two specifications similar to those described above - one based on a fixed time and place and attendees' locations, and another based on attendees' locations and a calendar entry. Also implemented were the three specifications featured earlier - a coarse-grained specification that could only identify presentation attendees, and two fine-grained specifications that could differentiate between speakers and audience members at the presentation, one based on fine-grained location information, and the other based on detecting whether a person was speaking or not. These are depicted in Figure 5.8.

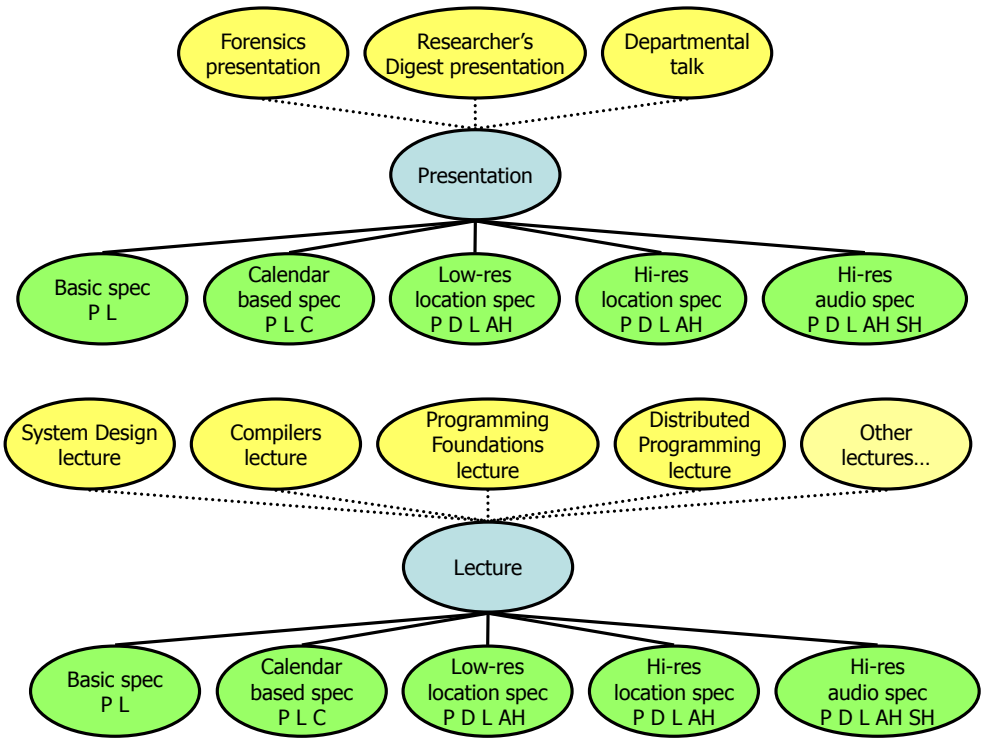



Figure 5.8: University presentation and lecture situations. In the interests of space, only a selection of the customisations for each of the different lectures are shown.

The University teaching situations included situations that recognised lectures, labs and tutorials. The lecture situation is illustrated in Figure 5.8. It featured a set of five specifications comparable to the Presentation specifications. For both the 'Lab' and 'Tutorial' situations, a set of three specifications was created similar to the meeting specifications - one based on a fixed time and place and attendees' locations, a calendar-based specification, and one that detects that the supervisors and students related to that particular lab or tutorial are present. Figures for the labs and tutorials follow a similar structure to the meeting example, and so have been omitted here in the interests of space. However, they are included in the appendices in Section B.1.3.

In addition to these situations, further customisation to the particular environment was achieved by creating several customisations for each of the meeting, presentation and teaching situations. For example, customisations of the group meeting situation were created for each of the specific research groups and administrative groups within the department, as well as customisations for each of

the lectures, labs and tutorials that took place for each course run within the department. This offered a large degree of flexibility and convenience in tailoring situations to the particular details of the local environment.

Table 5.1 presents a summary of the situations that were developed, showing the number of specifications that were written for each, the set of CEAs they were based on, as well as the number of customisations that were created. It can be seen that in these deployments, there were both more specifications and more customisations than there were situations. For customisations, there were seven times as many. The flexibility that this offers stems directly from the novel modelling features that the middleware provides. Furthermore, the data demonstrate that from just a handful of CEAs a large variety of situations can be created across several domains.

### 5.3.3 Developing the applications

This section describes three applications that demonstrate a variety of styles of situation-aware applications the pervasive situation determination middleware can support. The availability checker, presented in Section 5.3.3.1, is an example of an application that performs ad hoc situation queries about a person, device or location in real time, whether the target may be in the local environment or in an unknown remote environment. The mode manager, presented in Section 5.3.3.2, is an example of an application that continuously monitors and autonomously reacts to particular local situations in real time, constantly adapting to changes in sensing infrastructure and environment as the user travels from place to place throughout their day. Finally, the situation-enhanced file search application, which is presented in Section 5.3.3.3, is an example of an archival application that detects and records all of the situations the user is involved in and allows later searching and analysis of those situations for information and artefacts associated with them.

#### 5.3.3.1 The availability checker application

The availability checker application was the simplest of the three applications. It gives the user the ability to select a particular user, device or location, and the application will report the situation(s) they are currently involved in. This application helps the user in planning their actions. For example, a user may decide not to telephone someone at the current time as the availability checker

| Situation | Num. Spec. | CEAs | Num. Cust. |
|---|---|---|---|
| Use of devices and applications | | | |
| Working with media | 1 | P, D, L, AH | 10 |
| Viewing media | 1 | P, D, L, AH, KH, MH | 10 |
| Editing media | 1 | P, D, L, AH, KH, MH | 10 |
| Listening to music | 1 | P, D, L, AH | 0 |
| Making a phone call | 1 | P, D, L, AH | 0 |
| Making a video call | 1 | P, D, L, AH | 0 |
| Chatting | 1 | P, D, L, AH, KH | 0 |
| Domestic setting | | | |
| Sleeping | 1 | P, L | 0 |
| Entertaining | 1 | P, L | 0 |
| Family visit | 1 | P, L | 0 |
| Entertaining friends | 1 | P, L | 0 |
| Studying | 1 | P, D, L, AH | 0 |
| Watching a movie | 1 | P, D, L, AH | 0 |
| Making a video call | 1 | P, D, L, AH | 0 |
| Public setting | | | |
| Dining | 1 | P, LH | 6 |
| Shopping | 1 | P, LH | 8 |
| Swimming | 1 | P, LH | 0 |
| Waiting for a train | 1 | P, LH | 2 |
| Waiting for a bus | 1 | P, LH | 2 |
| University setting | | | |
| Working at desk | 1 | P, D, L | 0 |
| Working in office | 1 | P, D, L | 0 |
| Private phone call | 1 | P, D, L, AH | 0 |
| Bathroom break | 1 | P, LH | 0 |
| Coffee break | 1 | P, LH | 0 |
| Lunch break | 1 | P, LH | 0 |
| Meeting | 2 | P, L, C | 1 |
| Group meeting | 3 | P, L, C | 8 |
| PhD meeting | 3 | P, L, C | 2 |
| Demonstrator meeting | 3 | P, L, C | 41 |
| Presentation | 5 | P, D, L, C, AH, SH | 3 |
| Lecture | 5 | P, D, L, C, AH, SH | 50 |
| Lab | 3 | P, L, C | 41 |
| Tutorial | 3 | P, L, C | 30 |
| Total | | | |
| 32 | 53 | | 224 |

Table 5.1: A summary of the situations, specifications and customisations that were developed.

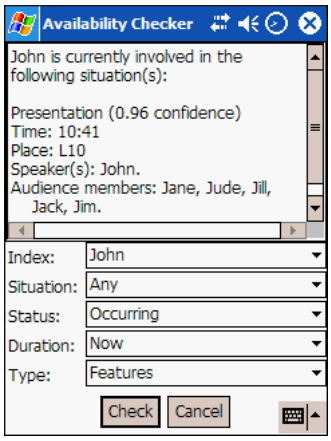



Figure 5.9: The availability checker application running on a Pocket PC.

reveals that they are currently in a meeting, or a user may check that there is currently nothing happening in a particular meeting room, and so it can be used for a quick, ad hoc meeting.

This application also acted as a useful debugging tool, as it provided a convenient means to check what the situation determination middleware believed the currently occurring situations were for any given person, device or location.

Figure 5.9 shows the main interface of the availability checker application. It is the Pocket PC version that is shown, though thanks to having been written in Java, it could also be run on a laptop or desktop machine.

The availability checker offers several options to the user. First, the user can select which person, device or location they want as the index of the situation request. Note that the application allows the user to select a display name for the index, and stores them for subsequent use, which is more convenient than having to remember the index's UUID.

Next, the user can select which specific situation or customisation they wish to detect, or choose all situations to be detected.

After that, the user can select to be notified if the situation is occurring, or when the selected situation starts to occur, or when the selected situation ends. The situation-aware application agent (SAA) already defines default methods that detect when a situation starts or ends that are available to situation-aware applications. The use of these methods is optional, and the application can always infer these transitions themselves by using the stream of situation responses directly. This provides simplicity in application development when the default implementation is adequate, as well as the opportunity to more exactly match

the specific needs of the application when required.

The user can also set the length of time the request will remain active through the duration option. For example, the user may request that the availability checker notify them if a particular person starts to take a coffee break over the next hour.

Finally, the user can select whether the situation response is reported in short, feature set or full form. The short form can be used when the user simply wants to know what situation(s) are occurring, as in the case mentioned above where the user checks if a room is vacant. The feature set form is most useful when the user wants to know about the features of the reported situations. For example, if a particular person was in the audience of a presentation. The full mode proved to be useful for debugging, as it reported all of the relevant details of each of the entities involved in a situation.

It is natural that the user should wish to control the distribution of their situation information and restrict who may or may not be able to discover where they are or what situations they are involved in. To revisit the characters from the opening scenario of Chapter 4, John may wish to temporarily block his wife Angela from being able to detect his situations while he was surreptitiously buying an anniversary gift, for example. Within the architecture, it is possible to implement such blocking as well as several other privacy policies. All situation requests are sent to an individual's Index Server Agent (ISA) and all responses to that request are sent back to the ISA before being forwarded to the application that originally made the request. Therefore, the ISA may act as a mediator. This would allow the ISA to block requests based on properties such as whom the request is from or which particular situations are requested. Furthermore, it allows the ISA to alter situation reports in the response. For example, a user could configure the ISA to obfuscate situation reports requested by particular sources to show their location as only which city they were currently in and not a specific building or room, or to report meeting situations only at that granularity and not which particular meetings or with whom they are held. In this manner, the ISA acts as the user's personal situation firewall.

Addressing the complete set of privacy concerns that may be encountered in a pervasive computing environment is beyond the scope of this work. However, ways in which the implementation of a full suite of privacy principles may be integrated with the architecture presented here is the subject of the discussion in Section 6.2.2.

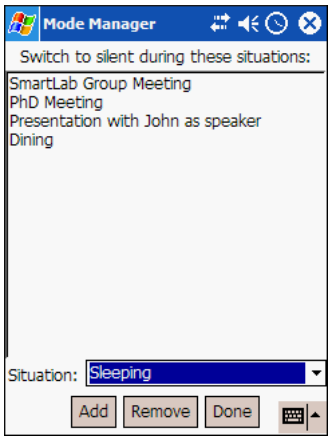



Figure 5.10: The configuration screen of the mode manager application.

Although this application is very simple, it demonstrates the full functionality of the situation determination middleware - it can interactively detect situations and customisations for a given person, device or location, either in the local environment, or in a distant environment.

#### 5.3.3.2 The mode manager application

The mode manager was an interesting application that allowed the user to configure their devices to automatically switch to a particular mode of operation when an appropriate situation was detected.

Two types of mode manager application were developed. The first was a Pocket PC-based application that muted the device when any of the appropriate situations were detected. The second was a Windows-based application that suspended the system's screen saver during the selected situations.

Figure 5.10 shows the configuration screen of the Pocket PC version of the mode manager application. From here, the user can configure which set of situations the device will react to.

The implementation of this application also relies on the situation starting / situation ending methods provided by the SAA. Both muting the Pocket PC and disabling the screen saver were achieved using Windows system calls. Further details of this are given in the appendices in Section C.2.

This application shows that the situation determination middleware can be used successfully to realise the notion of a 'smart device' - devices that autonomously and continuously adapt their behaviour to that which is most ap-

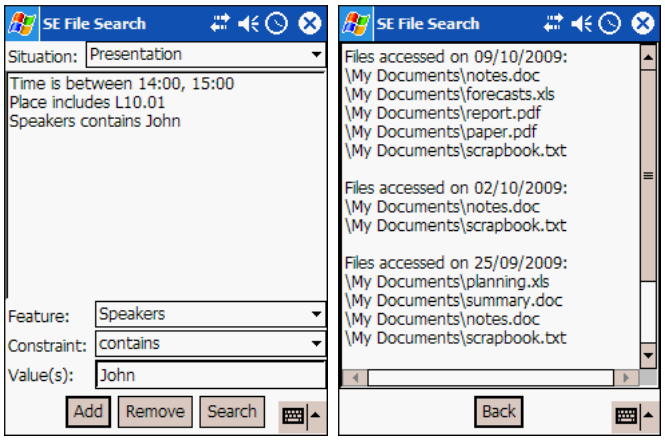



Figure 5.11: Creating search criteria with the situation-enhanced file search application (left), and browsing the results (right).

propriate for the current conditions they operate in.

#### 5.3.3.3 The situation-enhanced file search application

The situation-enhanced file search application is the most complex of the three applications developed. It allows the user to search for files they have accessed in the past, using details of the situations in which the files were accessed as search terms.

There are two parts to this application. One part provides the search interface and the query results, while the other part runs in the background continually collecting situation reports for the user and monitoring file access.

The left of Figure 5.11 shows the search interface and query result screens. Here the user can incrementally build up search criteria for the situation they wish to find. When the criterion is complete, the application will list any files that were accessed during matching situations. An example result is shown on the right of Figure 5.11. The results panel features a 'Back' button that allows the user to return to the search criteria, allowing them to iteratively refine their search.

The part of the application that monitors file access was based on the freely available File Watcher Utilities project [142]. This was used to monitor the local device for files that were opened, changed or renamed. The device requests notification of any situations that are occurring, using the current user of the device as the situation index. When any of the file events are detected, they are

paired with each of the situations that are currently occurring, and each pair is written to a database. This database is then later searched by the interface part.

In this example, detecting the actions involved in using files is performed by the situation-aware application itself, rather than the middleware. These actions are detected locally, along with their co-occurrence with one or more situations. This is appropriate, as simple actions such as a particular person accessing a particular file during a presentation is an incidental detail of a particular instance of a situation. There could be any number of such actions. Therefore, it is not appropriate to explicitly include these in a situation's features, where the focus is to capture the essential characteristics of the situation in general.

More complex actions however, such as taking notes, can be modelled as situations themselves (similar to the "Using devices and applications" situations presented earlier in this section). Then, a situation-aware application can detect co-occurrences of the action and main situations, by requesting both for the same situation index.

This application exemplifies how the situation determination middleware can be exploited to offer a powerful, high-level search interface in user applications, as well as utilising an archive of previous situations.

## 5.4 Middleware Performance Analysis

This section presents a performance evaluation of the pervasive situation determination middleware. Section 5.4.1 describes the experimental set up that was used, giving details of the equipment employed and the deployment environment. Section 5.4.2 reports several performance measurements tested in this environment, while Section 5.4.3 provides algorithmic analyses of key components of the middleware.

### 5.4.1 Experimental set up

The performance measurements presented in the following section were based on a deployment of the situation determination middleware prototype in a University environment. Specifically, the middleware was deployed within a Computer Science department, using the actual meeting rooms and laboratories available there.

Three different classes of computer were used in these measurements. First,

there were desktop PCs with a 3.4 GHz Intel Pentium 4 processor and 1 GB of RAM, which ran Debian GNU/Linux 3.1. Second, there was a Dell Inspiron 6000 laptop, with a 1.5 GHz Intel Pentium M processor and 1 GB of RAM, which ran Windows XP. And third, there was a collection of HP Pocket PC h5500 handheld computers with a 400 MHz Intel XScale processor and 128 MB RAM, running Windows CE 4.2.

The desktop PCs were connected to the department's 100 Mbps-BaseT Ethernet network, while the laptop and Pocket PCs connected via a WEP-encrypted 802.11b wireless network. When operating in ad hoc mode, an ad hoc wireless network was hosted by the laptop, using an Asus SpaceLink 11 Mbps 802.11b wireless network card.

In order for the middleware to correctly detect locations within the department, a location model had to be created for the department, the details of which are given in Appendix A.

The exact configuration varied for different measurements. Details of the specific configuration are given for each case below alongside the results.

### 5.4.2 Performance figures

In considering the performance of the situation determination middleware, the two following aspects are central to the utilisation of the middleware by situation-aware applications.

The first is the round trip time of a situation request. That is, the time it takes from a situation-aware application issuing a situation request, until it receives the response. This measure is significant as it acts as a summary of the performance of the system as a whole.

A device hosting a situation-aware application will rarely operate in isolation. Whether operating with rich infrastructure available or in ad hoc mode, a device must first discover and join a network of other devices to fully utilise the middleware. The second measure of interest is then the join times of devices connecting to both infrastructure-based and ad hoc environments.

In this section, a number of performance measures of the situation determination middleware are presented relating to these aspects. Two measures concern issues of scale. The first demonstrates that the performance of the middleware scales well for a set of individual situations as the number of entities involved in the situations increases from typical to larger sizes. The second shows that

the performance of the middleware also scales well as the number of situations simultaneously recognised by the middleware increases from light to heavy loads.

The increasing situation size measurements were first conducted using the environment-based mode of operation, where the situation recognition effort was offloaded to situation determination agents (SDAs) hosted on desktop machines in the environment. These measurements were then repeated using the ad hoc mode of operation, where the mobile devices involved in a situation do not rely on dedicated infrastructure for its recognition, but form an ad hoc network and recognise the situation themselves.

Also demonstrated are the gains in performance offered by the situation determination middleware as the configuration is altered from using a single SDA to sharing the recognition load between multiple SDAs.

In addition, a breakdown of the computational resources consumed by the middleware is presented, detailing the relative amounts of each component part for the situation recognition process.

#### 5.4.2.1 Increasing situation size

This section looks at round trip time (RTT) and join time measurements of both environment-based and ad hoc modes of operation.

A special tool was developed to collect these measurements. The tool was a situation-aware application that could record the times that events such as joining the agent substrate, sending situation requests and receiving situation reports occur.

The round trip time measurement captured by the tool represents the amount of time between the application sending a situation request and receiving a situation report in response. The measure incorporates the time consumed by sending the situation request out, the system gathering the necessary information it requires to recognise the situation(s) that has been published by the hardware and software sensing infrastructure within the environment, the cost of performing the recognition itself, and the delivery of the results back to the application. The join time measurement captured by the tool represents the length of time it takes from a device first discovering the address of the Environment Manager Agent (EMA) in the current environment, until it has connected to the agent substrate, interacted with the EMA to appropriately join the situation recognition environment, and is ready to send a situation request. The components of both of these

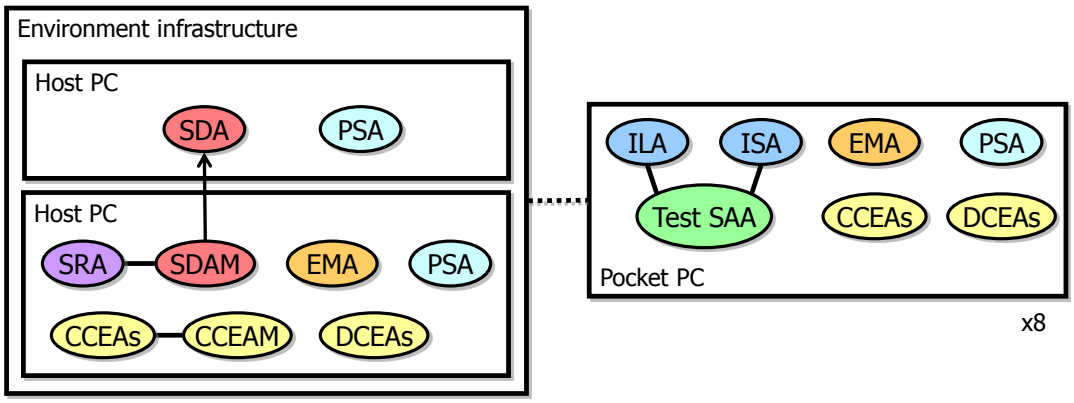



Figure 5.12: This diagram illustrates the environment-based set up used to measure the RTT and join times. 'Test SAA' indicates the application used to record the measurements.

measurements are summarised below, where $T_X$ indicates the time taken for the individual component $X$:

$$T_{RTT} = T_{Send\ request} + T_{Gather\ sensor\ data} + T_{Recognise\ situation} + T_{Deliver\ response}$$

$$T_{Join} = T_{Discover\ EMA} + T_{Join\ agent\ substrate} + T_{Join\ environment} + T_{Prepare\ request}$$

### Environment mode measurements

This section presents measurements from a deployment using the environment-based mode of operation. That is, dedicated infrastructure is available that hosts the SDAs, and mobile devices can offload situation recognition effort to them. An SDA was run on a desktop PC (specifications were given in Section 5.4.1), and another desktop PC hosted the other infrastructure agents, such as the EMA and the Situation Repository Agent (SRA). Pocket PCs were used to represent a person, and hosted the Context Entity Agents (CEAs) that reported the person's location and personal profile information, as well as each person's Index Server Agent (ISA). The special measurement application was also run on a Pocket PC. This set up is illustrated in Figure 5.12.

Table 5.2 presents the performance measurements of increasing the size of a set of situations. The table shows the average RTT and the average join time for a set of four different situations as the number of people involved in each situation increases. These situations were the 'Group Meeting', 'Presentation', 'Lab' and

| Group meeting | | | | |
|---|---|---|---|---|
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 9.114 | ±0.184 | 3392 | ±165 |
| 4 | 9.236 | ±0.187 | 3626 | ±218 |
| 8 | 9.675 | ±0.181 | 3583 | ±149 |
| *16* | 10.074 | ±0.309 | 3385 | ±101 |
| *32* | 11.512 | ±0.315 | 4063 | ±133 |
| *64* | 13.666 | ±0.359 | 4275 | ±349 |
| Presentation | | | | |
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 9.336 | ±0.242 | 3574 | ±124 |
| 4 | 9.468 | ±0.230 | 3485 | ±103 |
| 8 | 9.766 | ±0.204 | 3164 | ±287 |
| *16* | 10.909 | ±0.267 | 3187 | ±148 |
| *32* | 13.846 | ±0.313 | 3532 | ±121 |
| *64* | 18.739 | ±0.479 | 3490 | ±160 |
| Lab | | | | |
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 8.434 | ±0.179 | 3702 | ±193 |
| 4 | 8.700 | ±0.213 | 3542 | ±232 |
| 8 | 9.012 | ±0.159 | 3362 | ±163 |
| *16* | 9.740 | ±0.246 | 3575 | ±155 |
| *32* | 10.710 | ±0.217 | 4124 | ±144 |
| *64* | 12.114 | ±0.185 | 4187 | ±125 |
| Researcher's digest meeting | | | | |
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 9.959 | ±0.223 | 3378 | ±167 |
| 4 | 10.604 | ±0.294 | 3655 | ±184 |
| 8 | 11.375 | ±0.318 | 3516 | ±200 |
| *16* | 12.289 | ±0.205 | 3775 | ±131 |
| *32* | 13.455 | ±0.305 | 3876 | ±193 |
| *64* | 15.061 | ±0.438 | 4173 | ±122 |

Table 5.2: The mean round trip times (RTT) and mean join times for a selection of situations in an environment-based set up, as the size of the situations increases. All values are shown in milliseconds, and a 0.95 confidence level was used.

the customised 'Researcher's Digest Meeting' as described earlier in Section 5.3.2. These four were chosen as their specifications provide a variety of different forms. For 'Group Meeting', the specification must recognise each individual in the situation, but also match on membership of a common group. 'Presentation' involves fine-grained location information. The 'Lab' situation combines person, calendar, location and time information. While the 'Researcher's Digest Meeting' customisation of the 'Group Meeting' situation requires calculating the percentage of people present that are playing the attendee role, and incurs the additional processing of the customisation itself. All of the specifications that were developed for each of these situations, as described in Section 5.3.2 were used, as would be done in a real deployment to support dynamic adaptation to changing sensing infrastructure.

In Table 5.2, and in the following tables, each RTT measurement represents the mean value of 300 samples, and each join measurement the mean of 30 samples. These numbers were chosen to reflect a single situation recognition session that would be performed by a situation-aware application. That is, it first joins the network, issues a situation request, and then receives 10 situation reports during which the application would react to the situation or otherwise use the information. A total of 30 sessions were collected for each configuration, which was the largest number of samples that could practically be collected within the time available. The table also includes the confidence intervals for both of the measurements. The size values, listed in the leftmost column, represent the number of people that were involved in the situation. For example, a value of two represents that were two people involved, and therefore two Pocket PCs were running in the environment, each reporting location and personal properties for their respective user. For larger sizes the number appears in italics. This indicates that a single Pocket PC represented more than one person. A total of eight Pocket PCs were available to conduct these measurements, so in order to simulate larger situation sizes, a single Pocket PC had to represent more than one user. For example, an italicised value of sixteen indicates that sixteen people were involved in the situation, and that eight Pocket PCs were running in the environment, each reporting location and personal properties for two respective users.

The RTT measurements for each situation shown in Table 5.2 can be compared graphically in Figure 5.13. Overall, each of the situations shows an acceptable round trip time. Even at the largest situation size, all response times are under

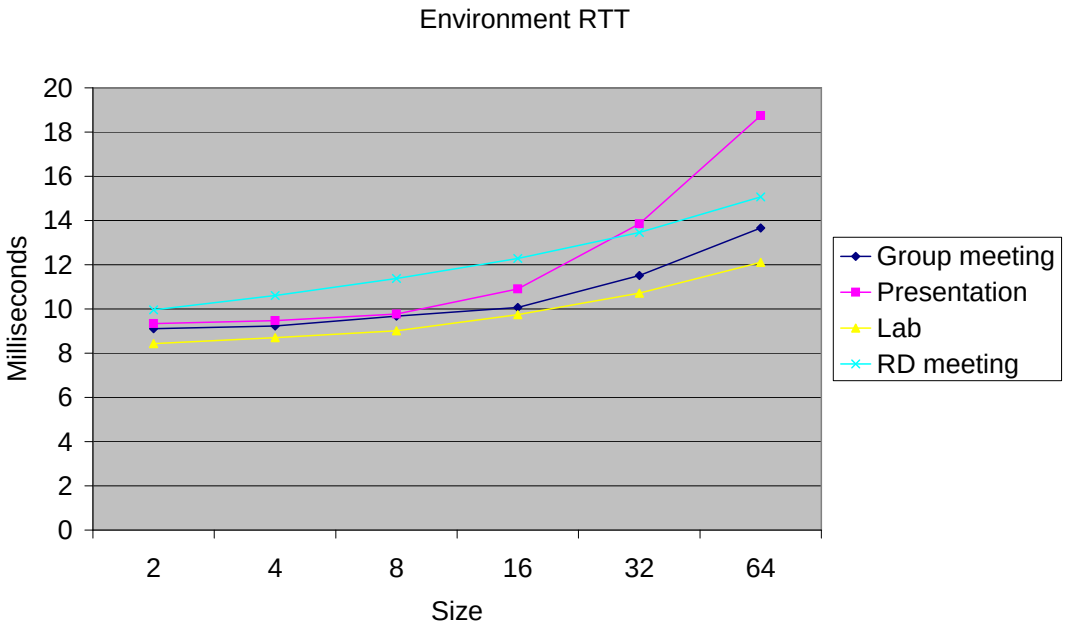



Figure 5.13: A graphical comparison of the mean round trip times (RTT) for a selection of situations in an environment-based set up, as the size of the situations increases.

nineteen milliseconds. For each doubling in situation size, the round trip time increases by only ten percent on average. Such low round trip times make the situation determination middleware suitable for use in interactive applications.

The join time measurements shown in Table 5.2 can be compared graphically in Figure 5.14. Join times are long and variable, but are not significantly impacted by increased situation sizes. The join times have a high base limit, the average minimum join time across all situations and sizes is 3,322 milliseconds. This is due to delays introduced by the JADE [118] and jSLP [139] implementations upon which the middleware is based. However, for each doubling in situation size, the join time increases by only three percent on average.

### Ad hoc mode measurements

This section presents performance measurements for the same set of situations shown in the previous section, though this time they were collected from a deployment using the ad hoc mode of operation. That is, there is no dedicated infrastructure available that hosts SDAs and other agents, the collection of devices must recognise the situations themselves. This deployment featured the set

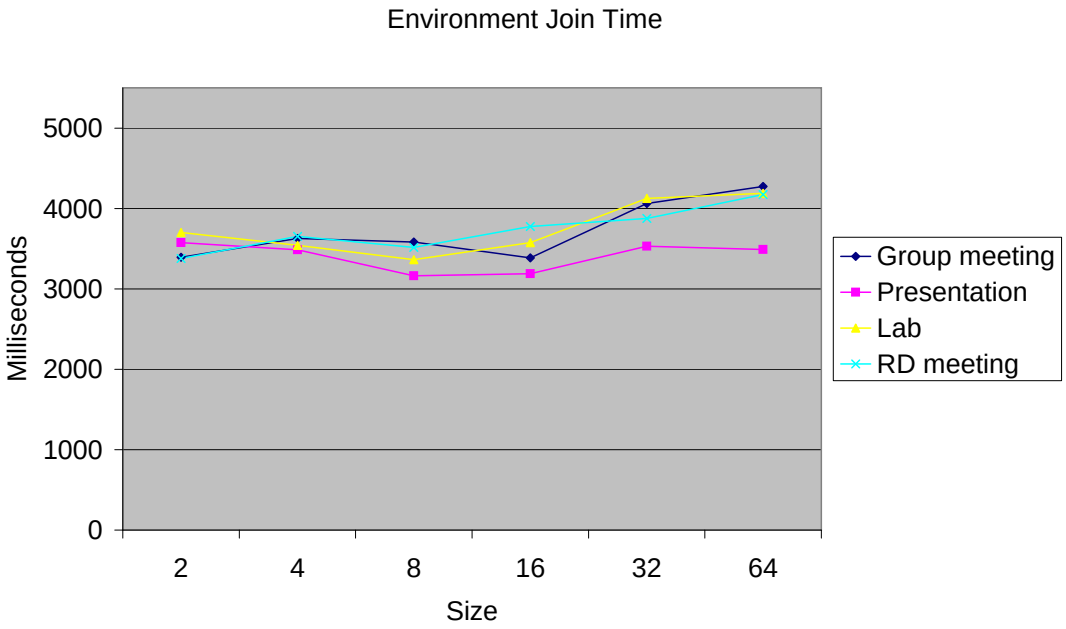



Figure 5.14: A graphical comparison of the mean join times for a selection of situations in an environment-based set up, as the size of the situations increases.

of eight Pocket PCs and the laptop computer (the specifications of these machines are given in Section 5.4.1). The laptop hosted an ad hoc wireless network, which each of the Pocket PCs connected to. This set up is illustrated in Figure 5.15. Using the offload list mechanism described in Chapter 4, the Pocket PCs detect the laptop as being a more powerful host which they are authorised to use, and so forward their situation requests to the SDA Manager (SDAM) agent hosted on the laptop, which uses the laptop's SDA. Other than this, the same set-up was used as before - a Pocket PC represents a single person and reports that person's location and personal profile information, except for the italicised sizes in Table 5.3 where more than one person would be represented by the same Pocket PC.

The round trip times and join times for the ad hoc mode deployment are shown in Table 5.3. Graphical comparisons of these measurements can be made in Figures 5.16 and 5.17 respectively. Notice that the round trip times for ad hoc mode deployment slightly improve on the environment-based deployment round trip times. Despite the situation recognition being performed on a slower CPU (the laptop CPU ran at approximately half that of the desktop machines), this has been offset by the smaller network size. At the largest situation size, all response

| Group meeting | | | | |
|---|---|---|---|---|
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 8.069 | ±0.208 | 4601 | ±120 |
| 4 | 8.750 | ±0.232 | 4544 | ±100 |
| 8 | 9.528 | ±0.262 | 4678 | ±142 |
| *16* | 9.817 | ±0.217 | 4880 | ±142 |
| *32* | 11.279 | ±0.353 | 4683 | ±216 |
| *64* | 13.472 | ±0.495 | 4777 | ±174 |
| Presentation | | | | |
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 8.133 | ±0.283 | 4435 | ±138 |
| 4 | 9.378 | ±0.394 | 4766 | ±117 |
| 8 | 9.974 | ±0.235 | 4486 | ±135 |
| *16* | 11.081 | ±0.547 | 5085 | ±247 |
| *32* | 11.919 | ±0.268 | 4511 | ±135 |
| *64* | 16.626 | ±0.355 | 4818 | ±86 |
| Lab | | | | |
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 7.854 | ±0.220 | 4634 | ±147 |
| 4 | 8.351 | ±0.195 | 4476 | ±127 |
| 8 | 8.910 | ±0.229 | 4569 | ±127 |
| *16* | 9.631 | ±0.329 | 4612 | ±150 |
| *32* | 10.607 | ±0.256 | 4955 | ±110 |
| *64* | 11.657 | ±0.356 | 4886 | ±112 |
| Researcher's digest meeting | | | | |
| Size | Mean RTT | RTT CI | Mean Join | Join CI |
| 2 | 8.782 | ±0.276 | 4568 | ±139 |
| 4 | 9.305 | ±0.274 | 4486 | ±239 |
| 8 | 10.329 | ±0.202 | 4537 | ±173 |
| *16* | 11.416 | ±0.278 | 4704 | ±163 |
| *32* | 12.704 | ±0.211 | 4598 | ±204 |
| *64* | 14.964 | ±0.426 | 4789 | ±188 |

Table 5.3: The mean round trip times (RTT) and mean join times for a selection of situations in an ad hoc-based set up, as the size of the situations increases. All values are shown in milliseconds, and a 0.95 confidence level was used.

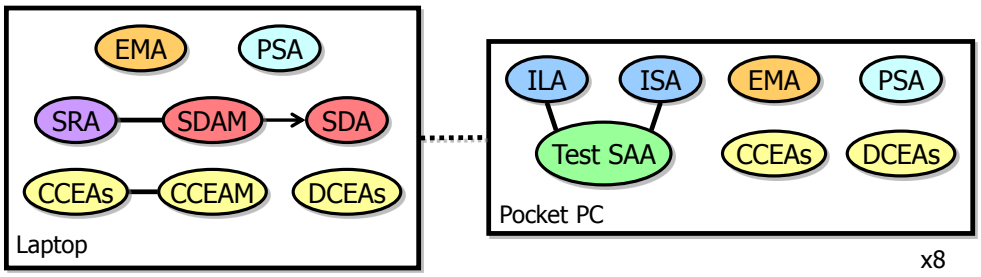



Figure 5.15: This diagram illustrates the ad hoc-based set up used to measure the RTT and join times. 'Test SAA' indicates the application used to record the measurements.

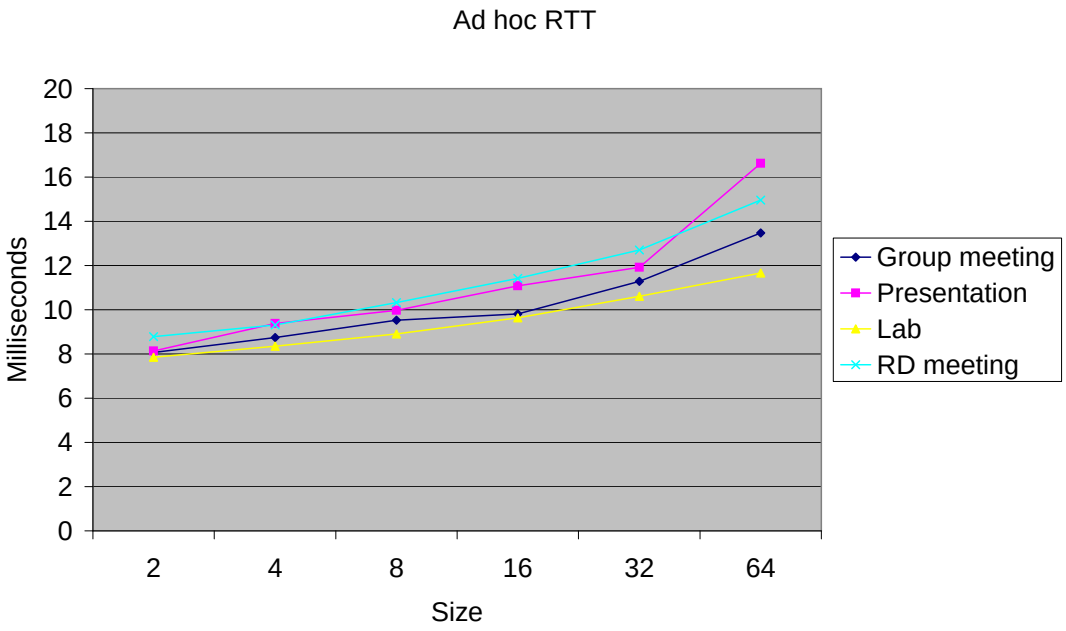



Figure 5.16: A graphical comparison of the mean round trip times (RTT) for a selection of situations in an ad hoc-based set up, as the size of the situations increases.

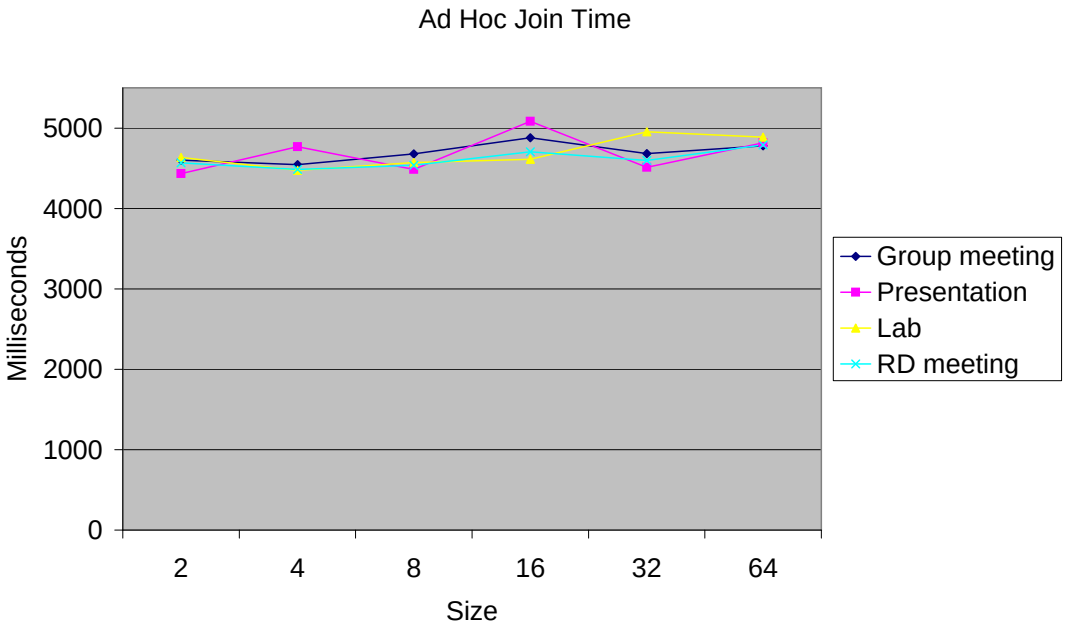



Figure 5.17: A graphical comparison of the mean join times for a selection of situations in an ad hoc-based set up, as the size of the situations increases.

times are under seventeen milliseconds. For each doubling in situation size, the round trip time increases by only 11.6 percent on average, just slightly more than the environment-based case. These low round trip times make the situation determination middleware also suitable for use for interactive applications in an ad hoc setting.

The join times for the ad hoc deployment are slightly higher than for the environment-based case, which is to be expected given the reduced network capacity. They also exhibit a similar pattern in being long and variable. This is again due to delays introduced by the JADE [118] and jSLP [139] frameworks which the middleware relies upon. However, they are also similar in that they are not significantly impacted by increased situation sizes. The average minimum join time across all situations and sizes is 4,485 milliseconds.

While this number is too high to satisfactorily support spontaneous interoperation [143], it is acceptable for the situation determination middleware. It is comparable to the time the user would expect their mobile phone to take to connect a call, and the wait is only incurred once when the user connects their device to the network, not each time a situation is requested. Furthermore, the situations that were developed naturally last in the order of minutes or hours,

making the average join time only a small fraction of a typical duration.

#### 5.4.2.2 Increasing situation recognition load

In this section, a set of measurements are presented that describe the change in round trip times and join times as the number of situations that are recognised simultaneously by the situation determination middleware is increased.

For these measurements, a slightly different set up was used. Here, the change in round trip times and join times was measured for a particular situation, while the number of instances of other situations being recognised in the environment increased. For this, the Presentation situation was chosen to be measured as this demonstrated the steepest round trip time scaling in the previous measurements shown in Table 5.4. This was due to the Presentation situation having the greatest number of distinct roles that must be matched against.

Typically, a user's personal device will represent that user, and for the measurements presented in this section, a Pocket PC has been employed as the personal device. Due to the limited number of Pocket PCs available, it was impossible to have a single Pocket PC represent a single person exclusively for higher numbers of situations, where the total number of people involved exceeded the total number of Pocket PCs available. Therefore, simulation was necessary for these higher numbers of situations. The strategy that was chosen was to have the situation for which the measurements were being made be as realistic as possible, and simulate the increasing load of the other situations. This was realised by fixing the number of people involved in a situation at eight, and using all eight Pocket PCs to represent the people involved in the situation being measured. Agents representing the people and artefacts involved in the other situations were deployed on desktop PCs (the specifications of the machines are given in Section 5.4.1). The physical set up was the same as that shown in Figure 5.12, but with the addition of these PCs.

Table 5.4 shows the results of these measurements taken in the same environment-based deployment as before. Mean round trip time, mean join time, and the confidence interval for each are given. The 'Scale' column indicates the number of situations that were occurring. At the lowest level of scale 1, there was a single instance of a number of group situations that were defined for the Computer Science department domain. These included the 'Lunch', 'Meeting', 'Group Meeting', 'Demonstrator Meeting', 'Lecture', 'Lab' and 'Tutorial' situations. Each

| Presentation | | | | |
|---|---|---|---|---|
| Scale | Mean RTT | RTT CI | Mean Join | Join CI |
| 1 | 16.792 | ±0.288 | 3660 | ±230 |
| 2 | 17.769 | ±0.181 | 4358 | ±366 |
| 3 | 57.235 | ±4.923 | 4285 | ±322 |
| 4 | 152.817 | ±14.525 | 4425 | ±289 |
| 5 | 386.747 | ±60.375 | 4506 | ±238 |

Table 5.4: The mean round trip times (RTT) and mean join times for the Presentation situation in an environment-based set up, as the number of simultaneously recognised situations increases. All values are shown in milliseconds, and a 0.95 confidence level was used.

instance included eight people, matching the size of the Presentation instance. Again, all of the specifications that were developed for each of these situations, as described in Section 5.3.2, were used.

At scale 1, there was a total of eight situations occurring, which included the instance of the Presentation situation that was being measured. Then, each scale number reflects a multiple of this set of situations. So at scale 2, there were sixteen situations occurring, with two instances of each type. At scale 5, forty situations were occurring with five instances of each type. At each scale, the single, realistic instance of the Presentation situation was measured.

The data from Table 5.4 are depicted in Figures 5.18 and 5.19. It can be seen from Figure 5.18 that the round trip times increase more steeply as the number of simultaneously recognised situations increases, than when the size of the situation is increased. For each doubling in the number of situations simultaneously recognised, the round trip time increased by 137 percent on average. However this level of performance is still acceptable - at the heaviest load, while recognising forty situations simultaneously with a single SDA, the average round trip time was just 387 milliseconds. Furthermore, as is demonstrated in the next section, these times can be reduced by sharing the recognition load between multiple SDAs. Figure 5.19 shows that the join times were similar to what has been seen before, having a high base limit, with average times ranging from 3,660 to 4,506 milliseconds.

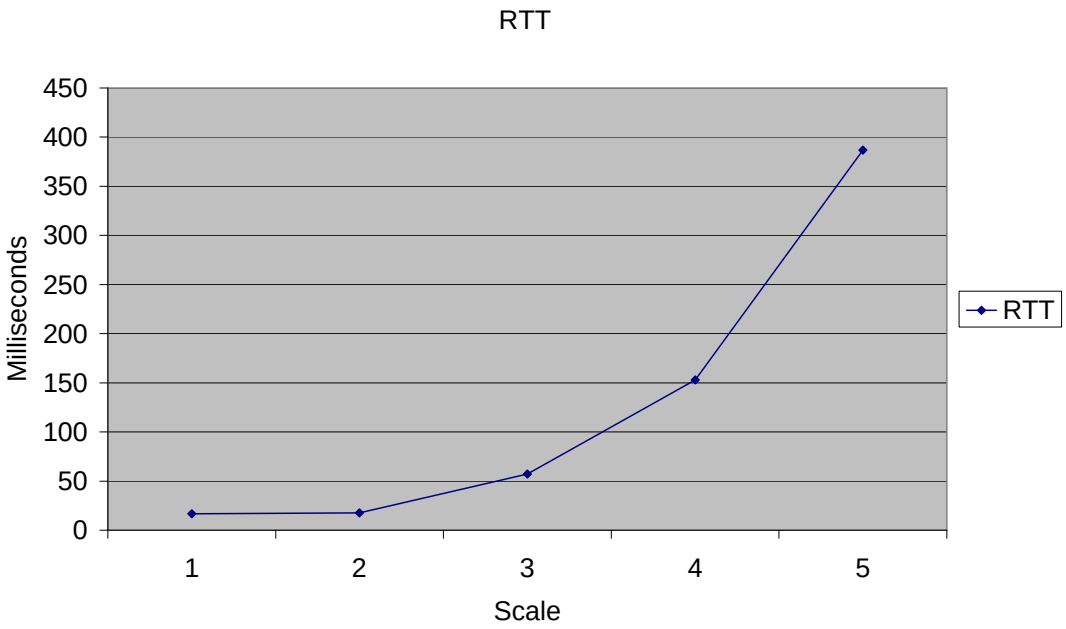



Figure 5.18: A graphical comparison of the mean round trip times (RTT) for the Presentation situation in an environment-based set up, as the number of simultaneously recognised situations increases.

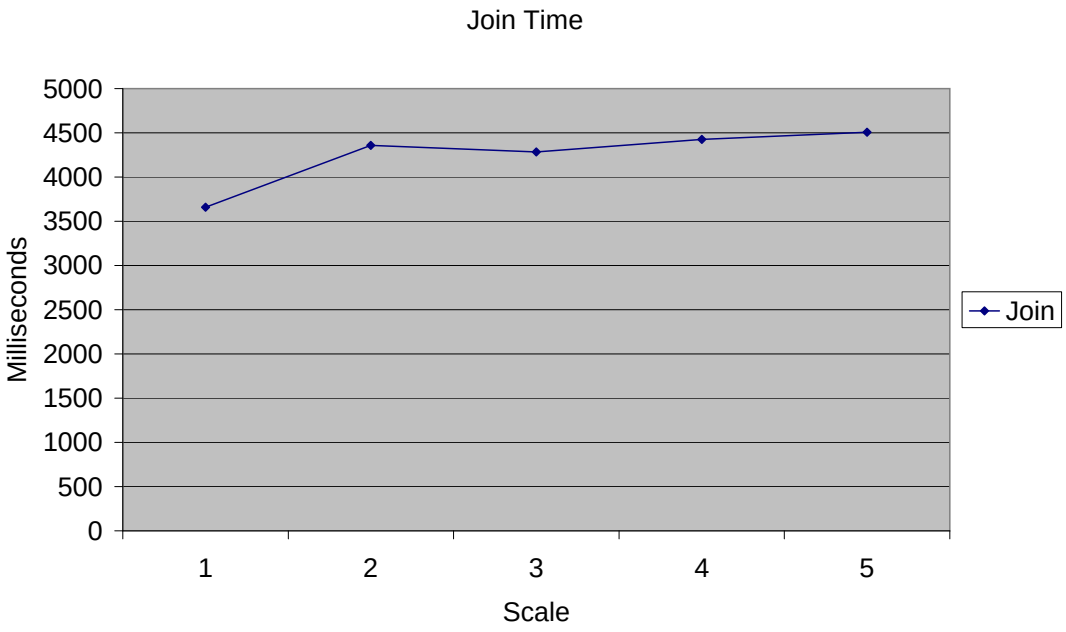



Figure 5.19: A graphical comparison of the mean join times for the Presentation situation in an environment-based set up, as the number of simultaneously recognised situations increases.

| Presentation | | | |
|---|---|---|---|
| SDAs | Mean RTT | % Prev. | % Orig. |
| 1 | 386.747 | | |
| 2 | 231.829 | 59.943 | |
| 4 | 144.112 | 62.163 | 37.263 |

Table 5.5: The mean round trip times (RTT) for the Presentation situation in an environment-based set up under heavy load, as the number of SDAs used to recognise the situations increases. RTT values are shown in milliseconds. %Prev. shows the RTT reduction over the previous number of SDAs, while %Orig. shows the RTT reduction over the original number.

#### 5.4.2.3 Sharing situation recognition load between multiple SDAs

The situation determination middleware makes it easy to deploy multiple SDAs in environments or particular locations that experience heavy load. The load distribution is automatically managed by the middleware.

To demonstrate the benefit of this, the largest scale set up from the previous section was redeployed, though this time the number of available SDAs was increased, and the reduction in the round trip times was measured. The environment-based deployment was similar to that used before, where a dedicated desktop PC hosted infrastructure agents including the EMA, SRA, and ILA, and a separate desktop PC hosted each individual SDA (specifications of the machines were given in Section 5.4.1). Each SDA was assigned to a particular area such that all SDAs processed an equal share of the total number of situations.

The results of these measurements are shown in Table 5.5. For each doubling of the number of available SDAs, the round trip time decreased to 61 percent of the previous time on average.

While Figure 5.18 showed that mean RTT could increase steeply when large numbers of situations were recognised simultaneously by a single SDA, it is demonstrated here that employing multiple SDAs can help to maintain a low RTT under periods of heavy load.

#### 5.4.2.4 Relative resource consumption of situation recognition

Table 5.6 presents the relative resource consumption of the various parts involved in the situation recognition process. For these measurements, the same scale 5

| Relative resource consumption | |
|---|---|
| Component | % Consumed |
| Jess | 40% |
| Jade | 35% |
| Sitdet | 16% |
| Pub/sub | 9% |

Table 5.6: Relative CPU resource consumption of each of the components involved in the situation recognition process.

set up described in Section 5.4.2.2 was used. That is, a single SDA was recognising forty situations simultaneously, comprised of five instances of each of the eight group situations, using all of the specifications developed for each situation. The measurements focus on the situation recognition process, and the SDA in particular, as this is the most resource-intensive element of the middleware, and this set up was chosen as it shows the SDA under continuous, intensive load.

In Table 5.6, 'Jess' is the rule engine component that performs rule-based matching to detect which situations are currently occurring, and consumed the largest proportion of CPU time at 40%. 'Jade', which sends, receives and decodes agent messages, consumed 35% CPU time. The custom situation determination code, 'Sitdet', which integrates and co-ordinates each of the other components, consumed 16% CPU time. While 'Pub/sub', which denotes the components that manage the publish / subscribe messaging, consumed 9% CPU time.

### 5.4.3 Algorithmic analysis

This section presents an algorithmic analysis of the recognition core, an examination of the power savings afforded by the situation determination middleware's grounding feature, as well as a brief look at the complexity of the agent infrastructure and distributed hash table implementation the middleware is based upon.

#### 5.4.3.1 Complexity analysis of the recognition core

As was shown in the previous section, the recognition core consumes the largest share of the computational resources required by the situation determination middleware. Therefore in this section, a more detailed look at the complexity of the recognition core is provided.

The recognition core of the situation determination middleware is based on the Jess implementation [135] of the Rete algorithm [66]. A Rete-based rule engine is known to have the complexity $O(RFP)$ where $R$ is the number of rules, $F$ is the number of facts in the knowledge base and $P$ is the average number of patterns per rule antecedent [66]. From this, it is possible to calculate the complexity of a set of situation specifications.

The complexity measure given above is based on properties of the rule-based representation. However, specifications are not written directly in Jess's rule-based language, but instead are expressed by their ontology-based OWL description. Therefore, it is necessary to express the complexity measure in terms of the properties of the OWL description. The full details of how role and situation specifications are transformed into their rule-based counterparts is given in the appendices in Section C.1, though the relevant parts are included here for convenience.

In this analysis, the parameter $N_s$ represents the size of the set of specifications, that is, the total number of role specifications and situation specifications. $N_s$ is equivalent to $R$.

The parameter $F$ represents the number of facts in the knowledge base. A fact corresponds to an instance of a tuple. A tuple will exist for each relation accessed for each instance of each type of entity referenced in the specification. Therefore, the exact number of tuples will depend on the detail of the particular specification and on the number of entities currently in the environment. However, this can be estimated by the product $N_e * N_r$, where $N_e$ represents the expected number of entities that will appear in the situation, and $N_r$ represents the average number of relations accessed in the role and situation specifications. Note that for a situation specification, the total number of relations accessed will include tuples that represent 'hasRole' relations, one for each role included in the specification. The product $N_e * N_r$ is equivalent to $F$.

Next, we'll examine the parameter $P$, which represents the average number of patterns that appear in the antecedent, or the 'if' part, of a rule. There are three properties of a specification that affect the average number of patterns. First, is the number of Boolean expressions that appear within the expression property of the specification. This also includes any comparisons that declare a confidence threshold. Secondly, there is the number of relations that are accessed within the specification's expression. For example, if 'person.location.type' appeared, meaning access the value of the 'type' relation of the value of the 'location' relation

of the entity named 'person', then two relations would be counted. Thirdly, there is the number of roles that a specification refers to. This is only important for a situation specification, as for a role specification this count will always be zero, as a role specification does not refer to further roles. The parameter $N_p$ represents the average of the sum of the number of Boolean expressions, relations and roles referred to in each specification. $N_p$ is equivalent to $P$.

These parameters can now be combined to derive the complexity metric based on the properties of role and situation specifications:

$$O(RFP) = O(N_s * (N_e * N_r) * N_p) = O(N_s N_e N_r N_p)$$

For a given set of situation specifications, the growth in complexity is linear in relation to the expected number of entities involved in the situation(s).

#### 5.4.3.2 Grounded specification analysis

As reported in Section 4.1, the power consumed by wireless transmission on a mobile device can be anywhere from 6 to 50 times that consumed by the CPU or memory, and it is therefore essential that wireless transmission should be limited where possible, in order to prolong the battery life of the device. This is the motivation behind the specification grounding mechanism described in Chapter 4, where situation specifications that are based entirely or largely on properties available on the device are recognised locally, rather than transmitting all of the properties to an external Situation Determination Agent (SDA).

In Section 5.3 it was shown that it was the set of situations that focused on an individual's use of a particular device or application that exploited the specification grounding mechanism. Out of these, the 'Editing media' situation was quite common, in that it had a large number of customisations, and it also employed the largest set of CEAs. Therefore, it is this situation's specification that is used here to demonstrate the savings afforded through grounded specifications. Furthermore, focus will be given to the case where the user is running the applications on their mobile phone, as in this case the mobile phone's battery is commonly small and limited.

This specification requires a number of pieces of context information to recognise the situation. First, it requires the properties of each application. There are five of these in total, including whether the application is running on the device, and whether it is the active application or not. Furthermore, a Compute Context

Entity Agent (CCEA) calculates whether an application has been active for a certain period of time. All of these properties are produced for each application that is running. Secondly, the CEAs that monitor keyboard and mouse activity each produce a tuple. Thirdly, the location of the device and the user must be included, as well as the 'is being used by' relation between them. Fourth, there are three tuples representing the user, device and active application roles. Finally, the situation response from the SDA must also be included.

This gives a total of six messages for each application that is running, two for the keyboard and mouse monitors, plus three for the location and 'is being used by' relations, another three for the roles, and one more for the situation response. The total messages required per clock tick can be calculated as $6A + 9$, where $A$ represents the average number of applications running on the host device. If $A$ is taken to be five, and one clock tick per second as an example, a saving of 140,400 messages per hour is achieved from grounding this specification.

#### 5.4.3.3 Performance measures for JADE and Pastry

As described in Chapter 4, the situation determination middleware is based upon the JADE agent infrastructure [118] and uses the Pastry distributed hash table implementation to conduct large-scale, inter-environment searches for situation requests' indexes. This section briefly mentions some performance characteristics of these components, as they affect the overall performance of the situation determination middleware.

Within the situation determination middleware, an agent is used as the basic software representation for all computational components, including applications, sensors, and context filtering and transformation components, and the agent infrastructure manages all communication between them. If it is sought to deploy the situation determination middleware on a large scale, the performance of the agent substrate places a limit on the overall situation determination middleware. Fortunately, the agent discovery time, average messaging round trip time and memory consumption scale linearly with the number of agents hosted within the infrastructure [144]. The capability of the desktop PC host machines that were used in the evaluation, as described in Section 5.4.1, could comfortably host tens of thousands of agents.

To perform inter-environment situation determination, the index of a situation must first be located. As the number of environments connected in an inter-

environment network increases, the time spent searching these environments for a situation index could significantly delay the middleware's response time. However, using a distributed hash table to implement inter-environment search guarantees that only a small number of environments need to be contacted in order to locate a situation's index. When performing such a lookup, the FreePastry / PAST implementation, which is used by the situation determination middleware prototype, is known to require a maximum route length of $\lceil log_{16} N \rceil$, where $N$ is the number of nodes in the network [123, 145]. For example, in a network of one million environments, at most only five environments would have to be contacted to locate a situation index.

## 5.5 Summary

This chapter presented an evaluation of the pervasive situation determination middleware developed in this thesis, assessing the complexity, expressivity and performance of the prototype implementation that supports the distinct modelling and architectural requirements of pervasive situation determination.

In addressing both complexity and expressivity, it was first shown that the model and architecture were straightforward to instantiate and use, with graphical tools available to create and alter situation descriptions. Following this, that more than thirty rich situations spanning four different domains were created, and could be detected from a relatively small set of only thirteen software sensors based on person and device profile information, calendar information, device input, software application status, and location. Creating customisations was simple, and could be performed using a dedicated customisation creation tool. More than two hundred customisations were created for the situations, featuring thirty of them in the use of personal devices and applications, eighteen for public setting situations, and a further one hundred and seventy six customisations based on University work setting situations. Even with only a small amount of sensing infrastructure in use, it was possible to create a variety of situation specifications, with some situations having as many as five different variations, and a total of fifty three specifications in all, allowing the middleware to appropriately adapt to changes in the available sensing infrastructure.

In addition to this, a suite of three distinct situation-aware applications was developed, each of which were simple to construct. They included an availability checker application that can perform ad hoc situation queries of a person, device

or location, whether the target may be in the local or a distant environment. In addition, there was a mode manager application that continuously monitors and autonomously reacts to a user's situations, constantly adapting to changes in sensing infrastructure and environment. Also included was a situation-enhanced file search application that detects and records all of the user's situations and allows them to later search through these situations to retrieve documents and files associated with them. Together, they demonstrate the breadth of styles of application the middleware can support.

In looking at performance, a number of performance measurements and analyses were presented. These included round trip time and join time measurements from light through heavy loads, the reductions afforded by sharing the load between multiple hosts, and algorithmic analyses of key components. Through these, the pervasive situation determination middleware was shown to be suitable for use by interactive situation-aware applications, and that it scales sufficiently to realistic deployment sizes, for both infrastructure-based and ad hoc environments.

In summary, the evaluation has illustrated that the approach presented in this thesis satisfies the modelling and architectural requirements of pervasive situation determination with sufficient expressivity, affording straightforward instantiation of situation models, of the middleware itself and of situation-aware applications, as well as offering adequate performance across a variety of settings.

# Chapter 6

# Conclusions and Future Work

This chapter concludes the work presented in this thesis by giving a summary of both the ways in which it has fulfilled the goals it set out to achieve and the main contributions of the work. The chapter closes by outlining possible extensions to the approach and other directions of future work.

## 6.1 Summary and Conclusions

As identified in Chapter 1, situation determination has become a key requirement for infrastructures supporting pervasive computing. However, as demonstrated in Chapter 2, current approaches do not provide sufficient support to fully realise the benefits of pervasive situation determination. To provide this support has been the aim of the work presented in this thesis.

In Chapter 1, a number of modelling and architectural requirements were established that are essential in realising pervasive situation determination. The modelling requirements included supporting customised situations, rich situation models, alternative situation descriptions and multiple situation viewpoints. The architectural requirements stipulated support for adaptable recognition, resource management, inter-environment operation and situation discovery.

In what follows, each of the original modelling and architectural requirements are listed in turn, along with a brief summary of how the approach developed in this thesis fulfils that requirement. Proceeding this, a summary of the evaluation of the approach is given, detailing each of its aims and how these were satisfied.

### 6.1.1 Modelling requirements

In this section, each of the modelling requirements, accompanied by a brief summary of how it was met by the modelling approach, is presented in turn:

**Customised situations** - *In current approaches, the situations that are recognised are typically general, created externally, and do not capture the distinctive features of situations that are particular to the individual user or their environment. None offer the capability for end-users themselves to create their own customised situations. It should be possible that customisation can be performed by end-users themselves, as this provides greater cover of the situations that the user is interested in without relying on specialist maintenance to achieve it.*

The two critical elements in supporting customised situations were simplicity and scope. The customisation process had to be simple enough that end-users could perform it themselves, yet at the same time it had to be sufficiently comprehensive to cover the characteristic elements of the bespoke, local situations that the user seeks to capture.

This balance was addressed through the observation that many customised situations bespoke to a particular environment may not be entirely distinct from each other, but rather form environment-specific variations of a more general situation. The structure of these general situations and the information required by a specification of how to recognise them, are likely to be similar or follow a set of similar patterns. In addition, the variations are likely to differ in the details of the situations. The difficult part of correlating particular sensor data to a general situation can then be performed by a skilled administrator or an external source, and the result can be reused in multiple environments. The general situation can then be customised to recognise the bespoke variations that occur in a particular environment.

The customisation process itself has been made very simple. To create a customisation, the user first selects the general situation they wish to customise. As shown in Chapter 3, a customisation is based on the features of the situation, where a feature is an externally observable property that characterises some aspect of the situation. The features include the roles that are played within the situation, as well as the people, devices or locations that play them. Furthermore, they are described in high-level terms that are meaningful to the user and reflect the real-world entities they represent.

The user forms the customisation by selecting a number of simple constraints

upon a situation's features. All of the features of a situation are available to the user, allowing any aspect of the situation to be customised. Furthermore, as the customisations are based solely upon the features of a situation description, they may be utilised by any of the user's situation-aware applications in a given environment, without concern for which appropriate specifications are being used to recognise the base situation.

Section 5.2 demonstrated a convenient GUI tool that allowed the user to graphically construct a customisation from a mobile device. Moreover, Section 5.3.2 described how this was used to create a wealth of customisations across a range of domains, each satisfying the particular needs of the individual user and/or local environment.

In enabling the recognition of a greater number of situations like this, this approach presents the user with a wider set of opportunities to tailor the behaviour of the situation-aware applications they use, which may be essential for them to function in the way the user wishes.

**Rich situation models** - *Representations of a situation are commonly coarse-grained, and lack details or a visible internal structure that can be exploited by situation-aware applications. Greater detail and specific roles should be captured in the situation to allow the desired system responses and outcomes to be more precisely defined and more closely matched to the user's needs.*

Chapter 3 introduced a situation representation that includes three main parts. The first part is the situation description itself, which is the abstract description of the features a situation contains. The second part is a customisation, which details simple constraints on the features of a situation, which allow environment administrators and end-users to tailor the situation to their own environment and preferences. The third part involves situation specifications, which are more complex documents describing the entities, properties, locations, and constraints upon them that must hold for the situation to be occurring. A specification provides a mapping from the low-level sensor data available in the environment to the high-level features of a situation and is used to recognise that situation.

The features included in a situation's description may cover an extensive set of details about it. Both situations and the features within them may be modelled hierarchically and reasoned about abstractly. Moreover, declaring a comprehensive set of features for a situation has the advantage of providing a consistent interface to situation-aware applications, and that the situation determination

system may utilise the finest-grained reports of a situation that the current environment is capable of providing. All of the features of a situation may be directly accessed by a situation-aware application, allowing the application to react precisely to the details of the situation.

The approach provides two types of specification that are used to recognise a situation - a role specification and a situation specification. A role specification describes the individual role an entity plays in the situation, describing the set of constraints and relations that must hold for the particular entity to be playing the role. The situation specification then describes the situation as a whole, describing a composition of roles and a set of constraints and relations upon them, which must hold in order for the situation to be occurring in the environment. This two-tiered structure allows for a simple and natural description of a situation. In particular, it is easy to express constraints over the set of all of the entities playing a role and on the properties of the set itself, in addition to grouping together all of the entities that relate to the same instance of a situation. Such constraints, which were used extensively throughout the specifications developed in Chapter 5, are difficult to express using conventional logic-based approaches. Furthermore, the set of constructs available to the specification author is extensible, allowing new entities, properties and relations to be introduced as required.

The specifications make use of location types, allowing fine-grained types for a particular space to be defined. These offer greater convenience and generality in writing rich situation descriptions. Furthermore, a situation specification introduces the notion of an 'area of influence' of a situation, that not only allows the specification author to set a physical boundary around the location of the entities that may have an influence on the situation, but also permits a reduction the computational effort required to recognise a situation. In addition, the inclusion of the resource requirements metrics in a specification allows the system to optimise the placement of the recognition of the situation.

As the sensor data that is used to detect the elements of a situation may suffer problems of accuracy or staleness that affect the confidence the system can have in that data, the representation had to provide a convenient means of incorporating and handling measures of confidence. This work developed an approach in which the system automatically handles the calculation of the confidence of a situation as a whole, based on the confidence measures of the complex composition of context information that is used to recognise the situation. The process offers a fast and efficient means of calculation based on monotonic selection. More-

over, the only additional element required from the specification author is to state a minimum confidence threshold for any roles referenced in the situation's specification.

Combined, these offer a rich suite of modelling constructs for representing situations. Chapter 5 illustrated a number of applications that took advantage of being able to capture the structure and detail of a situation like this, and how it allowed the ways in which the applications reacted to a situation to be precisely defined.

**Alternative descriptions** - *A description of how to recognise a situation generally relies upon particular sensing infrastructure being available and so cannot be used in its absence. To be more pervasive, the model of a situation should include multiple, alternative descriptions that specify how the situation can be recognised from a variety of different forms of sensing infrastructure.*

A situation description is an abstract description, as it declares *what* is recognised for a given situation, but says nothing about *how* it is recognised. It is the mappings from low-level sensor data to high-level features provided by situation specifications that describe how a situation is recognised. The features of a situation description serve as a common interface to the situation for customisations, specifications and situation-aware applications. As customisations and the interaction with applications are based solely upon this common interface, the specification used to recognise the situation can be changed at will. This simple decoupling enables the number and choice of specifications used to recognise a situation to be dynamically adapted, and makes the recognition of a situation robust against changes in the environment as well as the available sensing infrastructure.

The level of independence that this achieves allows the model to provide simultaneous support for the two dimensions of variability encountered in a pervasive computing environment, which were introduced in Chapter 1. The first dimension concerns what is recognised, which runs down through the hierarchy of situations and customisations. The second concerns how these are recognised, and spans across the many different environments and sensing infrastructures that may be encountered. In the model, these concerns have been made orthogonal, which allows them to vary dynamically and independently.

**Multiple viewpoints** - *Typically, a situation focuses exclusively on the viewpoint of the single, local, current user of a situation-aware application. While doing so makes the system simpler, it also significantly limits how pervasive the*

*system can be. The system should be capable of reporting the situations of any person, device or location to any user in the system.*

A model of a situation is not anchored to any particular viewpoint. A situation provides a full representation of all the people and artefacts that are involved within it. It is only when a situation is requested and a specific index is selected, that a particular viewpoint is set. A request for a situation may be from the perspective of any person, device or location featured within a situation, and requires no changes in the representation of the situation, its customisations or its specifications.

In conclusion, the modelling approach presented in this thesis aptly fulfils the modelling requirements necessary for pervasive situation determination.

### 6.1.2 Architectural requirements

This section presents each of the architectural requirements, along with a brief summary of how it was fulfilled, in turn:

**Adaptable recognition** - *In addition to the model being able to provide alternative descriptions of a particular situation, the infrastructure should be able to provide adaptable recognition for the process as a whole, incorporating new descriptions and sensing infrastructure as they appear in the environment, as well as adapting to the loss of existing sources. Enabling the system to fully exploit the dynamic environment in which it operates can result in greater availability of both the system and users' situation-aware applications.*

In Chapter 4, it was shown how the architecture can automatically exploit any new data sources or sensing technologies introduced into the environment in the form of data context entity agents (DCEAs), as well as any new computational, pattern recognition or conversion capabilities in the form of compute context entity agents (CCEAs), and how the flow of information from these was co-ordinated and fused by the compute context entity manager (CCEAM). Also shown was how specification repository agents (SRAs) act as dynamic sources of new situations, specifications and customisations within an environment. The flexibility of the situation recognition process used by a situation determination agent (SDA) was demonstrated, including how each of these sources can be dynamically incorporated to provide recognition of the broadest range of situations at the highest granularity possible. The autonomous co-ordination of these agents' actions provides an extremely flexible, extensible and adaptable situation

recognition process that can continually adapt to an environment as it evolves.

In supporting adaptive recognition like this, the architecture affords greater availability. This is done by being able to recognise the situations to the highest granularity and confidence possible, despite changes in the available sensing infrastructure or as the user changes environment. Providing a greater level of availability like this creates a wider set of opportunities in which the situation-aware applications a user wishes to use can function.

**Resource management** - *Recognising a situation can be a computationally expensive task, yet it often must be performed by resource-constrained devices. To mitigate the cost of this, the system should strive to manage the resources within it, shifting expensive tasks to the devices that can best afford them.*

Chapter 4 demonstrated how the architecture employs several techniques to evenly distribute the load between agents of the system and to strive to preserve the battery life of mobile devices. These included resource requirements based assignment of situation specifications to SDAs by the situation determination agent manager (SDAM), as well as a resource foraging approach to allow the computationally expensive operation of recognising situation specifications to be off-loaded from resource-constrained to resource-rich hosts, and also the specification grounding technique that can be used to minimise communication costs for mobile SDA hosts. After noting that wireless communication is often the biggest drain on the battery life of a mobile device, Chapter 5 provided an example of how the grounding technique can considerably reduce the number of messages that such a device must send.

**Inter-environment operation** - *Supporting inter-environment operation is essential to providing the necessary reach and scope of situation-aware applications, when the situations a user is interested in extend beyond the local environment.*

In Chapter 4, it was shown that this requirement is supported by the architecture in two ways. The first was the ability of the index server agent (ISA) of an individual person, device or location to receive situation requests from and send situation reports to a situation-aware application (SAA) anywhere within the network. The second was by supporting both environment and ad hoc modes of operation and having an environment manager agent (EMA) autonomously manage transitions between the two, allowing the architecture to support continuous recognition of a person or device's situations over changes in location, network or environment, as well as changes in the architecture's operational mode.

**Situation discovery** - *When the situations a user is interested in do extend beyond the local environment, the system must have a means of discovering situations that are occurring elsewhere within the network of environments.*

In Chapter 4 it was shown that within the architecture, an index server agent (ISA) and an index locator agent (ILA) make innovative use of distributed hash table (DHT) techniques in co-operating to provide large-scale situation discovery. Furthermore, it was illustrated how this enables situation-aware applications to base their behaviour not only upon situations that are occurring in the local environment, but also upon those in external, potentially remote environments elsewhere in the world.

To conclude, the architecture created in this thesis fully satisfies the architectural requirements necessary for pervasive situation determination.

### 6.1.3 Evaluation

Chapter 5 presented an evaluation of a prototype middleware implementation that supports the distinct modelling and architectural requirements of pervasive situation determination. Presented below are the aims of this evaluation along with a short summary of how each was satisfied.

**Sufficient expressivity** - *Establish that the model and architecture are sufficiently expressive to conveniently describe several situations, specifications and customisations spanning a range of domains and environments, and support a variety of styles of pervasive situation-aware applications, respectively.*

Section 5.3.2 demonstrated the expressiveness of the modelling approach by presenting a wide variety of situations that could be developed. A total of thirty-two situations were created, spanning four different domains. These were detected from a small set of thirteen software sensors based on person and device profile information, calendar information, device input, software application status and location. More than two hundred customisations were created for the situations, featuring thirty of them in the use of personal devices and applications, eighteen for public setting situations, and a further one hundred and seventy six customisations based on University work setting situations. Even with only a small amount of sensing infrastructure in use, it was possible to create a variety of situation specifications, with some situations having as many as five different variations, and a total of fifty-three specifications in all.

The support the architecture offers to pervasive situation-aware applications

was demonstrated in Section 5.3.3 through the breadth of styles of application that could be developed using the prototype. To this end, a suite of three distinct situation-aware applications was constructed. The first was an availability checker that can perform ad hoc situation queries of a person, device or location, whether the target may be in the local or distant environment. The second was a mode manager application that continuously monitors and autonomously reacts to a user's situations, constantly adapting to changes in sensing infrastructure and environment. The third was a situation-enhanced file search application that detects and records all of the user's situations and allows them to later search through these situations to retrieve documents and files associated with them. Furthermore, it was shown how modelling, processing and sensing infrastructure elements could easily be extended to accommodate the specific needs of a new application.

**Straightforward instantiation** - *Demonstrate that situation models, the architecture and pervasive situation-aware applications afford straightforward instantiation.*

In Sections 5.2 and 5.3, the simplicity of the modelling approach was illustrated by the ease with which situations could be constructed. Situation descriptions, specifications and customisations are represented as Web Ontology Language (OWL) [128] documents. OWL allows rich, extensible, semantic models to be constructed to represent the environment, the entities within them as well as the situations that occur. Broad tool support is available for editing OWL documents, which makes it easy to construct the situation descriptions. Graphical tools exist that allow the author to select the desired classes from an ontology to represent the people and artefacts involved in a situation, and to specify the appropriate relations and constraints. Furthermore, the validity and consistency of situation descriptions can be automatically checked.

The middleware prototype that was developed and discussed in Section 5.2, illustrated that instantiation of the architecture was possible. Furthermore, it was shown that the development of the prototype was simplified through the use of appropriate frameworks to implement the more standard elements of the architecture. These include the agent substrate that was used, the Rete network implementation used in recognising a situation, the ad hoc discovery mechanisms, as well as the distributed hash table implementation used in inter-environment operation and situation discovery. This reuse allowed the effort involved in producing the prototype to focus on the original parts of the architecture.

Section 5.3 established that the situation-aware applications that were developed were simple to construct. It was shown that an application developer need only include the situation-aware application agent (SAA) in their code to incorporate the pervasive situation determination middleware. Furthermore, the SAA can manage all of the situation reports requested and received by the application.

**Practical performance** - *Determine that the middleware provides a sufficient level of performance for the number and size of situations that would be encountered in real-world settings.*

The performance of the middleware prototype was assessed in Section 5.4, which presented a number of performance measurements and analyses.

The first of these measured situation request round trip times and join times. Several measurements were recorded as both the size of a situation was scaled from small to large instances, and the overall load increased from a few to many situations and specifications being recognised at the same time. These measurements were repeated for both infrastructure-based and ad hoc modes of operation. The analysis of the infrastructure-based mode additionally included measurements demonstrating the significant reductions in round trip times afforded by sharing the recognition load between multiple SDA hosts. In each case, the prototype middleware implementation was shown to be suitable for use by interactive situation-aware applications, and that it scales sufficiently to realistic deployment sizes.

Given this evaluation, it can be concluded the approach provides sufficient expressivity in both the model, to describe a wide variety of situations, specifications and customisations, and in the architecture, to support many styles of pervasive situation-aware application. Furthermore, that situation models, the architecture and applications all afford straightforward instantiation, and that the middleware provides a sufficient level of performance for interactive use in real-world settings.

### 6.1.4 Contributions

In conclusion, the novel aspects of the approach developed in this thesis combine to achieve the goal originally set out for this work: to effectively meet the requirements, and realise the benefits of, pervasive situation determination. This has been achieved through the following main contributions of this work:

- The presentation of an original model that provides support for the require-

ments concerning customised situations, rich situation models, alternative descriptions and multiple viewpoints. These are not addressed by current approaches.

- The development of a novel software architecture that fully supports the model while additionally addressing the architectural requirements of adaptable recognition, resource management, inter-environment operation and situation discovery. These are currently lacking in existing approaches.

- The evaluation of both the model and architecture through the development of an extensive number of situations, specifications, and customisations, spanning a range of domains and environments, and a suite of situation-aware applications.

## 6.2 Future Work

Branching from the work presented here, there are several further, exciting avenues that could be explored. This section examines some of these avenues concerning the model, architecture and evaluation of the situation determination middleware and situation-aware systems.

### 6.2.1 Model

In the current modelling approach, there is an implicit monotonicity constraint between a customisation and a specification in the case where, in order to recognise the situation, the specification must place constraints upon the members of the resulting feature set. In this case, the constraints a customisation places upon the feature set must be more restrictive than those of the specification, in order for the customisation to be matched. This was never a problem in practice for any of the situations, specifications or customisations developed in Chapter 5. There was a natural tendency for customisations to act as a refinement, and for specifications and customisations to focus on different properties. Typically, specifications would focus on the structure of the situation, while customisations would focus on the details of particular instances.

Lifting this constraint frees an end-user customisation from being a refinement, and allows it to act as a more general transformation. In order to do this, an *interpreter* could be associated and distributed with a specification. Then, given

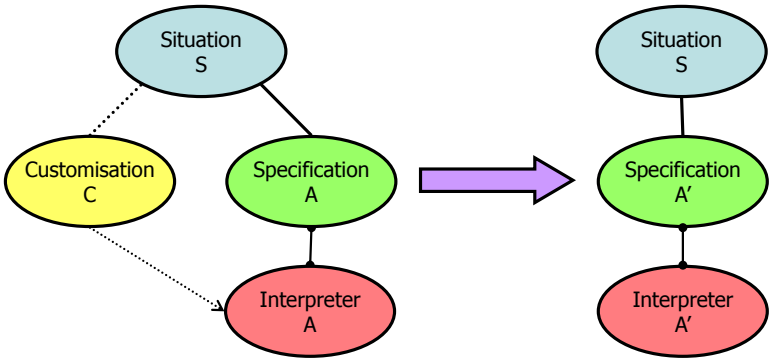



Figure 6.1: In this figure, specification A describes situation S. To apply customisation C, its interpreter takes customisation C, merges it with specification A to produce specification A′, which also describes situation S, and its associated interpreter A′.

a particular customisation, the interpreter could generate an altered specification that combines the constraints from the specification and the customisation, and resolves any conflicts as necessary. In order for further customisations to be applied to the resulting altered specification, an updated interpreter would also need to be produced, that was associated with the newly generated specification. This process is illustrated in Figure 6.1.

An interesting avenue of investigation would be to explore the computational efficiency of such a mechanism. To construct it, suitable combination operators would need to be defined, along with conflict resolution policies for each property type, in addition to the interpreter generation language and mechanism itself. The increase in the computational load required to recognise a customisation, possibly incurred at runtime, would include the cost of analysing and verifying the original specification and customisation, and then generating the new specification via the interpreter, for each customisation in the series. Also, the process would produce a larger number of specifications that must be recognised simultaneously, and for each specification, its associated interpreter must also be stored and distributed with it. The advantage of this enhancement would be that it could offer greater freedom of expression by potentially allowing a series of customisations to be applied to a specification without restraint. The challenge would be to offer full generality in the customisations, while limiting the additional overheads it incurs.

In this work, the focus has been on recognising situations that are currently occurring within a short temporal extent. This allowed a situation request to be

received, either in the local or a remote environment, processed, recognised, and the report posted back, all quickly enough to enable situation-aware applications such as the availability checker application from Chapter 5, to perform dynamic, interactive situation queries.

However, for other applications, the reports of which situations are occurring may not be used until later. For example, the situation-enhanced file search application from Chapter 5 logged situation occurrences that were later matched by search criteria for finding particular files. Situation-aware applications such as this could take advantage of situations that were modelled using explicit temporal dependencies, and may even require retrospective reasoning to be accurately identified. This would also allow richer situation queries to be performed, for example an application may search for 'the presentation in which John asked a question just after Beth'.

Another interesting avenue that could be explored is the possibility of learning customisations for a given environment. For example, the middleware could monitor the situations that occur in an environment over time, and where regularities exist in the values of a situation's feature set, the system could present this as a possible customisation. This would allow the middleware to automatically suggest customisations to the user, in addition to them being able to create customisations themselves.

In addition to this, allowing the users to give feedback to the pervasive situation determination middleware would allow the recognition of situations to be adapted to the local environment in a principled way, similar to that attempted by Ye et al. [75].

There are several ways that such feedback could be incorporated into the system. For example, given a set of situations that are currently reported as occurring, a user could identify any misreported situations and the system could appropriately adjust the confidence thresholds for these situations.

The user may also wish to adapt confidence thresholds for pairs of situation descriptions and application behaviours, in order to tailor the confidence thresholds to particular uses of an application. For example, behaviours for which the user considers the consequences of mis-enactment are low risk, such as a setting a screen saver on or off, may be set to use a lower confidence threshold, whereas higher risk behaviours would require a higher threshold.

Furthermore, feedback could be linked in to the interpreter discussed above, such that it may be possible that more general corrections could be performed by

the interpreter. That is, if the user has identified a number of cases of misreported situations, these could be analysed by the interpreter along with the situations' original specifications and identify not only where confidence thresholds could be adjusted, but also constraints and expressions that could be adapted to better match the expectation of the users in the local environment. Complementary to learning customisations, which provides the opportunity to automatically learn new, custom situations that are local to the environment, enabling user-driven corrections such as these would allow local adaptation of the base situations' specifications.

Currently, there is no way to define a situation or customisation as a composition of other situations. For example, a user may wish to define a particular situation as when they are 'travelling to work' and 'listening to music'. Furthermore, this could be useful for dealing with situations on a larger scale. For example, within a building, given the situations that are occurring within each of its rooms, the situation of the building itself could be defined as 'normal' or 'not normal'. It is possible that such situations could also be learnt over time.

### 6.2.2 Architecture

Privacy will be an important aspect of real-world deployments of pervasive computing systems. The pervasive situation determination approach presented in this thesis would have an impact on this issue. On one hand, being able to query the situations of a friend or colleague in whichever environment they currently happen to be in, significantly extends the scope and utility of situation-aware applications. On the other hand, unchecked use of such applications could pose a threat to the privacy of their users. Addressing the privacy issues associated with pervasive situation determination would form the priority development of future work, and so is given an extended overview here.

Privacy issues affect not only situation determination systems, but also context-aware and pervasive computing systems in general. Langheinrich provides a comprehensive privacy framework for pervasive computing systems that highlights six privacy principles [146, 147]. These principles provide a suitable starting point upon which privacy related extensions to the pervasive situation determination approach could be based. The six principles can be summarised as follows:

**Notice** - Any subject that has data collected about them, must be aware of what and how data is being collected.

**Choice and consent** - A subject must first give their explicit consent before any data is collected about them, and be able to choose the specific pieces of information that are collected.

**Anonymity and pseudonymity** - Anonymity can provide an alternative to personal data collection, in which data that is collected is not linked in any way to the identity of an individual. Pseudonymity schemes allow an individual to use a temporary id as an alias for a certain length of time.

**Proximity and locality** - An example of this principle would be that if someone's personal device was recording data about the people in the same room, it would only record data when the owner of the device was present in the room, and it would not propagate the data beyond the room. Such protection may be desirable in cases where clear notices, explicit consent or untraceable pseudonymity prove too difficult to implement.

**Adequate security** - All personal data access should be authenticated, and all transmission and storage of personal data should be encrypted.

**Access and recourse** - A subject can access the personal data a body holds about them, check the purpose for which the data is stored, and that it is retained only as long as is required for that purpose.

The main aim of these principles is to ensure that anybody that collects personal information can be made accountable for protecting its privacy. Ideally, a privacy-aware pervasive computing system would provide facilities to uphold all of these principles where appropriate.

An interesting area of future work would be to explore means of supporting these privacy principles within the pervasive situation determination architecture. Discussed below are the areas of the system that each of these principles affect, and some initial ways in which support for them could be provided.

Providing sufficient notice requires that every person that is located within the area of influence of any situation specification that is currently being recognised within the environment, must be notified of the data that are required by all of the specifications and any customisations. Furthermore, they must also be notified of the situations that the data are used to recognise, as well as who has requested the situations and for what purpose they are being requested.

Within the pervasive situation determination system, it is assumed that users carry a personal device with them, such as a mobile phone, that hosts their representative index server agent (ISA). It is possible that the situation determination agents (SDAs) operating within the environment could deliver notice to the ISAs

of each of the people whose personal data is currently referenced by a specification that is currently being processed in the SDA's active set. The personal device that hosts the ISA could then indicate that the user's personal data is currently being used in recognising situations, perhaps by displaying a particular symbol in the corner of the screen. Then, the user could click on the symbol to show the full details of the notice.

Choice and consent would be challenging to implement as it demands feedback from each person involved in a situation, for potentially every request. When a situation request is received by an ISA, each of the people whose personal data may be referenced by a specification that would be used to recognise any of the situations included in the request, would have to be prompted. Each of the people would then have to review the request, select how their personal data may be used for this request, and then grant their consent. The user could choose to be completely excluded, effectively making them invisible to the request, they may choose to appear anonymously or as their pseudonym, or they may choose to only permit certain pieces of information to be used. It could even be possible to choose to have the situation obfuscated in some way. For example, a user's index server agent (ISA) may intentionally blur or otherwise modify a situation report, such as it reveals only that the user is currently involved in a meeting, but not who the user is in a meeting with.

However, continued interruption from such prompting would quickly prove frustrating. A way to address this would be to provide user policy files that specify consent in advance for known situations, purposes and request issuers. Consent could then be granted automatically by the user's ISA. Furthermore, a user's privacy policy may address information that is produced about the user but is not under the user's direct control. A policy may contain stipulations such as that any location data about the user that is generated by the host environment itself (for example, by the video-based location system mentioned in Chapter 4) must be transmitted securely and only to their ISA, or that the identity of the user must be removed from any situation report that the user is involved in but is not the index. The challenge here is to provide a means for end-users to create and alter such policies that can be both sufficiently comprehensive and easy to use. Ideally, the user would be able to change their policy dynamically, such as to block current undesired usage of their personal data that is reported in the notice.

Attempting to support this principle further highlights the benefit of the alter-

native specification and adaptable recognition capabilities offered by the pervasive situation determination system. It may be possible to define alternative specifications not only for different sensing infrastructures, but also for expected privacy policy combinations. For example, one specification may assume all of the people involved in the situation granted access to the full set of required properties. Another specification may assume that all but the index will have granted only anonymous collection. In this way, recognition of situations could then also be made robust against a shifting sea of changing privacy policies.

Proximity and locality would form part of the stipulations of a user's privacy policy file. Proximity is implied for the index of a situation, as the area of influence of a situation specification is related to the current location of the index. However, it may be necessary to enforce proximity for the person or application that issues a situation request. In this case, a user's ISA may only grant consent if the issuer is at the same location. This could be made even more stringent by requiring the issuer to be part of the same situation.

Locality constraints influence which intended uses of a situation report are permissible, and so would affect the type of applications that could make use of them. For example, situation-aware applications that make immediate use of reports of which situations are currently occurring locally, such as the mode manager application presented in Chapter 5, would be able to make use of situation reports that are bound by locality agreements. As such reports would be received, checked and then discarded, the lifecycle of their use will be completely contained within the locality. Other applications that may intend to keep a history of situations, such as the situation-enhanced file search application also presented in Chapter 5, would not be able to make use of such reports, as their later use in searching through the history of situations may occur outwith the current locality.

Authentication of a situation request could be performed by the user's ISA. For example, a public key cryptography scheme could be used to verify the claimed identity of the issuer of the request. The ISA also provides an appropriate point at which to manage encryption, not only of the situation reports, but also of the individual pieces of context information pertaining to a particular user. The challenge here would be to retain the flexibility afforded by the openness of the publish / subscribe style of communication used to transmit a user's information within an environment, while being able to appropriately secure the information for each different type or instance of agent that must receive it, without incurring

inflated overheads in communication.

Access and recourse to situations recognised in an infrastructure-based environment could be supported by the organisation associated with the environment. A public, authenticated point of access could be provided, a web page for example, where a person that was involved in a situation recognised within the environment could access the data that is stored about them, and verify the purpose and intended duration for which the data is retained.

Supporting access and recourse for the ad hoc mode of operation may be more problematic. When operating in ad hoc mode, the onus will commonly be on the individual user, rather than a public organisation, to maintain a record of which situations were recognised and which data were used. This is a burden that many users would not wish to bear, particularly as a user may encounter many other people, for example in a busy train station or airport. In Chapter 4 it was suggested that a user may have a service provider which provides a fixed access point to the network of distributed hash table (DHT) nodes used for inter-environment operation and situation discovery. It may be possible that such a service provider could also be used to provide a public, authenticated access point for the personal information held by an individual user.

Another area that could be explored concerning the architecture, lies in further developing the CPU Load resource requirements measure presented in Chapter 4 to use platform-independent CPU predictors that can be used across processor architectures. Further exploration of techniques to help preserve the battery life of mobile devices used within the situation determination middleware, such as the specification grounding technique also presented in Chapter 4, that operate at a system-wide level is also possible.

### 6.2.3 Evaluation

Concerning the evaluation of the situation determination middleware, there are several further interesting studies that could be performed. These include further deployment in different environments and domains, exploring a larger variety of sensing infrastructure, situations and applications. Also, a thorough evaluation of the interaction between end-users and the situation determination middleware could be conducted, which explores the use of tools for creating customisations and managing their deployment, as well as the use of situation-aware applications in real environments. Additionally, there is the possibility of examining the novel

application opportunities that would be created by building up a large record of situation histories, which might include social networking applications, automatic diary applications, and stress and health monitoring applications.

Finally, making context and situation data sets publicly available from multiple environments, in similar spirit to the MIT House_n project [74], would help foster further research into the design and evaluation of situation determination techniques, inter-environment operation, sensing infrastructure and pervasive environments themselves.

# Appendix A

# Developing Location Awareness

This appendix presents how the location model was implemented and the development of the location detection system that were used by the situation determination middleware and situation-aware applications presented in Chapter 5.

To be appropriate for use by the pervasive situation determination middleware prototype, there were a number of requirements that a potential location system had to meet. First of all, there were several different types of target settings in which the location system would be used. These included a domestic setting, a University setting and public settings such as train stations, bus stops, shops and restaurants. Many of the situations that were developed for these settings had an Area of Influence (AoI) defined as a room. Therefore, the location system must have been able to support at least room level granularity. Furthermore, the location system was intended to be hosted on a user's personal device, so a system that could be run on a range of personal devices, including laptops, PDAs and mobile phones was sought. Finally, the software and data necessary to run the location system had to be freely available.

The open-source Placelab location system [33] is a software framework that can approximate the GPS co-ordinates of a host computing device based on 802.11 base station signal strength readings. The Placelab framework initially appeared to be a suitable choice for use in the middleware prototype due to a number of factors. It was based on 802.11 wireless networks, which were present in all of the target settings and available on each of the personal, mobile devices intended to host the system. It supported several host device platforms, including Windows XP-based laptop computers, and Windows Mobile-based PDAs and mobile

phones. It was also the only high quality software location system that was freely available.

The requirement the Placelab framework did not meet was in providing indoor, room level granularity. The level of accuracy that Placelab offered was sufficient to identify outdoor regions and create symbolic locations for public places. Sufficient accuracy could not be obtained however, to reliably identify distinct rooms within the domestic and University settings.

Extending the Placelab framework to be able to support room level granularity gave a means to produce a location system that could provide sufficient indoor accuracy that would enable testing of the pervasive situation determination middleware prototype in all of the target test environments, quickly and with reasonable effort. The development of this extension is presented in this Appendix.

The following sections look at how the location models and the location detection system extension were implemented. Section A.1 describes some of the test environments that were used as well as the location models that were constructed for them. Section A.2 presents each of the stages of development in implementing the location detection system extension. Finally, Section A.3 concludes this Appendix.

## A.1 Environments and location models

The set of environments encountered in the situations developed in Chapter 5 included a home and a University setting. It was these settings for which the most complex location models were created, and so make the most interesting examples for further detailed presentation here. Figures A.1 and A.2 show floor plans from the home and University test environments, respectively.

In the home environment, both bedrooms and the lounge area were used, and each room size was approximately 29m$^2$, 10m$^2$, and 8m$^2$, respectively. In the University environment, a mix of office areas and larger, open areas were used. The smaller office areas such as 12.05, were approximately 18m$^2$ in size, the larger office areas such as 12.06, were approximately 35m$^2$ in size, while the larger open areas such as 12.01, were approximately 94m$^2$. The University environment has a total of four floors, each with a similar layout to the floor shown in Figure A.2.

In order to implement location detection for a particular environment, a location model must be created for it. These are constructed by creating and

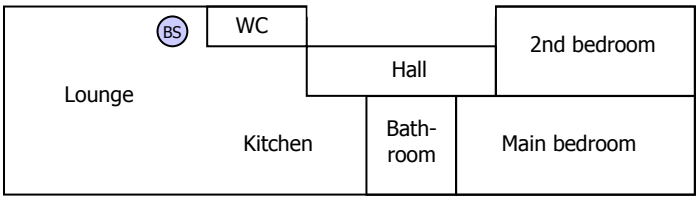



Figure A.1: A floor plan of the home test bed environment. The bubble with 'BS' inside it marks the location of the base station.

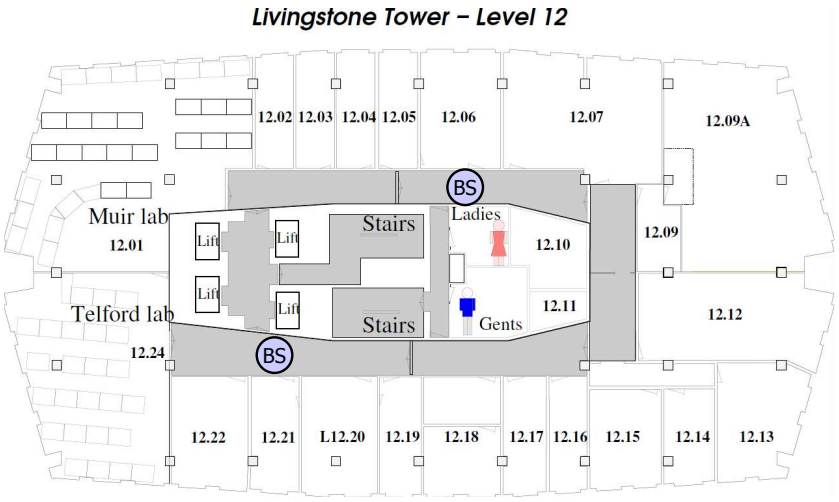



Figure A.2: A floor plan from the University test bed environment. The bubbles with 'BS' inside them mark the location of the base stations.

connecting instances of a location class, which is provided by the middleware ontology. Figure A.3 shows the structure of this location class and its properties. It states that a location may have a symbolic name, a location type, a parent location and a child location. There are no cardinality constraints on these properties, meaning that a given location may have any number of names, types, or parent or child locations. Using this simple structure, it is possible to build large, rich location graphs for an environment.

Figure A.4 shows some example locations from the location model created for the University test bed environment. It states that room "L13.18" is a presen-

```
<!-- Location. -->

<owl:Class rdf:ID="Location" />

<owl:DatatypeProperty rdf:ID="hasSymbolicName">
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasLocationType">
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasParentLocation">
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="#Location" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasChildLocation">
  <owl:inverseOf rdf:resource="#hasParentLocation" />
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="#Location" />
</owl:ObjectProperty>
```

Figure A.3: The structure of the location class as defined in the middleware ontology.

```
<!-- University locations. -->

<Location rdf:ID="LivingstoneTower13thFloor">
  <hasSymbolicName>L13</hasSymbolicName>
  <hasLocationType>Floor</hasLocationType>
  <hasParentLocation rdf:resource="#CISDepartment" />
</Location>

<Location rdf:ID="L1318">
  <hasSymbolicName>L13.18</hasSymbolicName>
  <hasLocationType>Room</hasLocationType>
  <hasLocationType>PresentationArea</hasLocationType>
  <hasParentLocation rdf:resource="#LivingstoneTower13thFloor" />
</Location>

<Location rdf:ID="L1318SpeakerArea">
  <hasSymbolicName>L13.18 Speaker Area</hasSymbolicName>
  <hasLocationType>SpeakerArea</hasLocationType>
  <hasParentLocation rdf:resource="#L1318" />
</Location>

<Location rdf:ID="L1318AudienceArea">
  <hasSymbolicName>L13.18 Audience Area</hasSymbolicName>
  <hasLocationType>AudienceArea</hasLocationType>
  <hasParentLocation rdf:resource="#L1318" />
</Location>
```

Figure A.4: Declarations of some example locations from the University test bed environment.

tation area, and contains two other areas - a speaker area and an audience area. L13.18 itself is part of floor L13, which is within the CIS Department.

The models for both the University and home environment used the common symbolic names for each location that already existed in the environment to give a familiar feel. Each model is defined within its own namespace to avoid name collisions of the symbolic co-ordinates.

## A.2 Implementing location detection

Several phases were involved in implementing the extension to the Placelab location detection system. These included first creating a sufficient signal detection capability, creating mappings from the received signals to known locations, improving confidence in the reported locations through exploiting the structure of the location model, and finally combining all of these aspects to form the location detection application itself. Each of these phases is described in turn throughout the following sections.

### A.2.1 Developing sufficient signal detection capability

When working indoors, there are several factors that can affect the accuracy of 802.11 signal strength detection, particularly interference from metal objects such as filing cabinets and metal girder structures in a building's support, as well as interference from moisture in the air and people. For a detailed account, please refer to [34].

Furthermore, the accuracy of the Placelab framework is affected by the wealth of stationary, infrastructure-mode base stations available in the environment [33]. The home test bed environment had a single infrastructure-mode base station within it, its location is marked on Figure A.1, and another eight to ten visible infrastructure-mode base stations externally located. The University test bed environment had a total of twelve infrastructure-mode base stations positioned throughout the environment. See Figure A.2.

Some initial experimentation was performed with an unmodified version of the Placelab framework. In both the home and University test environments, the detection granularity was low. At best, the application could detect different areas of approximately $100m^2$. In the University environment, this translated to being able to detect which floor the host device was on, and in which quarter of the building. In the home environment, it could effectively detect whether the host device was located within the home or not. Many of the situations that were to be detected required at least room level granularity. This made the performance of the unmodified Placelab software insufficient.

While the inferred GPS output of the Placelab software did not provide the required granularity, the software still provided an effective library for reading 802.11 base station signal strengths. Based upon this, a successful attempt was made at constructing room level location detection. Moreover, it was also possible

to detect finer-grained areas within larger rooms, such as the speaker and audience areas within a presentation area. The development of this application involved several phases - recording signal strength data from multiple points within each environment, analysis of these data, construction of a suitable location-detection algorithm and software, and testing in both the home and University environments. Each phase is discussed below in turn.

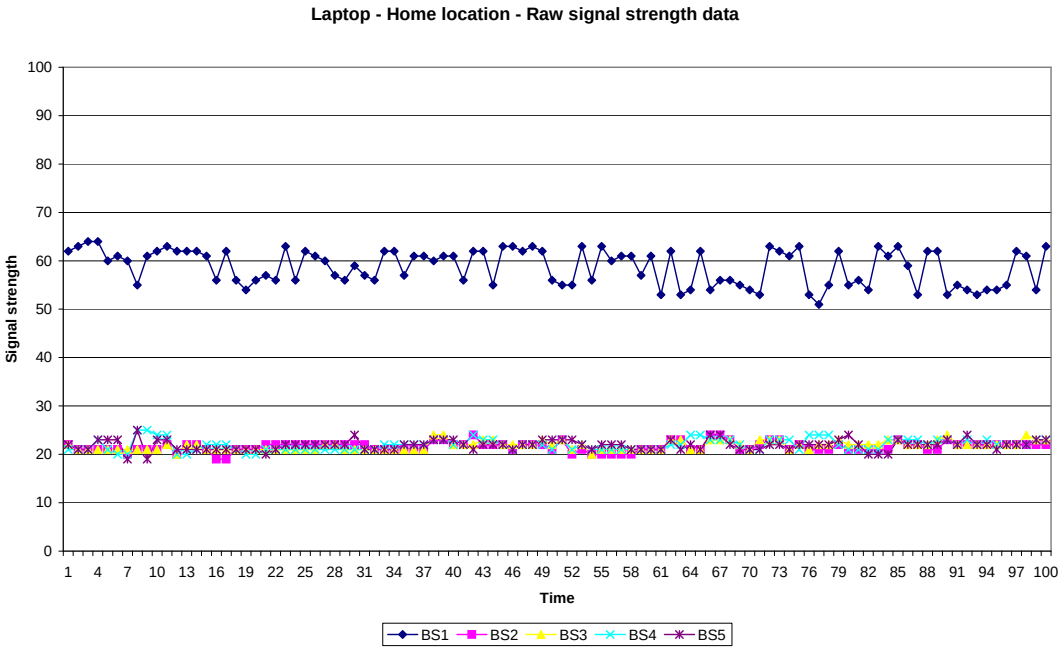
Figures A.5 and A.6 give examples of raw signal strength data collected using the Placelab software on both a Dell Inspiron 6000 laptop using an Intel PRO/Wireless 2200BG wireless network card, and an Hewlett Packard h5550 Pocket PC using the standard, built-in wireless network card. The specific locations are the lounge area of the home environment, and "12.01 Muir Lab", as shown in Figures A.1 and A.2, respectively. The values shown in the graphs are a normalised signal strength measure ranging from 100 (strongest) to 0 (weakest). In both examples, the host device was placed in a central position within the location, and was stationary for the length of the sample period.

The first observation about these data is that the signal was very noisy, particularly when using the Pocket PC. The reported signal strength oscillated rapidly between a range of values. This can be seen for the two specific locations in Figures A.5 and A.6.

More generally, when using the laptop, the upper and lower bound of this range shifted according to the distance from a base station. When close to a base station, the signal strength ranges between approximately 60 to 70, and when furthest from a base station, between approximately 10 to 20. When using the Pocket PC, the lower bound of the range would always be between approximately 10 to 30. The upper bound would range from 60 to 80 when close to a base station, and 10 to 30 when furthest away. This made the Pocket PC much noisier, and therefore more problematic to use to accurately detect location.

To counteract this noise for both the laptop and Pocket PC, filtering techniques had to be applied to the signal strength data in order to produce a steadier, consistent signal. In what follows however, focus is given to the noisier Pocket PC signal, though it was the same filtering techniques that were used in the final version of the location detector application for both the Pocket PC and laptop platforms.

Analysing the data from all locations within each environment, showed that low-strength signals (ranging between 0 to 25) occurred frequently in all location samples. These were then considered to be noise, and all values reported at 25



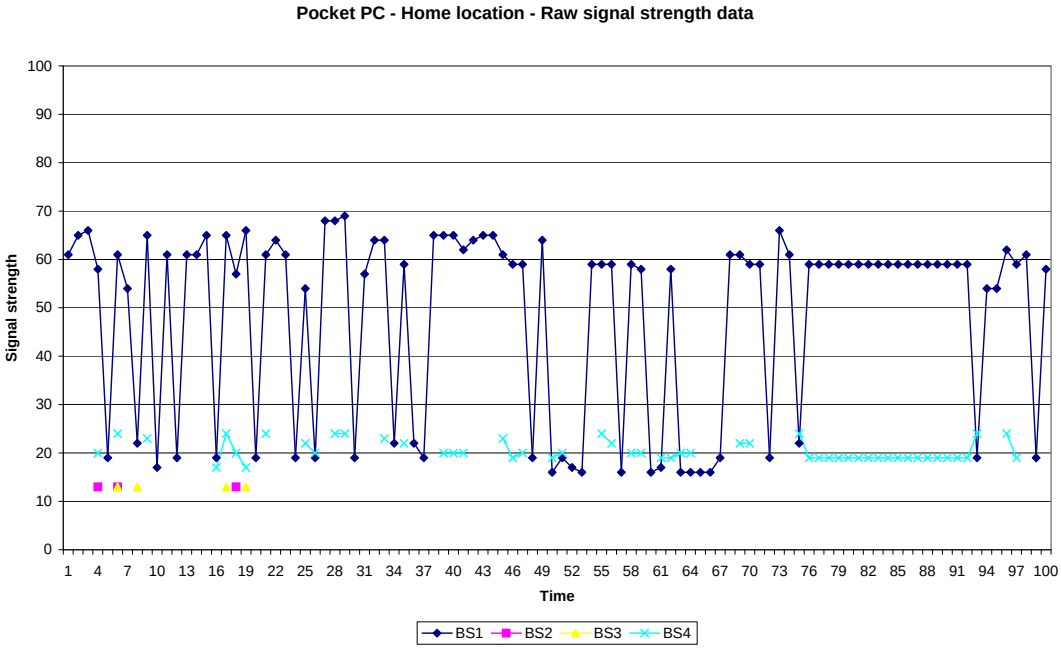




Figure A.5: These graphs show the raw 802.11 signal strength streams received by the laptop and Pocket PC devices for the home location. BS$n$ refers to the BSSID of particular base stations within each environment.

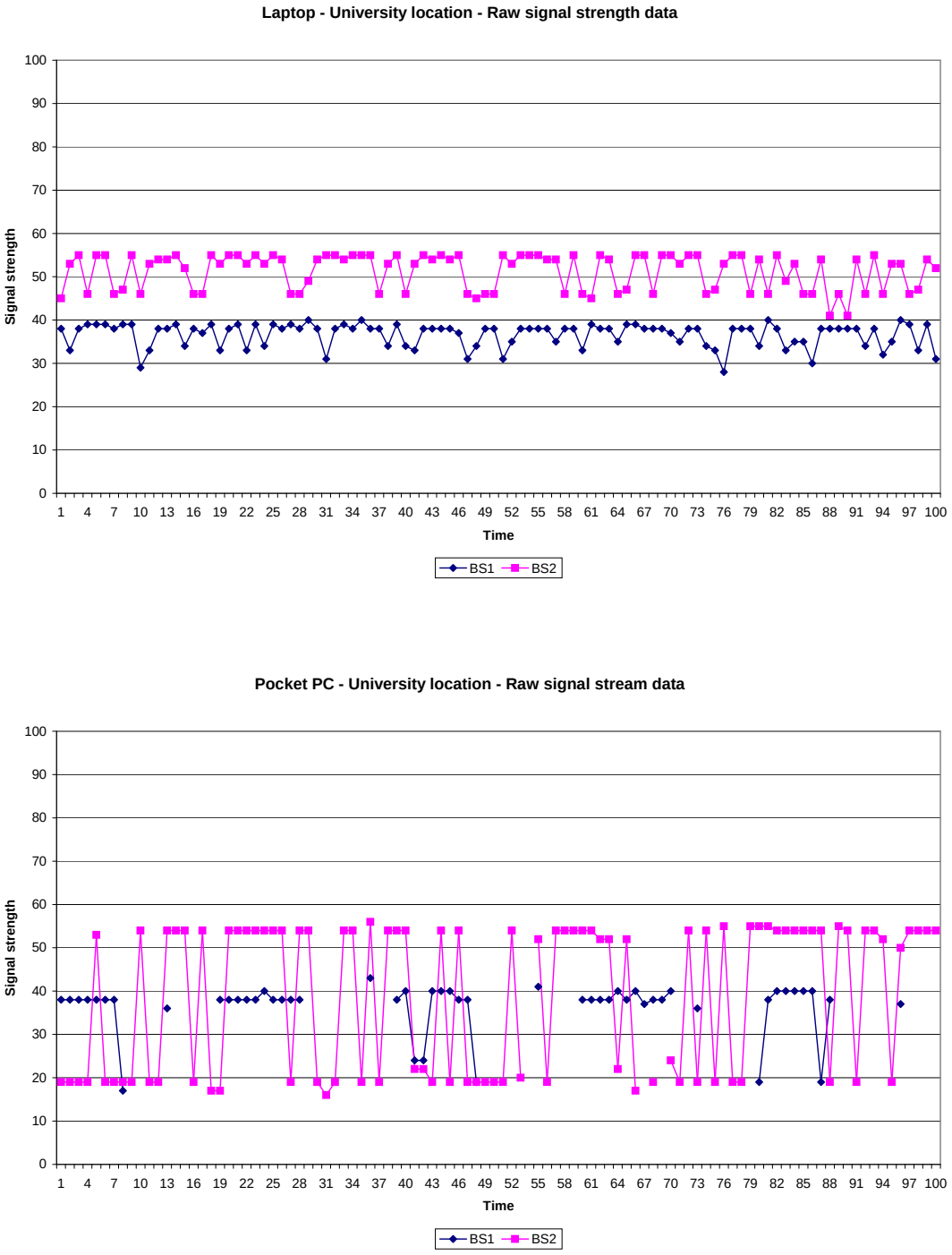
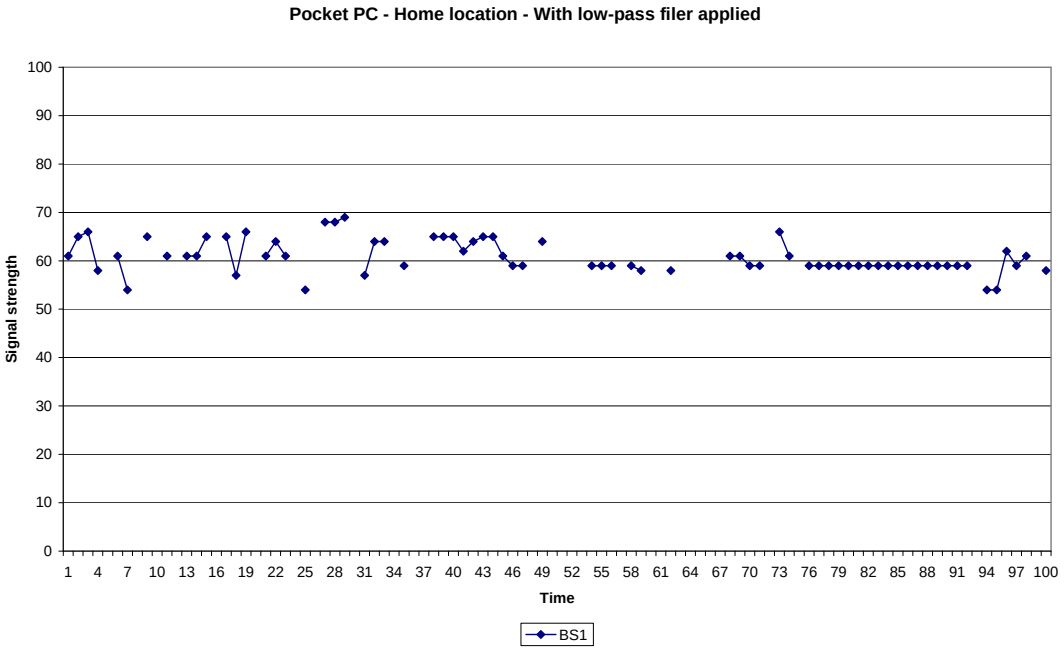



Figure A.6: These graphs show the raw 802.11 signal strength streams received by the laptop and Pocket PC devices for the University location. BS$n$ refers to the BSSID of particular base stations within each environment.



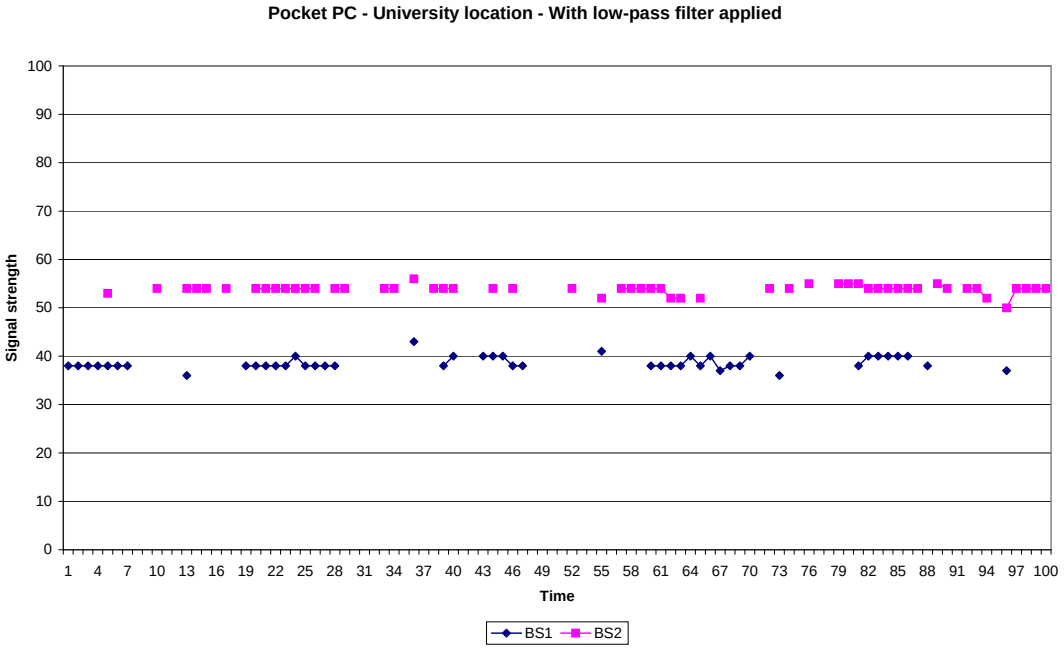
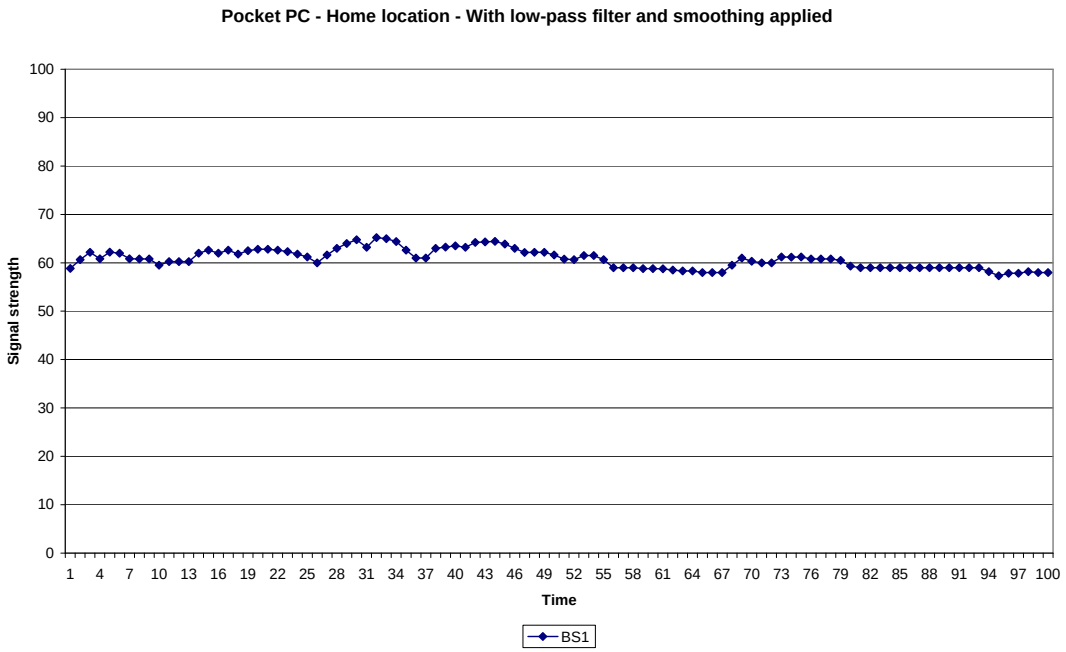




Figure A.7: These graphs show the resulting signal streams after applying a low-pass filter to the raw 802.11 signal strength streams shown in Figures A.5 and A.6.



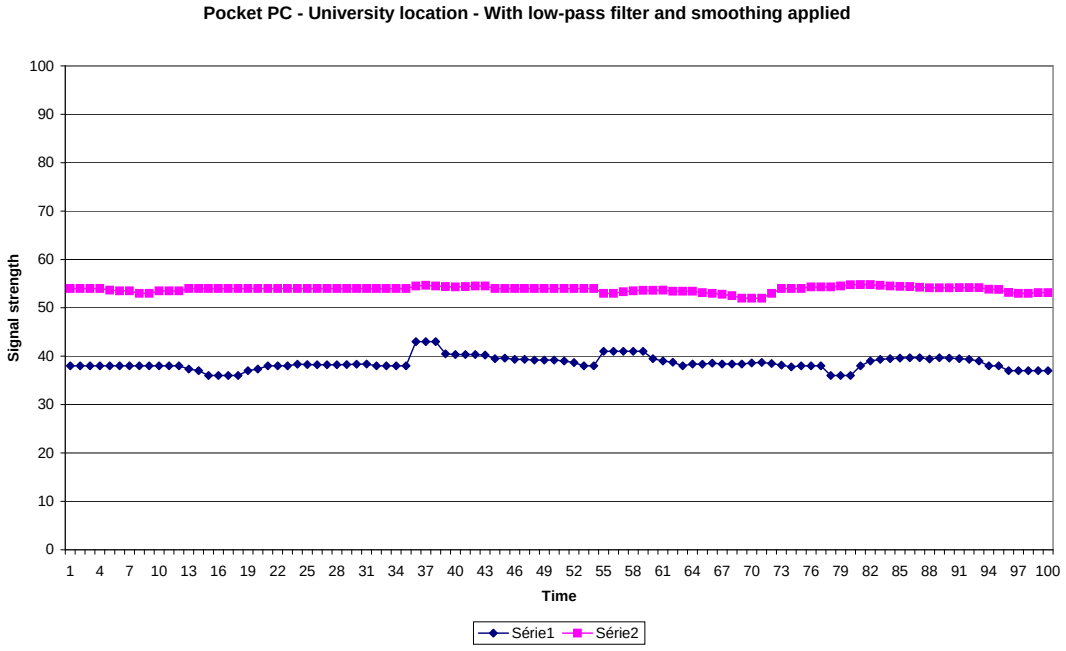




Figure A.8: These graphs show the resulting signal streams after applying a low-pass filter and smoothing to the raw 802.11 signal strength streams shown in Figures A.5 and A.6.

| Location | Signature |
|---|---|
| Lounge | $55 <= BS1 < 67$ |
| 12.01 Muir Lab | $50 <= BS1 < 57 \wedge 34 <= BS2 < 45$ |

Table A.1: Location signatures for the example "Lounge" and room "12.01 Muir Lab" locations from the home and University environments. BS$n$ refers to the BSSID of particular base stations within each environment.

or below were filtered out. Figure A.7 shows the same readings for the home and University locations shown in Figures A.5 and A.6, but with the low-value cut-off applied. While this removes a lot of noise from the signal, it can be seen that it still exhibits a lot of variability, and that the stream is frequently interrupted.

In a second stage of filtering, a smoothing technique was applied that produces the average value of a number of readings from the signal's recent history. When applying this technique, there was a tension between smoothness versus change detections. The Placelab software emits a signal strength reading for all visible base stations each second. A smoother signal can be created by increasing the length of the history vector, but this also increases the number of seconds until a change in location can be detected.

Having performed an analysis for all locations in each test environment, eight was shown to be a good common choice for the length of the history vector, as the stability of the signal did not improve significantly for vector lengths greater than eight. Figure A.8 shows the effect of applying smoothing to the signal strength data shown previously in Figure A.7.

### A.2.2 Creating signal to location mappings

Being able to detect a consistent signal strength reading for each visible base station, a unique 'location signature' was created for each location. That is, a single range or combination of ranges of base station signal strength data were calculated for each location that uniquely identified the particular location. For example, Table A.1 shows the location signatures for example locations in the home and University environments. The table states that only when the host device is at the particular location, will it see that particular combination of ranges. That is, only in the lounge area within the home environment will a signal strength between 55 and 67 for BS1 be received, and only in room 12.01 within the University environment will the combination of a signal strength between 50

and 57 for BS1 and between 34 and 45 for BS2 be received.

As both test environments had only a small number of locations, it was simple enough to calculate location signatures by hand. For larger deployments with many locations, this process could be automated by using a classifier system such as C5.0 [148].

It was then straightforward to determine the current location of a host device. The cut-off and smoothing filters were applied to the signal strength readings supplied by Placelab, and then the result was checked against the known location signatures.

### A.2.3 Improving confidence by exploiting location structure

With such a location system, there will be a degree of uncertainty in the results it produces. The system attempts to detect the locations of mobile entities, whose locations may have changed by the time the previous location reports are processed. The locations are detected from noisy signal strength data, where the readings may lie outwith the limits of known location signatures. In order to help applications cope with such uncertainty, locations were reported with an associated confidence value.

A simple and convenient means to support location confidence values was to maintain a history vector of the detected locations. This allowed confidence values of the reported locations to be calculated as $L_i/N$ where $L_i$ is the number of times a location $i$ featured in the history vector, over the size $N$ of the vector. The length of the vector was chosen to match the length of the smoothing history vector. That way, both vectors worked over the same time span (the previous eight seconds), and so calculating confidence values did not introduce any additional time delay in reporting the current location.

The structure present within the location model was exploited to increase the confidence of reported locations. Within the location model, two different locations may share the same parent location. When this occurs, the confidence of the parent location can be calculated, which may result in a higher overall confidence than for either of the child locations.

In the example given in Figure A.9, the location L10 Speaker Area has a confidence value of 0.875, meaning that it appeared 7 out of 8 times in the confidence history vector. The location L10 Audience Area has a confidence value of 0.125,

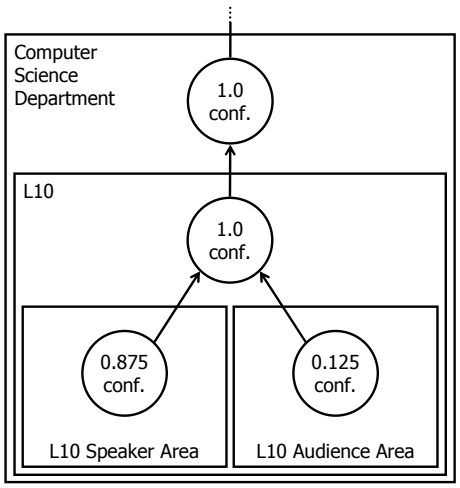



Figure A.9: The location model fuses confidence values of several inner locations to increase the confidence of outer locations.

as it appeared once in the confidence history vector. Both of these locations were within the location L10, meaning that all locations in the history vector are within L10, giving this location a confidence value of 1.0.

Calculating parent location confidences like this can be applied up through all parent locations. For example, in Figure A.9, as the location Computer Science Department contains L10, it also has a 1.0 confidence value.

Using this technique allowed higher confidence reports to be generated for larger locations, based upon several lower confidence reports detected by the system for smaller areas.

### A.2.4 Location detection application

With a unique location signature for each location of the test environments, it was simple to construct the location detection application that applied the cut-off and smoothing filters to the signal strength readings supplied by Placelab, and check the result against the known location signatures to determine the host device's current location, and calculate the confidence values of this location as well as any parent locations.

The location detection application was tested at both the home and University test environments. To test the application, it was run continuously on a Pocket PC that was moved between each location in the environment in turn. The

application logged the current location and confidence values every second. For each location, several points in the log were recorded - the point at which the Pocket PC was settled in the new location, the point at which the eight second length of the history vector had passed, and the point at which a further sixty seconds had passed. This essentially gives a 'settling in' period, while the history vector is flushed of previous locations' signals, and then a stationary period, in which only the current location should have been detected. At both environments, the tests were repeated on two separate occasions.

In both environments, for each location, the location detection application had settled on the correct location (the correct current location had the highest confidence of all locations reported) within the initial settling in period. Within the home environment, each location was correctly identified with an average confidence of 99.7%. Each location within the University environment was also correctly identified, but with a slightly lower average confidence of 91%.

## A.3 Conclusions

The aim of the work described in this Appendix was to implement appropriate location models and a location detection system that would enable testing of the pervasive situation determination middleware prototype and situation-aware applications that were developed in this thesis, in all of the target test environments and on each of the intended host device platforms.

This Appendix has shown how structured, hierarchical location models featuring symbolic locations and their associated location types were created for the test environments. Furthermore, it was shown how the Placelab location system was extended to enhance the accuracy of the system for use in the test environments. The extension applied a sequence of filters that successively refined the raw 802.11 signal strength data provided by Placelab, which were then mapped to symbolic location co-ordinates defined in the model. The structure of the model was then exploited to boost the confidence of the final location reported by the system.

In conclusion, these resulted in a location detection system that could produce sufficiently accurate location reports at the required granularity, that it could be used successfully by the pervasive situation determination middleware and situation-aware applications.

# Appendix B

# Additional situation diagrams and ontology

This appendix presents additional materials related to the modelling aspects of the pervasive situation determination approach, which in the interests of space were omitted from the main body of the thesis, and are included here for reference.

Additional situation diagrams that were omitted from Chapter 5 are included in Section B.1. The full OWL listings of the situation ontology used by the situation determination middleware prototype implementation as well as of the example Presentation situation, specification and customisations discussed in Chapter 3 are presented in Section B.2.

## B.1 Situation diagrams

This section includes additional situation diagrams drawn from the different settings that were presented in Chapter 5. Additional diagrams concerning the use of devices and applications are shown in Section B.1.1, additional public setting diagrams are included in Section B.1.2, and additional University setting diagrams are shown in Section B.1.3. A key explaining the different shapes, colours and links used in the figures in this section is given in Figure B.1.

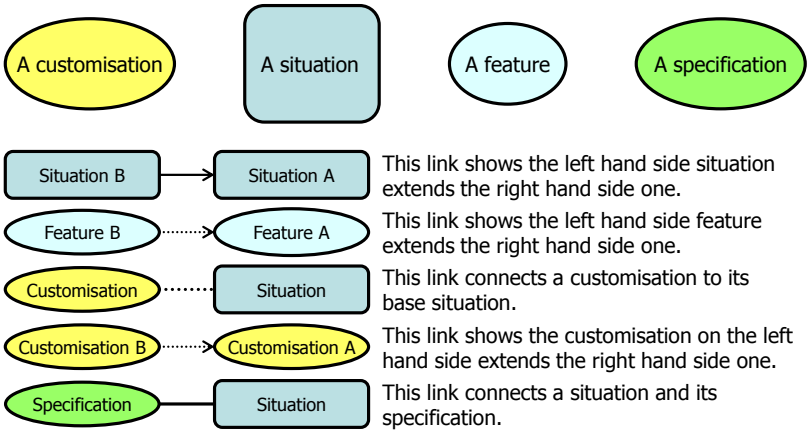



Figure B.1: A key for the figures included in this section.

### B.1.1 Situations concerning the use of devices and applications

Figure B.2 shows the 'Working with media', 'Editing media' and 'Viewing media' situations, which were originally shown in Figure 5.4 in Section 5.3.2, with the full set of customisations that were developed for them.

### B.1.2 Public situations

Figure B.3 shows the full set of customisations that were developed for the 'Dining' and 'Shopping' situations, which were originally shown in Figure 5.6 in Section 5.3.2.

### B.1.3 University situations

Figure B.4 presents diagrams from the 'Working at desk', 'Working in office', 'Private phone call' and break situations that were omitted from Section 5.3.2.

Figure B.5 shows all of the customisations that were developed for the 'Demonstrator meeting' situation shown previously in Figure 5.7 in Section 5.3.2.

The full set of customisations developed for the 'Lecture' situation shown in Figure 5.8 in Section 5.3.2 are shown in Figure B.6.

The diagrams for the 'Lab' and 'Tutorial' situations that were omitted from Section 5.3.2 are shown in Figure B.7 and Figure B.8 respectively.

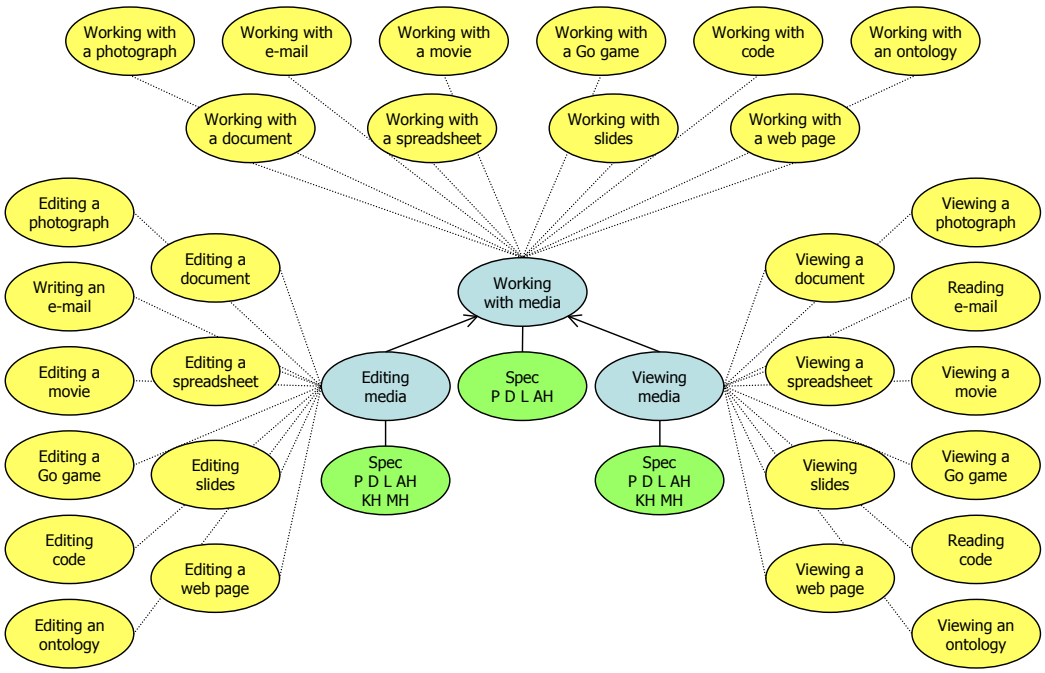



Figure B.2: All customisations for the working with media based situations.

## B.2 Ontologies and examples

This section presents the OWL representation of the ontologies used by the situation determination middleware and the example Presentation situation, specification and customisations discussed in Chapter 3.

### B.2.1 Situation ontology

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
]>


<rdf:RDF
  xmlns     = "http://sitdet.org/sitdet#"
  xml:base  = "http://sitdet.org/sitdet#"
  xmlns:owl = "http://www.w3.org/2002/07/owl#"
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">


  <!-- Situation -->

  <owl:DatatypeProperty rdf:ID="feature">
    <rdfs:domain rdf:resource="#Situation" />
    <rdfs:range  rdf:resource="&xsd;anyType"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="Situation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#feature"/>
        <owl:minCardinality rdf:datatype=
            "&xsd;nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>


  <!-- Specifications -->

  <!-- Role Specifications -->

  <owl:Class rdf:ID="EntityDeclaration">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#name"/>
```

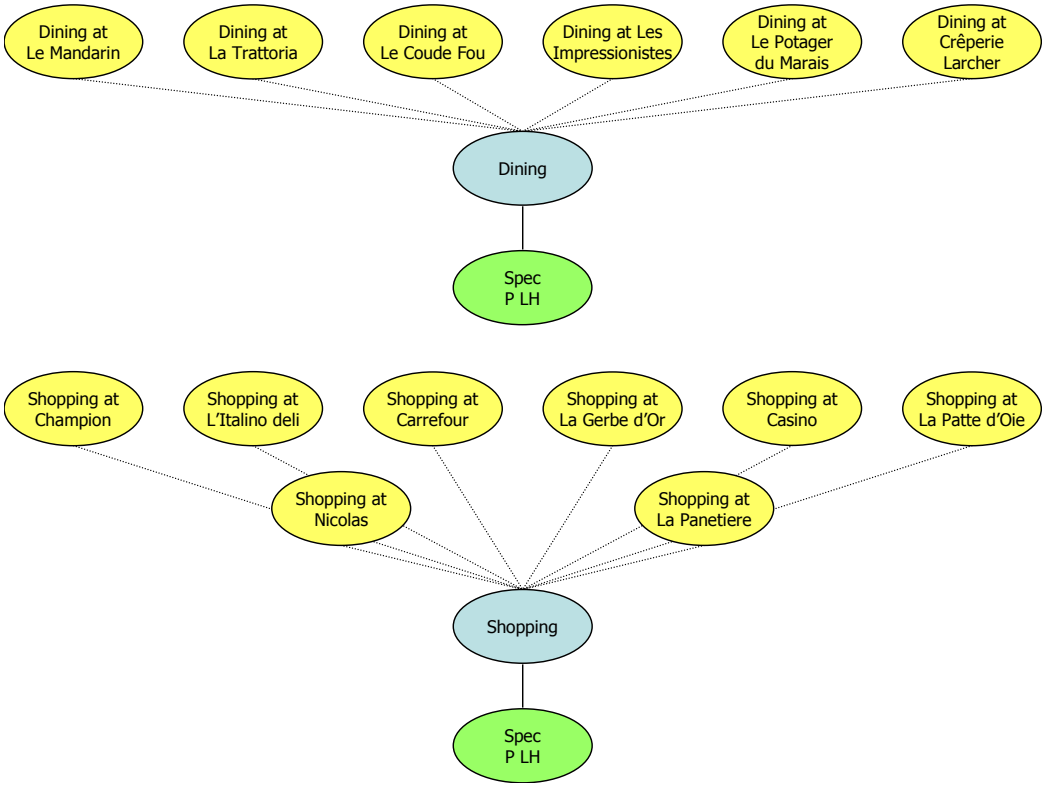



Figure B.3: All customisations for the 'Dining' and 'Shopping' situations.

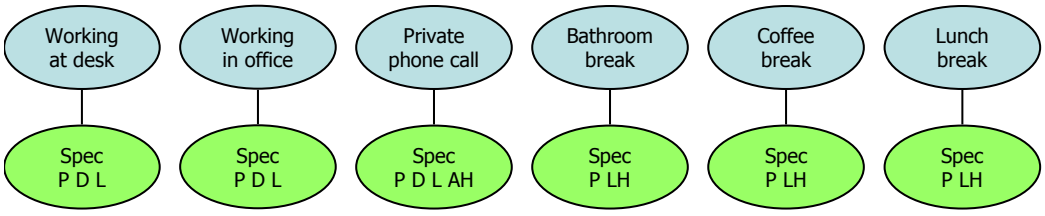



Figure B.4: Additional University setting situations.

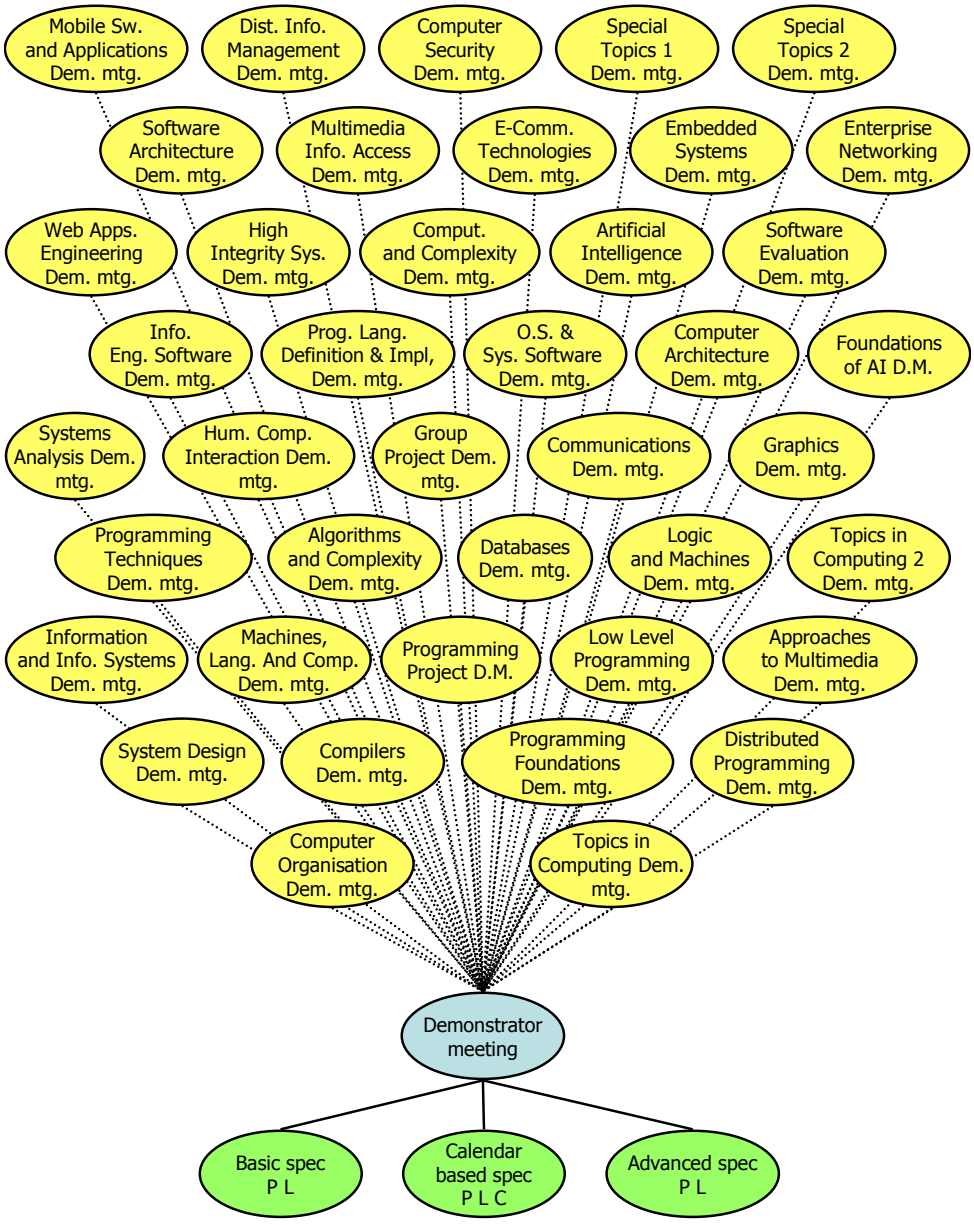



Figure B.5: All customisations for the 'Demonstrator meeting' situation.

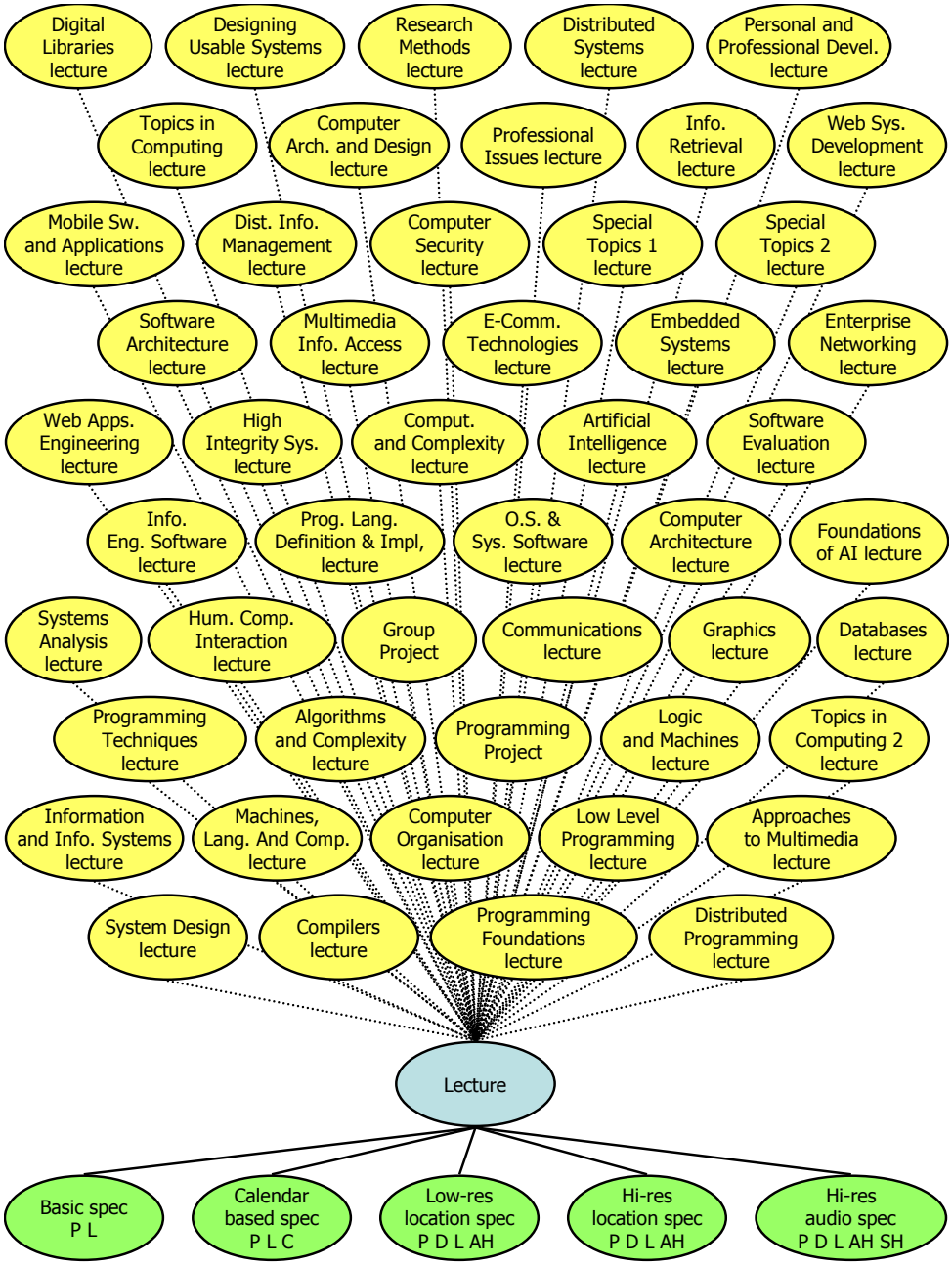



Figure B.6: All customisations for the 'Lecture' situation.

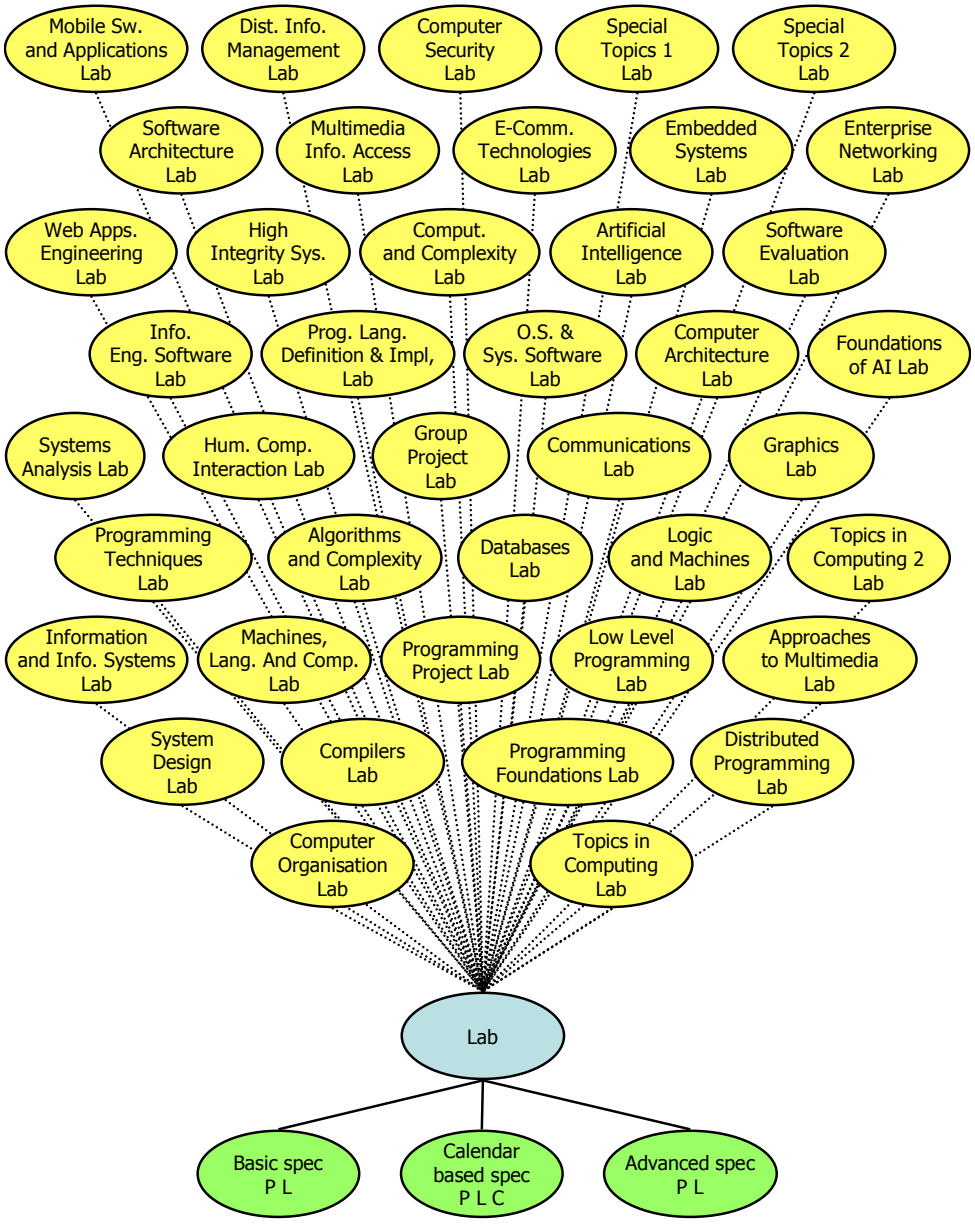



Figure B.7: All customisations for the 'Lab' situation.

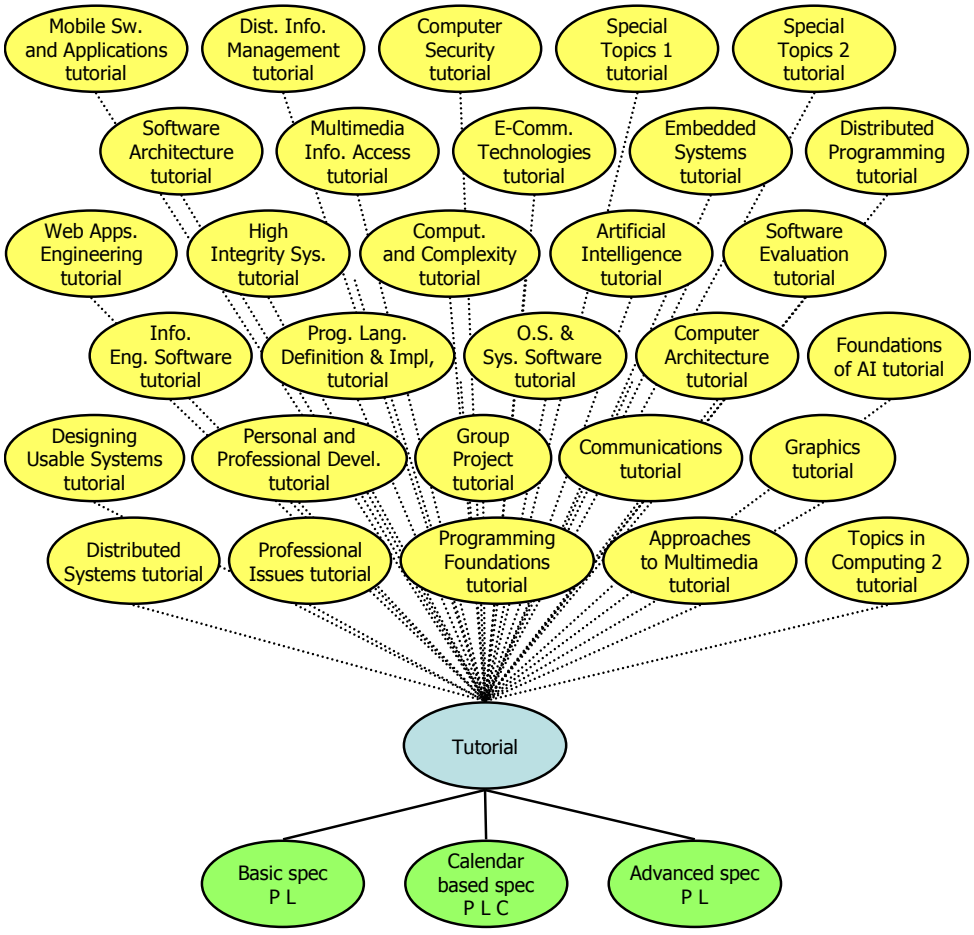



Figure B.8: All customisations for the 'Tutorial' situation.

```
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#type"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#EntityDeclaration" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="type">
  <rdfs:domain rdf:resource="#EntityDeclaration" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="RoleSpecification">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#roleType"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#primaryEntity"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#roleExpression"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="roleType">
  <rdfs:domain rdf:resource="#RoleSpecification" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="primaryEntity">
  <rdfs:domain rdf:resource="#RoleSpecification" />
  <rdfs:range rdf:resource="#EntityDeclaration" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="auxiliaryEntity">
  <rdfs:domain rdf:resource="#RoleSpecification" />
  <rdfs:range rdf:resource="#EntityDeclaration" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="roleExpression">
  <rdfs:domain rdf:resource="#RoleSpecification" />
  <rdfs:range rdf:resource="#Expression" />
</owl:ObjectProperty>


<!-- Situation Specifications. -->

<owl:Class rdf:ID="RoleDeclaration">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#declRoleType"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#entityName"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#entityType"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="declRoleType">
  <rdfs:domain rdf:resource="#RoleDeclaration" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="entityName">
  <rdfs:domain rdf:resource="#RoleDeclaration" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="entityType">
  <rdfs:domain rdf:resource="#RoleDeclaration" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="Specification">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#describes"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#areaOfInfluence"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#role"/>
      <owl:minCardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#expression"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="describes">
  <rdfs:domain rdf:resource="#Specification" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="areaOfInfluence">
```

```
  <rdfs:domain rdf:resource="#Specification" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="role">
  <rdfs:domain rdf:resource="#Specification" />
  <rdfs:range  rdf:resource="#RoleDeclaration"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="expression">
  <rdfs:domain rdf:resource="#Specification" />
  <rdfs:range  rdf:resource="#Expression"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="featureBinding">
  <rdfs:domain rdf:resource="#Specification" />
  <rdfs:range  rdf:resource="#FeatureBinding"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="averageResourceRequirements">
  <rdfs:domain rdf:resource="#Specification" />
  <rdfs:range  rdf:resource="#ResourceRequirementsMetrics"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="minimumResourceRequirements">
  <rdfs:domain rdf:resource="#Specification" />
  <rdfs:range  rdf:resource="#ResourceRequirementsMetrics"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="ResourceRequirementsMetrics" />

<owl:DatatypeProperty rdf:ID="cpuLoad">
  <rdfs:domain rdf:resource="#ResourceRequirementsMetrics" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="battery">
  <rdfs:domain rdf:resource="#ResourceRequirementsMetrics" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="memory">
  <rdfs:domain rdf:resource="#ResourceRequirementsMetrics" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="bandwidth">
  <rdfs:domain rdf:resource="#ResourceRequirementsMetrics" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>


<!-- Customisation. -->

<owl:Class rdf:ID="Customisation">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#extendsSituation"/>
      <owl:maxCardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#extendsCustomisation"/>
      <owl:maxCardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#customisationExpression"/>
      <owl:cardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="extendsSituation">
  <rdfs:domain rdf:resource="#Customisation" />
  <rdfs:range  rdf:resource="#Situation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="extendsCustomisation">
  <rdfs:domain rdf:resource="#Customisation" />
  <rdfs:range  rdf:resource="#Customisation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="customisationExpression">
  <rdfs:domain rdf:resource="#Customisation" />
  <rdfs:range  rdf:resource="#Expression"/>
</owl:ObjectProperty>


<!-- Expressions -->

<owl:Class rdf:ID="Expression" />

<owl:Class rdf:ID="Literal">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="data">
  <rdfs:domain rdf:resource="#Literal" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="BinaryBooleanExpression">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:ObjectProperty rdf:ID="lhs">
  <rdfs:domain rdf:resource="#BinaryBooleanExpression" />
  <rdfs:range rdf:resource="#Expression" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="rhs">
  <rdfs:domain rdf:resource="#BinaryBooleanExpression" />
  <rdfs:range rdf:resource="#Expression" />
</owl:ObjectProperty>

<owl:Class rdf:ID="And">
  <rdfs:subClassOf rdf:resource="#BinaryBooleanExpression" />
</owl:Class>

<owl:Class rdf:ID="GreaterThanOrEqualTo">
  <rdfs:subClassOf rdf:resource="#BinaryBooleanExpression" />
</owl:Class>

<owl:Class rdf:ID="EqualTo">
  <rdfs:subClassOf rdf:resource="#BinaryBooleanExpression" />
</owl:Class>

<owl:Class rdf:ID="UnaryBooleanExpression">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:Class rdf:ID="HasLocationType">
  <rdfs:subClassOf rdf:resource="#UnaryBooleanExpression" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="location">
  <rdfs:domain rdf:resource="#HasLocationType" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="locationType">
  <rdfs:domain rdf:resource="#HasLocationType" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="GetLocationByType">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="loc">
  <rdfs:domain rdf:resource="#GetLocationByTypeRoom" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="locType">
  <rdfs:domain rdf:resource="#GetLocationByTypeRoom" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="CurrentTime">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:Class rdf:ID="HasBeenSpeaking">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="person">
  <rdfs:domain rdf:resource="#HasBeenSpeaking" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="predicate">
  <rdfs:domain rdf:resource="#HasBeenSpeaking" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="percentOfDuration">
  <rdfs:domain rdf:resource="#HasBeenSpeaking" />
  <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="durationInSeconds">
  <rdfs:domain rdf:resource="#HasBeenSpeaking" />
  <rdfs:range rdf:resource="&xsd;int" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="IsRunningOn">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="iroApplication">
  <rdfs:domain rdf:resource="#IsRunningOn" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="iroComputer">
  <rdfs:domain rdf:resource="#IsRunningOn" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="IsDisplayedOn">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="idoDocument">
  <rdfs:domain rdf:resource="#IsDisplayedOn" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="idoDisplay">
  <rdfs:domain rdf:resource="#IsDisplayedOn" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="True">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:Class rdf:ID="False">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:Class rdf:ID="FeatureContains">
  <rdfs:subClassOf rdf:resource="#Expression" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="fcName">
  <rdfs:domain rdf:resource="#FeatureContains" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="fcValue">
  <rdfs:domain rdf:resource="#FeatureContains" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="FeatureBinding" />

<owl:DatatypeProperty rdf:ID="fbName">
  <rdfs:domain rdf:resource="#FeatureBinding" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="fbValue">
  <rdfs:domain rdf:resource="#FeatureBinding" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>


<!-- Location. -->

<owl:Class rdf:ID="Location" />

<owl:DatatypeProperty rdf:ID="hasSymbolicName">
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasLocationType">
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasParentLocation">
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="#Location" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasChildLocation">
  <owl:inverseOf rdf:resource="#hasParentLocation" />
  <rdfs:domain rdf:resource="#Location" />
  <rdfs:range rdf:resource="#Location" />
</owl:ObjectProperty>


</rdf:RDF>
```

## B.2.2 Example presentation ontology

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
]>

<rdf:RDF
  xmlns     = "http://sitdet.org/sitdet#"
  xml:base  = "http://sitdet.org/sitdet#"
  xmlns:owl = "http://www.w3.org/2002/07/owl#"
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">


  <!-- Presentation situation. -->

  <owl:Class rdf:ID="Presentation">
    <rdfs:subClassOf rdf:resource="#Situation" />
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="time">
    <rdfs:subPropertOf rdf:resource="#feature" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;dateTime" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="place">
    <rdfs:subPropertOf rdf:resource="#feature" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;anyURI" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="presentationAttendee">
    <rdfs:subPropertOf rdf:resource="#feature" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;string" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="speaker">
    <rdfs:subPropertOf rdf:resource="#presentationAttendee" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;string" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="audienceMember">
    <rdfs:subPropertOf rdf:resource="#presentationAttendee" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;string" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="display">
    <rdfs:subPropertOf rdf:resource="#feature" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;string" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="presentationDocument">
    <rdfs:subPropertOf rdf:resource="#feature" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;string" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="presentationApplication">
    <rdfs:subPropertOf rdf:resource="#feature" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;string" />
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="applicationHost">
    <rdfs:subPropertOf rdf:resource="#feature" />
    <rdfs:domain rdf:resource="#Presentation" />
    <rdfs:range rdf:resource="&xsd;string" />
  </owl:DatatypeProperty>


  <!-- Presentation hi-res specification. -->

  <!-- Hi-res location-based role specifications. -->
  <RoleSpecification rdf:ID="SpeakerHiResLocRoleSpec">
    <roleType>Speaker</roleType>
    <primaryEntity>
      <EntityDeclaration>
        <name>p</name>
        <type>Person</type>
      </EntityDeclaration>
    </primaryEntity>
    <roleExpression>
      <HasLocationType>
        <location>p.location</location>
        <locationType>SpeakerArea</locationType>
      </HasLocationType>
    </roleExpression>
  </RoleSpecification>

  <RoleSpecification rdf:ID="AudienceMemberHiResLocRoleSpec">
    <roleType>AudienceMember</roleType>
    <primaryEntity>
      <EntityDeclaration>
        <name>p</name>
        <type>Person</type>
      </EntityDeclaration>
    </primaryEntity>
    <roleExpression>
      <HasLocationType>
        <location>p.location</location>
        <locationType>AudienceArea</locationType>
      </HasLocationType>
    </roleExpression>
  </RoleSpecification>

  <!-- Hi-res audio-based role specifications. -->

  <RoleSpecification rdf:ID="SpeakerIsSpeakingRoleSpec">
    <roleType>Speaker</roleType>
    <primaryEntity>
      <EntityDeclaration>
        <name>p</name>
        <type>Person</type>
      </EntityDeclaration>
    </primaryEntity>
    <roleExpression>
      <And>
        <lhs>
          <HasLocationType>
            <location>p.location</location>
            <locationType>PresentationArea</locationType>
          </HasLocationType>
        </lhs>
        <rhs>
          <HasBeenSpeaking>
            <person>p</person>
            <predicate>greaterThanOrEqualTo</predicate>
            <percentOfDuration rdf:datatype=
                "&xsd;double">0.8</percentOfDuration>
            <durationInSeconds rdf:datatype=
                "&xsd;int">300</durationInSeconds>
          </HasBeenSpeaking>
        </rhs>
      </And>
```

```
  </roleExpression>
</RoleSpecification>

<RoleSpecification rdf:ID="AudienceMemberIsSpeakingRoleSpec">
  <roleType>AudienceMember</roleType>
  <primaryEntity>
    <EntityDeclaration>
      <name>p</name>
      <type>Person</type>
    </EntityDeclaration>
  </primaryEntity>
  <roleExpression>
    <And>
      <lhs>
        <HasLocationType>
          <location>p.location</location>
          <locationType>PresentationArea</locationType>
        </HasLocationType>
      </lhs>
      <rhs>
        <HasBeenSpeaking>
          <person>p</person>
          <predicate>lessThanOrEqualTo</predicate>
          <percentOfDuration rdf:datatype=
              "&xsd;double">0.2</percentOfDuration>
          <durationInSeconds rdf:datatype=
              "&xsd;int">300</durationInSeconds>
        </HasBeenSpeaking>
      </rhs>
    </And>
  </roleExpression>
</RoleSpecification>

<!-- Other role specifications. -->

<RoleSpecification rdf:ID="DisplayRoleSpec">
  <roleType>Display</roleType>
  <primaryEntity>
    <EntityDeclaration>
      <name>d</name>
      <type>Display</type>
    </EntityDeclaration>
  </primaryEntity>
  <roleExpression>
    <HasLocationType>
      <location>d.location</location>
      <locationType>PresentationArea</locationType>
    </HasLocationType>
  </roleExpression>
</RoleSpecification>

<RoleSpecification rdf:ID="PresentationDocumentRoleSpec">
  <roleType>PresentationDocument</roleType>
  <primaryEntity>
    <EntityDeclaration>
      <name>pd</name>
      <type>Document</type>
    </EntityDeclaration>
  </primaryEntity>
  <roleExpression>
    <True />
  </roleExpression>
</RoleSpecification>

<RoleSpecification rdf:ID="PresentationApplicationRoleSpec">
  <roleType>PresentationApplication</roleType>
  <primaryEntity>
    <EntityDeclaration>
      <name>pa</name>
      <type>Application</type>
    </EntityDeclaration>
  </primaryEntity>
  <roleExpression>
    <True />
  </roleExpression>
</RoleSpecification>

<RoleSpecification rdf:ID="ApplicationHostRoleSpec">
  <roleType>ApplicationHost</roleType>
  <primaryEntity>
    <EntityDeclaration>
      <name>cmp</name>
      <type>Computer</type>
    </EntityDeclaration>
  </primaryEntity>
  <roleExpression>
    <HasLocationType>
      <location>cmp.location</location>
      <locationType>PresentationArea</locationType>
    </HasLocationType>
  </roleExpression>
</RoleSpecification>

<!-- The Presentation situation specification. -->

<Specification rdf:ID="PresentationHiResSpec">
  <describes>Presentation</describes>

  <areaOfInfluence>Room</areaOfInfluence>

  <role>
    <RoleDeclaration>
      <declRoleType>Speaker</declRoleType>
      <entityName>spkr</entityName>
      <entityType>Person</entityType>
    </RoleDeclaration>
  </role>
  <role>
    <RoleDeclaration>
      <declRoleType>AudienceMember</declRoleType>
      <entityName>aud</entityName>
      <entityType>Person</entityType>
    </RoleDeclaration>
  </role>
  <role>
    <RoleDeclaration>
      <declRoleType>Display</declRoleType>
      <entityName></entityName>
      <entityType></entityType>
    </RoleDeclaration>
  </role>
  <role>
    <RoleDeclaration>
      <declRoleType>PresentationDocument</declRoleType>
      <entityName>doc</entityName>
      <entityType>Document</entityType>
    </RoleDeclaration>
  </role>
  <role>
    <RoleDeclaration>
      <declRoleType>PresentationApplication</declRoleType>
      <entityName>app</entityName>
      <entityType>Application</entityType>
    </RoleDeclaration>
  </role>
  <role>
    <RoleDeclaration>
      <declRoleType>ApplicationHost</declRoleType>
      <entityName>cmp</entityName>
      <entityType>Computer</entityType>
    </RoleDeclaration>
  </role>

  <expression>
    <And>
      <lhs>
```

```
<And>
  <lhs>
    <And>
      <lhs>
        <And>
          <lhs>
            <GreaterThanOrEqualTo>
              <lhs>
                <Literal>
                  <data>spkr.cardinality</data>
                </Literal>
              </lhs>
              <rhs>
                <Literal>
                  <data>1</data>
                </Literal>
              </rhs>
            </GreaterThanOrEqualTo>
          </lhs>
          <rhs>
            <GreaterThanOrEqualTo>
              <lhs>
                <Literal>
                  <data>spkr.confidence</data>
                </Literal>
              </lhs>
              <rhs>
                <Literal>
                  <data>0.8</data>
                </Literal>
              </rhs>
            </GreaterThanOrEqualTo>
          </rhs>
        </And>
      </lhs>
      <rhs>
        <And>
          <lhs>
            <GreaterThanOrEqualTo>
              <lhs>
                <Literal>
                  <data>aud.cardinality</data>
                </Literal>
              </lhs>
              <rhs>
                <Literal>
                  <data>2</data>
                </Literal>
              </rhs>
            </GreaterThanOrEqualTo>
          </lhs>
          <rhs>
            <GreaterThanOrEqualTo>
              <lhs>
                <Literal>
                  <data>aud.confidence</data>
                </Literal>
              </lhs>
              <rhs>
                <Literal>
                  <data>0.8</data>
                </Literal>
              </rhs>
            </GreaterThanOrEqualTo>
          </rhs>
        </And>
      </rhs>
    </And>
  </lhs>
  <rhs>
    <And>
      <lhs>
        <EqualTo>
          <lhs>
            <GetLocationByType>
              <loc>spkr.loc</loc>
              <locType>Room</locType>
            </GetLocationByType>
          </lhs>
          <rhs>
            <GetLocationByType>
              <loc>dsp.loc</loc>
              <locType>Room</locType>
            </GetLocationByType>
          </rhs>
        </EqualTo>
      </lhs>
      <rhs>
        <EqualTo>
          <lhs>
            <GetLocationByType>
              <loc>aud.loc</loc>
              <locType>Room</locType>
            </GetLocationByType>
          </lhs>
          <rhs>
            <GetLocationByType>
              <loc>dsp.loc</loc>
              <locType>Room</locType>
            </GetLocationByType>
          </rhs>
        </EqualTo>
      </rhs>
    </And>
  </rhs>
</And>
</lhs>
<rhs>
  <And>
    <lhs>
      <And>
        <lhs>
          <IsRunningOn>
            <iroApplication>app</iroApplication>
            <iroComputer>cmp</iroComputer>
          </IsRunningOn>
        </lhs>
        <rhs>
          <IsDisplayedOn>
            <idoDocument>doc</idoDocument>
            <idoDisplay>dsp</idoDisplay>
          </IsDisplayedOn>
        </rhs>
      </And>
    </lhs>
    <rhs>
      <And>
        <lhs>
          <EqualTo>
            <lhs>
              <Literal>
                <data>app.mode</data>
              </Literal>
            </lhs>
            <rhs>
              <Literal>Slideshow</Literal>
            </rhs>
          </EqualTo>
        </lhs>
        <rhs>
          <EqualTo>
            <lhs>
              <Literal>
                <data>app.file</data>
              </Literal>
            </lhs>
```

```
              <rhs>
                <Literal>
                  <data>doc</data>
                </Literal>
              </rhs>
            </EqualTo>
          </rhs>
        </And>
      </rhs>
    </And>
  </rhs>
</And>
</expression>

<featureBinding>
  <FeatureBinding>
    <fbName>time</fbName>
    <fbValue>
      <CurrentTime />
    </fbValue>
  </FeatureBinding>
</featureBinding>
<featureBinding>
  <FeatureBinding>
    <fbName>place</fbName>
    <fbValue>
      <GetLocationByType>
        <loc>dsp.location</loc>
        <locType>Room</locType>
      </GetLocationByType>
    </fbValue>
  </FeatureBinding>
</featureBinding>
<featureBinding>
  <FeatureBinding>
    <fbName>speaker</fbName>
    <fbValue>
      <Literal>
        <data>spkr</data>
      </Literal>
    </fbValue>
  </FeatureBinding>
</featureBinding>
<featureBinding>
  <FeatureBinding>
    <fbName>audienceMember</fbName>
    <fbValue>
      <Literal>
        <data>aud</data>
      </Literal>
    </fbValue>
  </FeatureBinding>
</featureBinding>
<featureBinding>
  <FeatureBinding>
    <fbName>display</fbName>
    <fbValue>
      <Literal>
        <data>dsp</data>
      </Literal>
    </fbValue>
  </FeatureBinding>
</featureBinding>
<featureBinding>
  <FeatureBinding>
    <fbName>presentationDocument</fbName>
    <fbValue>
      <Literal>
        <data>doc</data>
      </Literal>
    </fbValue>
  </FeatureBinding>
</featureBinding>
<featureBinding>
  <FeatureBinding>
    <fbName>presentationApplication</fbName>
    <fbValue>
      <Literal>
        <data>app</data>
      </Literal>
    </fbValue>
  </FeatureBinding>
</featureBinding>
<featureBinding>
  <FeatureBinding>
    <fbName>applicationHost</fbName>
    <fbValue>
      <Literal>
        <data>cmp</data>
      </Literal>
    </fbValue>
  </FeatureBinding>
</featureBinding>

<!-- The expected average resource requirements
        that this specification will consume. -->
<averageResourceRequirements>
  <ResourceRequirementsMetrics>
    <cpuLoad>1500</cpuLoad>
    <memory>8M</memory>
    <bandwidth>100Kb</bandwidth>
    <battery>5W</battery>
  </ResourceRequirementsMetrics>
</averageResourceRequirements>

<!-- The minimum resources that must be available
       for this specification to run. -->
<minimumResourceRequirements>
  <ResourceRequirementsMetrics>
    <cpuLoad>300</cpuLoad>
    <memory>4M</memory>
    <bandwidth>50Kb</bandwidth>
    <battery>5W</battery>
  </ResourceRequirementsMetrics>
</minimumResourceRequirements>

</Specification>


<!-- Example presentation customisations. -->

<Customisation rdf:ID="JohnAsSpeakerCustomisation">
  <extendsSituation rdf:resource="#Presentation" />
  <customisationExpression>
    <FeatureContains>
      <fcName>Speakers</fcName>
      <fcValue>johnUUID</fcValue>
    </FeatureContains>
  </customisationExpression>
</Customisation>

<Customisation rdf:ID="ForensicsPresentation">
  <extendsSituation rdf:resource="#Presentation" />
  <customisationExpression>
    <And>
      <lhs>
        <IncludesLocation>
          <ilName>Place</ilName>
          <ilValue>L10.01</ilValue>
        </IncludesLocation>
      </lhs>
      <rhs>
        <And>
          <lhs>
            <TimeIsBetween>
              <tibName>Time</tibName>
```

```
      <tibValue>15:00,16:00</tibValue>
    </TimeIsBetween>
  </lhs>
  <rhs>
    <DayEquals>
      <deName>Time</deName>
      <deValue>Friday</deValue>
    </DayEquals>
                                        </rhs>
                                      </And>
                                    </rhs>
                                  </And>
                                </customisationExpression>
                              </Customisation>

                            </rdf:RDF>
```

# Appendix C

# Implementation issues

This appendix presents further details about implementation issues concerning the pervasive situation determination middleware prototype and situation-aware applications that were developed, but in the interests of space were omitted from the main body of the thesis, and are included here for reference.

Translation of the Presentation specification into Jess code as discussed in Chapter 5, is detailed in Section C.1. Following this, Section C.2 highlights some of the implementation techniques that were used in developing the sensing infrastructure and situation-aware applications as discussed in Chapter 5.

## C.1 Translating situation specifications

When a situation determination agent (SDA) receives a specification to be recognised, it must transform the specification into Jess code in order for it to be incorporated into the SDA's Rete network. This transformation process also identifies which properties are required to recognise the specification, and to which the SDA must subscribe.

As an example of this transformation, this section looks at how the SDA would process the Presentation situation specification presented earlier in Chapter 3 in Figure 3.4, and listed in full in Section B.2.2 of Appendix B.

There are three main steps to the translation. The first is to identify all the properties that are referenced in the specification's expression and feature bindings parts. Note that some properties are identified through a series of relations. For example, the location of the speaker that has type 'Room' must be identified.

```
(defrule recognise-PresentationHiResSpec
    ; First, make matches on tuples for each role
    (tuple (subject-value ?spkr)
        (subject-type Person)
        (predicate hasRole)
        (object-value Speaker)
        (object-type Role)
        (confidence ?spkrConf))
    (tuple (subject-value ?aud)
        (subject-type Person)
        (predicate hasRole)
        (object-value Audience-member)
        (object-type Role)
        (confidence ?audConf))
    (tuple (subject-value ?dsp)
        (subject-type Display)
        (predicate hasRole)
        (object-value Presentation-display)
        (object-type Role)
        (confidence ?dspConf))
    (tuple (subject-value ?doc)
        (subject-type File)
        (predicate hasRole)
        (object-value Presentation-document)
        (object-type Role)
        (confidence ?docConf))
    (tuple (subject-value ?app)
        (subject-type Application)
        (predicate hasRole)
        (object-value Presentation-application)
        (object-type Role)
        (confidence ?appConf))
    (tuple (subject-value ?cmp)
        (subject-type Device)
        (predicate hasRole)
        (object-value Application-host)
        (object-type Role)
        (confidence ?hostConf))
    ; Then, expand all x.x.x forms
    (tuple (subject-value ?spkr)
        (subject-type Person)
        (predicate hasLocation)
        (object-value ?spkr-loc)
        (object-type Location)
        (confidence ?spkrLocConf))
    (tuple (subject-value ?spkr-loc)
        (subject-type Location)
        (predicate hasLocationType)
        (object-value Room)
        (object-type LocationType))
    (tuple (subject-value ?aud)
        (subject-type Person)
        (predicate hasLocation)
        (object-value ?aud-loc)
        (object-type Location)
        (confidence ?audLocConf))
    (tuple (subject-value ?aud-loc)
        (subject-type Location)
        (predicate hasLocationType)
        (object-value Room)
        (object-type LocationType))
    (tuple (subject-value ?dsp)
        (subject-type Display)
        (predicate hasLocation)
        (object-value ?dsp-loc)
        (object-type Location)
        (confidence ?dspLocConf))
    (tuple (subject-value ?dsp-loc)
        (subject-type Location)
        (predicate hasLocationType)
        (object-value Room)
        (object-type LocationType))
    (tuple (subject-value ?app)
        (subject-type Application)
        (predicate hasMode)
        (object-value ?app-mode)
        (object-type Mode)
        (confidence ?appModeConf))
    (tuple (subject-value ?app)
        (subject-type Application)
        (predicate hasActiveFile)
        (object-value ?doc)
        (object-type File))
    ; Then, check expressions
    (test (>= ?spkrConf 0.8))
    (test (>= ?audConf 0.8))
    (test (eq ?spkr-loc ?dsp-loc))
    (test (eq ?aud-loc ?dsp-loc))
    (tuple (subject-value ?app)
        (subject-type Application)
        (predicate isRunningOn)
        (object-value ?cmp)
        (object-type Device))
    (tuple (subject-value ?app)
        (subject-type Application)
        (predicate isDisplayedOn)
        (object-value ?dsp)
        (object-type Display))
    (test (eq ?app-mode Slideshow))
=>
    (call ?result add "Time" (time))
    (call ?result add "Place" ?dsp-loc)
    (call ?result add "Speakers" ?spkr)
    (call ?result add "Audience Members" ?aud)
    (call ?result add "Display" ?dsp)
    (call ?result add "Presentation Documents" ?doc)
    (call ?result add "Presentation Application" ?app)
    (call ?result add "Application Host" ?cmp)
    (call ?result add (min ?spkrConf ?audConf ?dspConf
            ?docConf ?appConf ?hostConf ?spkrLocConf
                ?audLocConf ?dspLocConf)))
```

Figure C.1: The resulting Jess code of translating the example presentation specification. In this listing, some comments have been added to the code to assist the explanation.

To do this, we must first find out the location of the speaker, and then given that location, identify whether it or one of its parent locations has the type 'Room'. The expanded set of properties enumerates all of the tuples that the SDA must subscribe to, and therefore which DCEAs and CCEAs are required to recognise the specification.

The second step of the translation is to identify the roles that appear in the specification, and extract any cardinality constraints that are placed upon them. These are stored for later processing, which is described below.

The third step of the translation is to generate Jess code for the specification. The result of the code generation for the presentation specification is shown in Figure C.1. The figure shows a single Jess rule. The 'if' part of the rule, shown before the '=>' symbol, lists all of the patterns that must match for the rule to fire. The 'then' part of the rule, shown after the '=>' symbol, lists all of the actions that are executed when the rule is fired.

The 'if' part concatenates three different parts. The first part includes expressions that match against each of the roles present in the specification. The second part lists each of the tuples and expansions required to access the necessary properties. The third part states the expressions of the specification.

Note that the cardinality expressions are missing from the 'if' part of the rule. In the implementation, the Jess code is executed inside a harness written in Java code. The harness creates the resulting feature set, executes the Jess code that fills the feature set, and then the harness checks that the final cardinalities of the features meet the constraints on them stated in the specification. In Figure C.1, in the 'then' part of the rule, the resulting feature set is filled, and the confidence of the match is added to it. Using a harness like this offers a simpler implementation than trying to do the cardinality checking with Jess code alone.

This example focused on translating a situation specification. The transformation of a role specification follows a similar process, though the expressions of the role specification will refer to the properties of its primary and auxiliary entities, rather than as here, where the expressions of the situation specification refer to the properties of the entities playing one of its roles. The Jess code translations of the role specifications used by the Presentation specification are listed in Figure C.2.

It is possible to improve execution time by pre-translating the specifications at design time, and thus saving the SDA from performing this at runtime. For example, in the University test environment presented in Chapter 5, translation could take up to 4 seconds (this cost is dominated by the libraries used to process the OWL descriptions). Doing so however, could limit the availability of the specification to SDAs that use the specific Rete framework, perhaps even the specific version, that was the target of the translated specification. Situation specifications are intended to run on a variety of SDAs in a variety of environments as they evolve over time. In the implementation, runtime translation was chosen as this allows the specifications to maintain the full richness of their OWL description, increase availability, and to automatically take advantage of any future

```
(defrule recognise-SpeakerHiResLocRoleSpec
    (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate location)
        (object-value ?loc)
        (object-type Location)
        (confidence ?c))
    (test (has-location-type
        (location ?loc)
        (type SpeakerArea)))
    =>
    (assert (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate hasRole)
        (object-value Speaker)
        (object-type Role)
        (confidence ?c))))

(defrule recognise-AudienceMemberHiResLocRoleSpec
    (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate location)
        (object-value ?loc)
        (object-type Location)
        (confidence ?c))
    (test (has-location-type
        (location ?loc)
        (type AudienceArea)))
    =>
    (assert (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate hasRole)
        (object-value AudienceMember)
        (object-type Role)
        (confidence ?c))))

(defrule recognise-SpeakerIsSpeakingRoleSpec
    (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate location)
        (object-value ?loc)
        (object-type Location)
        (confidence ?c))
    (test (has-location-type
        (location ?loc)
        (type PresentationArea)))
    (test (has-been-speaking p >= 0.8 300))
    =>
    (assert (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate hasRole)
        (object-value Speaker)
        (object-type Role)
        (confidence ?c))))

(defrule recognise-AudienceMemberIsSpeakingRoleSpec
    (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate location)
        (object-value ?loc)
        (object-type Location)
        (confidence ?c))
    (test (has-location-type
        (location ?loc)
        (type PresentationArea)))
    (test (has-been-speaking p <= 0.2 300))
    =>
    (assert (tuple
        (subject-value ?p)
        (subject-type Person)
        (predicate hasRole)
        (object-value AudienceMember)
        (object-type Role)
        (confidence ?c))))

(defrule recognise-DisplayRoleSpec
    (tuple
        (subject-value ?d)
        (subject-type Display)
        (predicate location)
        (object-value ?loc)
        (object-type Location)
        (confidence ?c))
    (test (has-location-type
        (location ?loc)
        (type PresentationArea)))
    =>
    (assert (tuple
        (subject-value ?d)
        (subject-type Display)
        (predicate hasRole)
        (object-value Display)
        (object-type Role)
        (confidence ?c))))

(defrule recognise-PresentationDocumentRoleSpec
    (tuple
        (subject-value ?pd)
        (subject-type Document)
        (predicate ?p)
        (object-value ?ov)
        (object-type ?ot)
        (confidence ?c))
    =>
    (assert (tuple
        (subject-value ?pd)
        (subject-type Document)
        (predicate hasRole)
        (object-value PresentationDocument)
        (object-type Role)
        (confidence ?c))))

(defrule recognise-PresentationApplicationRoleSpec
    (tuple
        (subject-value ?pa)
        (subject-type Application)
        (predicate ?p)
        (object-value ?ov)
        (object-type ?ot)
        (confidence ?c))
    =>
    (assert (tuple
        (subject-value ?pa)
        (subject-type Application)
        (predicate hasRole)
        (object-value PresentationApplication)
        (object-type Role)
        (confidence ?c))))

(defrule recognise-ApplicationHostRoleSpec
    (tuple
        (subject-value ?cmp)
        (subject-type Computer)
        (predicate location)
        (object-value ?loc)
        (object-type Location)
        (confidence ?c))
    (test (has-location-type
        (location ?loc)
        (type PresentationArea)))
    =>
    (assert (tuple
        (subject-value ?cmp)
        (subject-type Computer)
        (predicate hasRole)
        (object-value ApplicationHost)
        (object-type Role)
        (confidence ?c))))
```

Figure C.2: The resulting Jess code of translating the example presentation role specifications.

enhancement of an SDA's recognition mechanism.

## C.2 Implementation techniques

This section includes examples and pointers to further references for some of the more advanced implementation techniques used in the development of the pervasive situation determination middleware and example situation-aware applications that were discussed in Chapter 5. This section covers techniques for application monitoring and intercepting input events, which were used in implementing data context entity agents (DCEAs), as well as techniques for controlling system settings that were used in implementing the mode manager application.

### C.2.1 Application monitoring

In Chapter 5, the application monitor DCEA reported the window id of an application, the host machine id (these combined give a unique id for the application instance), the text of the application's title bar, the name of the executable of the application, whether the application was the active application on the host machine (the application that currently has input focus), and where possible, the active file (extracted from the title bar text). The code that detected these machine properties was implemented using the system diagnostic capabilities of the .NET platform, as well as Windows systems calls.

The following two sections provide examples of how to monitor applications on Windows XP and on a Pocket PC, respectively.

#### C.2.1.1 Monitoring applications on Windows XP

The following C# code can be used to build a log of all applications currently running on a Windows XP machine. These classes can be used to represent information about the application id, the executable name, the title bar text and whether or not it is the active application, for all open applications.

The following class represents an application's GUI window, and can be used to access the properties about that window:

```
public class Window
{
    /// <summary>
    /// Win32 API Imports
    /// </summary>
    [DllImport("user32.dll")]
    private static extern
    bool ShowWindowAsync(
        IntPtr hWnd, int nCmdShow);
    [DllImport("user32.dll")] private static extern
        bool SetForegroundWindow(IntPtr hWnd);
    [DllImport("user32.dll")] private static extern
```

```
    bool IsIconic(IntPtr hWnd);
[DllImport("user32.dll")] private static extern
    bool IsZoomed(IntPtr hWnd);
[DllImport("user32.dll")] private static extern
    IntPtr GetForegroundWindow();
[DllImport("user32.dll")] private static extern
    IntPtr GetWindowThreadProcessId(
        IntPtr hWnd, IntPtr ProcessId);
[DllImport("user32.dll")] private static extern
    IntPtr AttachThreadInput(IntPtr idAttach,
        IntPtr idAttachTo, int fAttach);

/// <summary>
/// Win32 API Constants for ShowWindowAsync()
/// </summary>
private const int SW_HIDE = 0;
private const int SW_SHOWNORMAL = 1;
private const int SW_SHOWMINIMIZED = 2;
private const int SW_SHOWMAXIMIZED = 3;
private const int SW_SHOWNOACTIVATE = 4;
private const int SW_RESTORE = 9;
private const int SW_SHOWDEFAULT = 10;

/// <summary>
/// Private Fields
/// </summary>
private IntPtr m_hWnd;
private string m_Title;
private bool m_Visible = true;
private string m_Process;
private bool m_WasMax = false;

/// <summary>
/// Window Object's Public Properties
/// </summary>
public IntPtr hWnd
{
    get{return m_hWnd;}
}
public string Title
{
    get{return m_Title;}
}
public string Process
{
    get{return m_Process;}
}
public bool IsForegroundWindow
{
    get{return m_hWnd == GetForegroundWindow();}
}

/// <summary>
/// Sets this Window Object's visibility
/// </summary>
public bool Visible
{
    get{return m_Visible;}
    set
    {
        //show the window
        if(value == true)
        {
            if(m_WasMax)
            {
                if(ShowWindowAsync(
                    m_hWnd,SW_SHOWMAXIMIZED))
                    m_Visible = true;
            }
            else
                {
                if(ShowWindowAsync(
                    m_hWnd,SW_SHOWNORMAL))
                    m_Visible = true;
                }
            }
            //hide the window
            if(value == false)
            {
                m_WasMax = IsZoomed(m_hWnd);
                if(ShowWindowAsync(m_hWnd,SW_HIDE))
                    m_Visible = false;
            }
        }
    }

    /// <summary>
    /// Constructs a Window Object
    /// </summary>
    /// <param name="Title">Title Caption</param>
    /// <param name="hWnd">Handle</param>
    /// <param name="Process">Owning Process</param>
    public Window(string Title, IntPtr hWnd, string Process)
    {
        m_Title = Title;
        m_hWnd = hWnd;
        m_Process = Process;
    }

    //Override ToString()
    public override string ToString()
    {
        return m_hWnd + "|" + m_Process + "|" + m_Title +
            "|" + IsForegroundWindow;
    }

    /// <summary>
    /// Sets focus to this Window Object
    /// </summary>
    public void Activate()
    {
        if(m_hWnd == GetForegroundWindow())
            return;

        IntPtr ThreadID1 = GetWindowThreadProcessId(
            GetForegroundWindow(), IntPtr.Zero);
        IntPtr ThreadID2 = GetWindowThreadProcessId(
            m_hWnd,IntPtr.Zero);

        if (ThreadID1 != ThreadID2)
        {
            AttachThreadInput(ThreadID1,ThreadID2,1);
            SetForegroundWindow(m_hWnd);
            AttachThreadInput(ThreadID1,ThreadID2,0);
        }
        else
        {
            SetForegroundWindow(m_hWnd);
        }

        if (IsIconic(m_hWnd))
        {
            ShowWindowAsync(m_hWnd,SW_RESTORE);
        }
        else
        {
            ShowWindowAsync(m_hWnd,SW_SHOWNORMAL);
        }
    }
}
```

The Window class relies on the following helper class, which provides wrappers

around the system calls to determine a thread or process id:

```
public class win32
{
    [DllImport("user32")]
    private static extern UInt32 GetWindowThreadProcessId(
        Int32 hWnd, out Int32 lpdwProcessId);

    public static Int32 GetWindowProcessID(Int32 hwnd)
    {
        Int32 pid = 1;
        GetWindowThreadProcessId(hwnd, out pid);
        return pid;
    }
}
```

The following class provides a means to access and iterate all of the application windows that are currently open in the GUI:

```
public class Windows : IEnumerable, IEnumerator
{
    /// <summary>
    /// Win32 API Imports
    /// </summary>
    [DllImport("user32.dll")] private static extern
        int GetWindowText(int hWnd, StringBuilder title, int size);
    [DllImport("user32.dll")] private static extern
        int GetWindowModuleFileName(
            int hWnd, StringBuilder title, int size);
    [DllImport("user32.dll")] private static extern
        int EnumWindows(EnumWindowsProc ewp, int lParam);
    [DllImport("user32.dll")] private static extern
        bool IsWindowVisible(int hWnd);
    [DllImport("user32.dll")] private static extern
        IntPtr GetWindowThreadProcessId(
            IntPtr hWnd, IntPtr ProcessId);

    //delegate used for EnumWindows() callback function
    public delegate bool EnumWindowsProc(
        int hWnd, int lParam);

    // holds current index of wndArray,
    // necessary for IEnumerable
    private int m_Position = -1;

    //array of windows
    ArrayList wndArray = new ArrayList();

    //Object's private fields
    private bool m_invisible = false;
    private bool m_notitle = false;

    /// <summary>
    /// Collection Constructor with additional options
    /// </summary>
    /// <param name="Invisible">Include invisible
        Windows</param>
    /// <param name="Untitled">Include untitled
        Windows</param>
    public Windows(bool Invisible, bool Untitled)
    {
        m_invisible = Invisible;
        m_notitle = Untitled;

        //Declare a callback delegate for
        // EnumWindows() API call
        EnumWindowsProc ewp = new EnumWindowsProc(
            EvalWindow);
        //Enumerate all Windows
        EnumWindows(ewp, 0);
    }
    /// <summary>
    /// Collection Constructor
    /// </summary>
    public Windows()
    {
        //Declare a callback delegate for
        //EnumWindows() API call
        EnumWindowsProc ewp = new EnumWindowsProc(
            EvalWindow);
        //Enumerate all Windows
        EnumWindows(ewp, 0);
    }
    //EnumWindows CALLBACK function
    private bool EvalWindow(int hWnd, int lParam)
    {
        if (m_invisible == false && !IsWindowVisible(hWnd))
        {
            return(true);
        }

        Int32 pid = win32.GetWindowProcessID(hWnd);
        Process p = Process.GetProcessById(pid);
        string appName = p.ProcessName;

        StringBuilder title = new StringBuilder(256);
        GetWindowText(hWnd, title, 256);

        if (m_notitle == false && title.Length == 0)
            return(true);

        wndArray.Add(new Window(
            title.ToString(), (IntPtr)hWnd, appName));

        return(true);
    }

    //implement IEnumerable
    public IEnumerator GetEnumerator()
    {
        return (IEnumerator)this;
    }

    //implement IEnumerator
    public bool MoveNext()
    {
        m_Position++;
        if (m_Position < wndArray.Count)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public void Reset()
    {
        m_Position = -1;
    }

    public object Current
```

```
    {
        get
        {
            return wndArray[m_Position];
        }
    }
}
```

Then, the applications' window information and name of the host machine on which the applications are running can be added to the log:

```
Windows windows = new Windows();
string log = "";
for (IEnumerator e = windows.GetEnumerator();
    e.MoveNext(); )
{
    Window w = (Window) e.Current;
    log += w.ToString() + "|" +
        System.Environment.MachineName + "\n";
}
```

### C.2.1.2 Monitoring applications on a Pocket PC

The following C# code can be used to detect which applications are running on a Pocket PC.

The following Process class represents an application process running on the Pocket PC, and can be used to access properties about the process, including the process name and its id (handle):

```
public class Process
{
    private string processName;
    private IntPtr handle;
    private int threadCount;
    private int usageCount;
    private int baseAddress;

    //default constructor
    public Process() {}

    private Process(IntPtr id, string procname,
        int threadcount, int usagecount, int baseaddress)
    {
        handle = id;
        processName = procname;
        threadCount = threadcount;
        usageCount = usagecount;
        baseAddress = baseaddress;
    }

    public string ToLongString()
    {
        return "Process name: " + processName +
            ", Base address: " + baseAddress +
            ", Thread Count: " + threadCount +
            ", Usage Count: " + usageCount +
            ", Handle: " + handle;
    }

    //ToString implementation for ListBox binding
    public override string ToString()
    {
        return processName;
    }

    public int BaseAddress
    {
        get{ return baseAddress; }
    }

    public int ThreadCount
    {
        get{ return threadCount; }
    }

    public int UsageCount
    {
        get{ return usageCount; }
    }

    public IntPtr Handle
    {
        get{ return handle; }
    }

    public string ProcessName
    {
        get{ return processName; }
    }

    public void Kill()
    {
        IntPtr hProcess;

        hProcess = OpenProcess(PROCESS_TERMINATE, false,
            (int) handle);

        if(hProcess != (IntPtr) INVALID_HANDLE_VALUE)
        {
            bool bRet;
            bRet = TerminateProcess(hProcess, 0);
            CloseHandle(hProcess);
        }
    }
```

The following static method is used to access and iterate the process information of all of the applications running on the Pocket PC:

```
public static Process[] GetProcesses()
{
    ArrayList procList = new ArrayList();

    IntPtr handle = CreateToolhelp32Snapshot(
        TH32CS_SNAPPROCESS, 0);

    if ((int)handle > 0)
    {
        try
        {
            PROCESSENTRY32 peCurrent;
            PROCESSENTRY32 pe32 = new PROCESSENTRY32();
            //Get byte array to pass to the API calls
            byte[] peBytes = pe32.ToByteArray();
            //Get the first process
            int retval = Process32First(handle,
                peBytes);

            while(retval == 1)
            {
                //Convert bytes to the class
                peCurrent = new PROCESSENTRY32(
                    peBytes);
                //New instance
                Process proc = new Process(
                    new IntPtr((int)peCurrent.PID),
                    peCurrent.Name,
                    (int)peCurrent.ThreadCount,
                    (int)peCurrent.UsageCount,
                    (int)peCurrent.BaseAddress);

                procList.Add(proc);

                retval = Process32Next(handle,
                    peBytes);
            }
        }
        catch(Exception ex)
        {
            throw new Exception("Exception: " +
                ex.Message);
        }

        //Close handle
        CloseToolhelp32Snapshot(handle);

        return (Process[])procList.ToArray(
            typeof(Process));
    }
    else
    {
        throw new Exception(
            "Unable to create snapshot");
    }
}
```

The remainder of the class provides wrappers around the system data structures and calls that are used to determine the process information:

```
private class PROCESSENTRY32
{
    // constants for structure definition
    private const int SizeOffset = 0;
    private const int UsageOffset = 4;
    private const int ProcessIDOffset=8;
    private const int DefaultHeapIDOffset = 12;
    private const int ModuleIDOffset = 16;
    private const int ThreadsOffset = 20;
    private const int ParentProcessIDOffset = 24;
    private const int PriClassBaseOffset = 28;
    private const int dwFlagsOffset = 32;
    private const int ExeFileOffset = 36;
    private const int MemoryBaseOffset = 556;
    private const int AccessKeyOffset = 560;
    private const int Size = 564;
    private const int MAX_PATH = 260;

    // data members
    public uint dwSize;
    public uint cntUsage;
    public uint th32ProcessID;
    public uint th32DefaultHeapID;
    public uint th32ModuleID;
    public uint cntThreads;
    public uint th32ParentProcessID;
    public long pcPriClassBase;
    public uint dwFlags;
    public string szExeFile;
    public uint th32MemoryBase;
    public uint th32AccessKey;

    //Default constructor
    public PROCESSENTRY32() {}

    // create a PROCESSENTRY instance
    // based on a byte array
    public PROCESSENTRY32(byte[] aData)
    {
        dwSize = GetUInt(aData, SizeOffset);
        cntUsage = GetUInt(aData, UsageOffset);
        th32ProcessID = GetUInt(aData,
            ProcessIDOffset);
        th32DefaultHeapID = GetUInt(aData, DefaultHeapIDOffset);
        th32ModuleID = GetUInt(aData, ModuleIDOffset);
        cntThreads = GetUInt(aData, ThreadsOffset);
        th32ParentProcessID = GetUInt(aData,
            ParentProcessIDOffset);
        pcPriClassBase = (long) GetUInt(aData,
            PriClassBaseOffset);
        dwFlags = GetUInt(aData, dwFlagsOffset);
        szExeFile = GetString(aData, ExeFileOffset,
            MAX_PATH);
        th32MemoryBase = GetUInt(aData,
            MemoryBaseOffset);
        th32AccessKey = GetUInt(aData,
            AccessKeyOffset);
    }

    #region Helper conversion functions
    // utility:  get a uint from the byte array
    private static uint GetUInt(byte[] aData,
        int Offset)
    {
        return BitConverter.ToUInt32(aData, Offset);
    }

    // utility:  set a uint int the byte array
```

```
        private static void SetUInt(byte[] aData,
            int Offset, int Value)
        {
            byte[] buint = BitConverter.GetBytes(Value);
            Buffer.BlockCopy(buint, 0, aData, Offset,
                buint.Length);
        }

        // utility:  get a ushort from the byte array
        private static ushort GetUShort(byte[] aData,
            int Offset)
        {
            return BitConverter.ToUInt16(aData, Offset);
        }

        // utility:  set a ushort int the byte array
        private static void SetUShort(byte[] aData,
            int Offset, int Value)
        {
            byte[] bushort = BitConverter.GetBytes(
                (short)Value);
            Buffer.BlockCopy(bushort, 0, aData, Offset,
                bushort.Length);
        }

        // utility:  get a unicode string
        // from the byte array
        private static string GetString(byte[] aData,
            int Offset, int Length)
        {
            String sReturn =  Encoding.Unicode.GetString(
                aData, Offset, Length);
            return sReturn;
        }

        // utility:  set a unicode string
        // in the byte array
        private static void SetString(byte[] aData,
            int Offset, string Value)
        {
            byte[] arr = Encoding.ASCII.GetBytes(Value);
            Buffer.BlockCopy(arr, 0, aData, Offset,
                arr.Length);
        }
        #endregion

        // create an initialized data array
        public byte[] ToByteArray()
        {
            byte[] aData;
            aData = new byte[Size];
            //set the Size member
            SetUInt(aData, SizeOffset, Size);
            return aData;
        }

        public string Name
        {
            get
            {
                return szExeFile.Substring(0,
                    szExeFile.IndexOf('\0'));
            }
        }

        public ulong PID
        {
            get{ return th32ProcessID; }
        }

        public ulong BaseAddress
        {
            get{ return th32MemoryBase; }
        }

        public ulong ThreadCount
        {
            get{ return cntThreads; }
        }

        public ulong UsageCount
        {
            get{ return cntUsage; }
        }
    }

    #region PInvoke declarations
    private const int TH32CS_SNAPPROCESS = 0x00000002;
    [DllImport("toolhelp.dll")]
    public static extern IntPtr CreateToolhelp32Snapshot(
        uint flags, uint processid);
    [DllImport("toolhelp.dll")]
    public static extern int CloseToolhelp32Snapshot(
        IntPtr handle);
    [DllImport("toolhelp.dll")]
    public static extern int Process32First(
        IntPtr handle, byte[] pe);
    [DllImport("toolhelp.dll")]
    public static extern int Process32Next(
        IntPtr handle, byte[] pe);
    [DllImport("coredll.dll")]
    private static extern IntPtr OpenProcess(
        int flags, bool fInherit, int PID);
    private const int PROCESS_TERMINATE = 1;
    [DllImport("coredll.dll")]
    private static extern bool TerminateProcess(
        IntPtr hProcess, uint ExitCode);
    [DllImport("coredll.dll")]
    private static extern bool CloseHandle(IntPtr handle);
    private const int INVALID_HANDLE_VALUE = -1;
    #endregion
}
```

A log entry can then be written for the Pocket PC with:

```
string log = "";
foreach (Process p in Process.GetProcesses())
{
    log += p.ToLongString() + "\n";
}
```

Further details of these features can be found in the 'System.Diagnostics' and 'System.Runtime.InteropServices' sections of the .NET platform documentation [149].

### C.2.2 Detecting input events

The keyboard and mouse DCEAs featured in Chapter 5 were implemented using a feature of the Windows operating system known as 'hooks'. A hook is a piece of code that receives and can manipulate system events. Systems events are passed through each hook installed, before passing the event to its destination application. Therefore, each key press and mouse event can be intercepted, recorded and communicated to the situation determination middleware before passing it on.

Keyboard presses on a Windows XP machine can be intercepted with the following C++ code:

```
HINSTANCE hins;
static HHOOK hkb=NULL;
string log;

LRESULT __declspec(dllexport)__stdcall  CALLBACK KeyboardProc(
    int nCode, WPARAM wParam, LPARAM lParam)
{
    char ch;
    if (((DWORD)lParam & 0x40000000) &&(HC_ACTION==nCode))
    {
        if ((wParam==VK_SPACE)
            ||(wParam==VK_RETURN)
            ||(wParam>=0x2f )
            &&(wParam<=0x100))
        {
            if (wParam==VK_RETURN)
            {
                ch='\n';
            }
            else
            {
                BYTE ks[256];
                GetKeyboardState(ks);
                WORD w;
                UINT scan;
                scan=0;
                ToAscii(wParam,scan,ks,&w,0);
                ch =char(w);
            }

            SYSTEMTIME st;
            GetSystemTime(&st);
            stringstream ss;
            ss << ch << ' ' << st.wYear
                << setfill('0') << setw(2) << st.wMonth
                << setw(2) << st.wDay
                << 'T'
                << setw(2) << st.wHour
                << setw(2) << st.wMinute
                << setw(2) << st.wSecond
                << '.' << setw(3) << st. wMilliseconds
                << char(255);
            log += ss.str();
        }
    }

    LRESULT RetVal = CallNextHookEx(
        hkb, nCode, wParam, lParam);
    return  RetVal;
}

BOOL __declspec(dllexport)__stdcall InstallKeyboardHook()
{
    hkb=SetWindowsHookEx(WH_KEYBOARD,
        (HOOKPROC) KeyboardProc, hins, 0);
    return TRUE;
}

void init()
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    hins=AfxGetInstanceHandle();
}
```

Similarly, the time of the last mouse event to occur can be captured with the following code:

```
static HHOOK hmouse=NULL;
SYSTEMTIME lastMouseActionTime;

LRESULT CALLBACK MouseProc(int nCode, WPARAM wParam,
    LPARAM lParam)
{
    LRESULT lResult = CallNextHookEx(
        hmouse, nCode, wParam, lParam);

    if(HC_ACTION == nCode)
    {
        GetSystemTime(&lastMouseActionTime);
    }

    return lResult;
}

BOOL __declspec(dllexport)__stdcall InstallMouseHook()
{
    hmouse = SetWindowsHookEx(WH_MOUSE,
        (HOOKPROC) MouseProc, hins, 0);
    return TRUE;
}
```

Further details and examples can be found in [150] and [151].

### C.2.3 Controlling system settings

Both muting a Pocket PC and disabling the Windows XP screen saver, as used by the mode manager application presented in Chapter 5, were achieved using Windows system calls.

Muting a Pocket PC can be done by setting the volume of the 'wave out' device to 0, using the following C++ code:

```
void SetSoundVolume(DWORD dwVolume)
{
    WAVEFORMATEX wf;
    wf.wFormatTag = WAVE_FORMAT_PCM;
    wf.nChannels = 1;
    wf.nSamplesPerSec = 8000 * 1000;
    wf.wBitsPerSample = 8;
    wf.nBlockAlign = wf.nChannels * wf.wBitsPerSample / 8;
    wf.nAvgBytesPerSec = wf.nSamplesPerSec * wf.nBlockAlign;
    wf.cbSize = 0;

    HWAVEOUT hwo;
    for (UINT id = 0; id < waveOutGetNumDevs(); id++)
    {
        if (waveOutOpen(&hwo, id, &wf, 0, 0, CALLBACK_NULL) ==
            MMSYSERR_NOERROR)
        {
            waveOutSetVolume(hwo, dwVolume);
            waveOutClose(hwo);
            break;
        }
    }
}

SetSoundVolume((DWORD) 0);
```

Disabling and re-enabling the screen saver of a Windows XP machine can be controlled through the following C# code:

```
public class ScreenSaverController
{
    [DllImport("user32.dll", CharSet=CharSet.Auto)]
    public static extern int SystemParametersInfo(
        int uAction, int uParam, int lpvParam, int fuWinIni);

    public const int SPI_SETSCREENSAVEACTIVE = 17;

    public static int disableScreenSaver()
    {
        return SystemParametersInfo(
            SPI_SETSCREENSAVEACTIVE, 0, 0, 0);
    }

    public static int enableScreenSaver()
    {
        return SystemParametersInfo(
            SPI_SETSCREENSAVEACTIVE, 1, 0, 0);
    }
}
```

See [152] and [153] for further details and examples.

# Bibliography


[1] Daniel Salber Anind K. Dey and Gregory D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI Journal*, 16(2-4), 2001.

[2] Roy Want, editor. *IEEE Pervasive Computing*, volume 1(1). IEEE, January-March 2002.

[3] Roy Want, editor. *The Smart Phone, IEEE Pervasive Computing*, volume 4(2). IEEE, April-June 2005.

[4] Norbert Streitz and Paddy Nixon, editors. *The Disappearing Computer, Communications of the ACM*, volume 48(3). ACM, New York, NY, USA, March 2005.

[5] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, 1992.

[6] Jason Pascoe. The Stick-e Note Architecture: Extending the Interface Beyond the User. In Johanna Moore, Ernest Edmonds, and Angel Puerta, editors, *1997 International Conference on Intelligent User Interfaces*, pages 261–264. ACM, January 1997.

[7] Lonnie D. Harvel, Ling Liu, Gregory D. Abowd, Yu-Xi Lim, Chris Scheibe, and Chris Chatham. Context cube: Flexible and effective manipulation of sensed context data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing, Second International Conference, PERVASIVE 2004*, pages 51–68, Vienna, Austria, April 18-23 2004. Springer.

[8] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37–64, February 2006.

[9] Anand Ranganathan and Roy H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 16-20*, 2003.

[10] Simon Dobson, Lorcan Coyle, and Paddy Nixon. Hybridising events and knowledge as a basis for building autonomic systems. *Journal of Trusted and Autonomic Computing*, 2006.

[11] C.B. Anagnostopoulos, Y. Ntarladimas, and S. Hadjiefthymiades. Situational computing: An innovative architecture with imprecise reasoning. *Journal of Systems and Software*, 80(12):1993–2014, December 2007.

[12] Matthai Philipose, Kenneth P. Fishkin, Mike Perkowitz, Donald J. Patterson, Dieter Fox, Henry Kautz, and Dirk Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.

[13] Paul Lukowicz, Jamie A. Ward, Holger Junker, Mathias Stäger, Gerhard Tröster, Amin Atrash, and Thad Starner. Recognizing workshop activity using body worn microphones and accelerometers. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing, Second International Conference, PERVASIVE 2004*, pages 18–32, Vienna, Austria, April 18-23 2004. Springer.

[14] Thomas Stiefmeier, Daniel Roggen, Georg Ogris, Paul Lukowicz, and Gerhard Trster. Wearable activity tracking in car manufacturing. *IEEE Pervasive Computing*, 7(2):42–50, 2008.

[15] Mark Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991.

[16] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Jeijten, and J-C. Burgelman. Scenarios for ambient intelligence in 2010. Technical report, Institute for Prospective Technological Studies (IPTS), February 2001. Available at: ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf.

[17] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM.

[18] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *Conference on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands*, April 2000.

[19] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[20] Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[21] Anand Ranganathan and Roy H. Campbell. An infrastructure for context-awareness based on first order logic. *Personal and Ubiquitous Computing*, 7(6):353–364, December 2003.

[22] A. Ranganathan, J. Al-Muhtadi, and R. H. Campbell. Reasoning about uncertain contexts in pervasive computing environments. *IEEE Pervasive Computing*, 3(2):62–70, 2004.

[23] Earl Cox. *The fuzzy systems handbook: A practitioner's guide to building, using, and maintaining fuzzy systems.* Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[24] Harry Chen, Tim Finin, and Anupam Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, November 2003.

[25] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, MA, August 2004.

[26] X. Wang, T. Gu, D. Zhang, and H. K. Pung. Ontology based context modeling and reasoning using owl. In *Workshop on Context Modeling and Reasoning (CoMoRea) at IEEE International Conference on Pervasive Computing and Communication (PerCom'04)*, March 2004.

[27] Graeme Stevenson, Stephen Knox, Simon Dobson, and Paddy Nixon. Ontonym: a collection of upper ontologies for developing pervasive systems. In *CIAO '09: Proceedings of the 1st Workshop on Context, Information and Ontologies*, pages 1–8. ACM, 2009.

[28] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Proceedings of the First International Conference on Pervasive Computing*, volume 2414 of *Lecture Notes in Computer Science*, pages 167–180, Zurich, August 2002. Springer-Verlag.

[29] K. Henricksen and J. Indulska. Modelling and using imperfect context information. In *Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, 2004.

[30] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *PerCom* [154], pages 77–86.

[31] Terry Halpin. *Information modeling and relational databases: From conceptual analysis to logical design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[32] The UbiSense real-time location platform. See: http://www.ubisense.net.

[33] B. Schilit, A. LaMarca, G. Borriello, W. Griswold, D. McDonald, E. Lazowska, A. Balachandran, J. Hong, and V. Iverson. Challenge: Ubiquitous location-aware computing and the Place Lab initiative. In *Proceedings of The First ACM International Workshop on Wireless Mobile Applications and Services on WLAN (WMASH 2003), San Diego, CA*, September 2003.

[34] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place Lab: Device positioning using radio beacons in the wild. In *Proceedings of PERVASIVE 2005, Third International Conference on Pervasive Computing*, Munich, Germany, 2005.

[35] James L. Crowley, Joëlle Coutaz, Gaeten Rey, and Patrick Reignier. Perceptual components for context aware computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*, pages 117–134, London, UK, 2002. Springer-Verlag.

[36] Joëlle Coutaz and Gaeten Rey. Foundations for a theory of contextors. In *Computer Aided Design of User Interfaces, Springer Verlag*, 2002.

[37] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[38] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2002.

[39] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME, Series D*, 82:35–45, 1960.

[40] Christian Becker and Frank Durr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[41] Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[42] H. Liu, Houshang Darabi, P. Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 37(6):1067–1080, 2007.

[43] Global Positioning System: Serving the World. See: http://www.gps.gov.

[44] Daniela Nicklas and Bernhard Mitschang. The NEXUS augmented world model: An extensible approach for mobile, spatially aware applications. In Yingxu Wang, Shushma Patel, and Ronald Johnston, editors, *7th International Conference on Object Oriented Information Systems (OOIS'2001)*, pages 392–. Springer, 2001.

[45] Kimberle Koile, Konrad Tollmar, David Demirdjian, Howard E. Shrobe, and Trevor Darrell. Activity zones for context-aware computing. In Dey et al. [155], pages 90–106.

[46] Gary Look, Buddhika Kottahachchi, Robert Laddaga, and Howard Shrobe. A location representation for generating descriptive walking directions. In *Proc. Intelligent User Interfaces*, pages 122–129, New York, NY, USA, 2005. ACM Press.

[47] Graeme Stevenson, Juan Ye, Simon Dobson, and Paddy Nixon. Loc8: A location model and extensible framework for programming with location. *IEEE Pervasive Computing*, 9:28–37, 2010.

[48] Seon-Woo Lee and Kenji Mase. Activity and location recognition using wearable sensors. *IEEE Pervasive Computing*, 1(3):24–32, 2002.

[49] Ling Bao and Stephen S. Intille. Activity recognition from user-annotated acceleration data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing, Second International Conference, PERVASIVE 2004*, pages 1–17, Vienna, Austria, April 18-23 2004. Springer.

[50] Paul Lukowicz, Friedrich Hanser, Christoph Szubski, and Wolfgang Schobersberger. Detecting and interpreting muscle activity with wearable force sensors. In Fishkin et al. [156], pages 101–116.

[51] Jonathan Lester, Tanzeem Choudhury, and Gaetano Borriello. A practical approach to recognizing physical activities. In Fishkin et al. [156], pages 1–16.

[52] Raphael Wimmer, Matthias Kranz, Sebastian Boring, and Albrecht Schmidt. A capacitive sensing toolkit for pervasive activity detection and recognition. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.

[53] Yoshifumi Nishida, Toshio Hori, Takeo Kanade, Koji Kitamura, Akifumi Nishitani, and Hiroshi Mizoguchi. Quick realization of function for detecting human activity events by ultrasonic 3D tag and stereo vision. In *PerCom* [154], pages 43–54.

[54] Shwetak N. Patel, Thomas Robertson, Julie A. Kientz, Matthew S. Reynolds, and Gregory D. Abowd. At the flick of a switch: Detecting and classifying unique electrical events on the residential power line. In Krumm et al. [157], pages 271–288.

[55] Jonathan Lester, Blake Hannaford, and Gaetano Borriello. "Are You with Me" - Using Accelerometers to Determine If Two Devices Are Carried by the Same Person. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing, Second International Conference, PERVASIVE 2004*, pages 33–50, Vienna, Austria, April 18-23 2004. Springer.

[56] Panu Korpipää, Miika Koskinen, Johannes Peltola, Satu-Marja Mäkelä, and Tapio Seppänen. Bayesian approach to sensor-based context awareness. *Personal and Ubiquitous Computing*, 7(2):113–124, 2003.

[57] MPEG-7 Overview. See: http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

[58] Kenneth P. Fishkin, Bing Jiang, Matthai Philipose, and Sumit Roy. I Sense a Disturbance in the Force: Unobtrusive Detection of Interactions with RFID-tagged Objects. In *Proceedings of Ubiquitous Computing 6th International Conference (UbiComp 2004)*, pages 268–282, 2004.

[59] Sengul Vurgun, Matthai Philipose, and Misha Pavel. A statistical reasoning system for medication prompting. In Krumm et al. [157], pages 1–18.

[60] Keng hao Chang, Mike Y. Chen, and John Canny. Tracking free-weight exercises. In Krumm et al. [157], pages 19–37.

[61] Keng hao Chang, Shih yen Liu, Hao-Hua Chu, Jane Yung jen Hsu, Cheryl Chen, Tung yun Lin, Chieh yu Chen, and Polly Huang. The diet-aware dining table: Observing dietary behaviors over a tabletop surface. In Fishkin et al. [156], pages 366–382.

[62] Emmanuel Munguia Tapia, Stephen S. Intille, and Kent Larson. Activity recognition in the home using simple and ubiquitous sensors. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing, Second International Conference, PERVASIVE 2004*, pages 158–175, Vienna, Austria, April 18-23 2004. Springer.

[63] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, and Wendy Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *CoBuild*, volume 1670 of *Lecture Notes in Computer Science*, pages 191–198. Springer, 1999.

[64] Easy Living. See: http://research.microsoft.com/easyliving.

[65] The Adaptive House. See: http://www.cs.colorado.edu/∼mozer/house.

[66] Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artifical Intelligence*, 19(1):17–37, 1982.

[67] Graham Thomson, Paddy Nixon, and Sotirios Terzis. Towards ad-hoc situation determination. In *First International Workshop on Advanced Context*

*Modelling, Reasoning And Management*, Nottingham, England, September 2004. UbiComp 2004, The Sixth International Conference on Ubiquitous Computing.

[68] P. H. Chen, C. J. Lin, and B. Schölkopf. A tutorial on $\nu$-support vector machines. *Applied Stochastic Models in Business and Industry*, 2004.

[69] Martin Mühlenbrock, Oliver Brdiczka, Dave Snowdon, and Jean-Luc Meunier. Learning to detect user activity and availability from a variety of sensor data. In *PerCom* [154], pages 13–22.

[70] Alexander G. Hauptmann, Jiang Gao, Rong Yan, Yanjun Qi, Jie Yang, and Howard D. Wactlar. Automated analysis of nursing home observations. *IEEE Pervasive Computing*, 03(2):15–21, 2004.

[71] N. Oliver, A. Garg, and E. Horvitz. Layered representations for learning and inferring office activity from multiple sensory channels. *Computer Vision and Image Understanding*, 96(2):163–180, 2004.

[72] Fahd Albinali, Nigel Davies, and Adrian Friday. Structural learning of activities from sparse datasets. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07)*, pages 221–228. IEEE Computer Society, 2007.

[73] Stephen S. Intille, Kent Larson, J. S. Beaudin, J. Nawyn, E. Munguia Tapia, and P. Kaushik. A living laboratory for the design and evaluation of ubiquitous computing technologies. In *CHI '05: Extended Abstracts on Human Factors in Computing Systems*, pages 1941–1944, New York, NY, USA, 2005. ACM.

[74] The MIT Placelab Datasets. See: http://architecture.mit.edu/house_n/data/PlaceLab/PlaceLab.htm.

[75] Juan Ye, Lorcan Coyle, Simon Dobson, and Paddy Nixon. Using situation lattices in sensor analysis. In *PerCom*, pages 1–11. IEEE Computer Society, 2009.

[76] M. C. Mozer. Lessons from an adaptive house. In D. Cook and R. Das, editors, *Smart environments: Technologies, protocols, and applications*, pages 273–294. John Wiley & Sons, 2005.

[77] Donald J. Patterson, Lin Liao, Dieter Fox, and Henry A. Kautz. Inferring high-level behavior from low-level sensors. In Dey et al. [155], pages 73–89.

[78] Nirmalya Roy, Abhishek Roy, and Sajal K. Das. Context-Aware Resource Management in Multi-Inhabitant Smart Homes: A Nash H-Learning based Approach. In *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2006)*, pages 148–158. IEEE Computer Society, 2006.

[79] Guanling Chen and David Kotz. Solar: An open platform for context-aware mobile applications. In *Proceedings of the First International Conference on Pervasive Computing (Short paper)*, pages 41–47, June 2002.

[80] Hui Lei, Daby M. Sow, John S. Davis II, Guruduth Banavar, and Maria R. Ebling. The design and applications of a context service. *ACM SIGMOBILE Mobile Computing and Communications*, 6(4), 2002.

[81] Davis, Sow, Blount, and Ebling. Context tailor: Towards a programming model for context-aware computing. In *1st International ACM Workshop on Middleware for Pervasive and Ad-Hoc Computing*. ACM Press, 2003.

[82] Robert Grimm, Janet Davis, Eric Lemar, Adam Macbeth, Steven Swanson, Thomas Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble, and David Wetherall. System support for pervasive applications. *ACM Transactions on Computer Systems (TOCS)*, 22(4):421–486, 2004.

[83] H. Chen, T. Finin, and A. Joshi. A context broker for building smart meeting rooms. In *Proc. Knowledge Representation and Ontology for Autonomous Systems*. AAAI, March 2004.

[84] Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing*, 04(2):51–59, 2005.

[85] Mik Lamming and Denis Bohm. Specx: Another approach to human context and activity sensing research, using tiny peer-to-peer wireless computers. In Dey et al. [155], pages 192–199.

[86] Martin Strohbach, Gerd Kortuem, Hans-Werner Gellersen, and Christian Kray. Using cooperative artefacts as basis for activity recognition. In Panos

Markopoulos, Berry Eggen, Emile H. L. Aarts, and James L. Crowley, editors, *EUSAI*, volume 3295 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2004.

[87] Martin Strohbach, Hans-Werner Gellersen, Gerd Kortuem, and Christian Kray. Cooperative artefacts: Assessing real world situations with embedded technology. In Nigel Davies, Elizabeth D. Mynatt, and Itiro Siio, editors, *Ubicomp*, volume 3205 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2004.

[88] DIY Smart-its Homepage. See: http://ubicomp.lancs.ac.uk/smart-its.

[89] Markus C. Huebscher and Julie A. McCann. An adaptive middleware framework for context-aware applications. *Personal and Ubiquitous Computing*, 10(1):12–20, 2006.

[90] Jalal Al-Muhtadi, Shiva Chetan, Anand Ranganathan, and Roy H. Campbell. Super spaces: A middleware for large-scale pervasive computing environments. In *2nd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2004 Workshops)*, pages 198–202, March 2004.

[91] Alan Dearle, Graham N. C. Kirby, Ronald Morrison, Andrew McCarthy, Kevin Mullen, Yanyan Yang, Richard C. H. Connor, Paula Welen, and Andy Wilson. Architectural support for global smart spaces. In *Proceedings of the 4th International Conference on Mobile Data Management*, pages 153–164. Springer-Verlag, 2003.

[92] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 01(2):22–31, 2002.

[93] Glenn Judd and Peter Steenkiste. Providing contextual information to pervasive computing applications. In *First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, 2003.

[94] Mohan Kumar, Behrooz A. Shirazi, Sajal K. Das, Byung Y. Sung, David Levine, and Mukesh Singhal. Pico: A middleware framework for pervasive computing. *IEEE Pervasive Computing*, 02(3):72–79, 2003.

[95] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010.

[96] Simon Dobson and Juan Ye. Using fibrations for situation identification. In *Pervasive 2006 workshop proceedings*, pages 645–651. Springer Verlag, 2006.

[97] Bertrand Meyer. The many faces of inheritance: A taxonomy of taxonomy. *IEEE Computer*, 29(5):105–108, 1996.

[98] The FuzzyJ Toolkit. See: http://www.nrc-cnrc.gc.ca/eng/projects/iit/fuzzyj-toolkit.html.

[99] Susan McKeever, Juan Ye, Lorcan Coyle, and Simon Dobson. A context quality model to support transparent reasoning with uncertain context. In Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger, and Frank Dürr, editors, *QuaCon*, volume 5786 of *Lecture Notes in Computer Science*, pages 65–75. Springer, 2009.

[100] Thomas Buchholz and Michael Schiffers. Quality of context: What it is and why we need it. In *In Proceedings of the 10th Workshop of the OpenView University Association: OVUA03*, 2003.

[101] Thad E. Starner. Powerful change part 1: Batteries and possible alternatives for the mobile market. *IEEE Pervasive Computing*, 02(4):86–88, 2003.

[102] Rajesh Krishna Balan. Powerful change part 2: Reducing the power demands of mobile devices. *IEEE Pervasive Computing*, 03(2):71–73, 2004.

[103] Marc A Viredaz, Lawrence S Brakmo, and William R Hamburgen. Energy management on handheld devices. *ACM Queue*, 1(7):44–52, 2003.

[104] Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, and Roy Want. Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 220–232, New York, NY, USA, 2006. ACM.

[105] Marc A. Viredaz and Deborah A. Wallach. Power evaluation of a handheld computer. *IEEE Micro*, 23(1):66–74, 2003.

[106] William R. Hamburgen, Deborah A. Wallach, Marc A. Viredaz, Lawrence S. Brakmo, Carl A. Waldspurger, Joel F. Bartlett, Timothy Mann, and Keith I. Farkas. Itsy: Stretching the bounds of mobile computing. *IEEE Computer*, 34(4):28–36, 2001.

[107] Eugene Shih, Paramvir Bahl, and Michael J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 160–171, New York, NY, USA, 2002. ACM.

[108] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European Workshop: Beyond the PC*, pages 87–92, New York, NY, USA, 2002. ACM Press.

[109] Sachin Goyal and John Carter. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *WMCSA '04: Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04)*, pages 186–195, Washington, DC, USA, 2004. IEEE Computer Society.

[110] Nicholas R. Jennings and Michael J. Wooldridge. Applications of intelligent agents. In *Agent Technology: Foundations, Applications, and Markets*, pages 3–28. Springer-Verlag: Heidelberg, Germany, 1998.

[111] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[112] H. Van Dijk, K. Langendoen, and H. Sips. ARC: A Bottom-Up Approach to Negotiated QoS. In *WMCSA '00: Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'00)*, page 128, Washington, DC, USA, 2000. IEEE Computer Society.

[113] Cristina Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A survey of QoS architectures. *Multimedia Systems*, 6(3):138–151, 1998.

[114] Jinshan Liu and Valerie Issarny. QoS-Aware Service Location in Mobile Ad-Hoc Networks. In *IEEE International Conference on Mobile Data Management (MDM'04)*, page 224, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[115] Selim Gurun, Chandra Krintz, and Rich Wolski. NWSLite: A light-weight prediction utility for mobile devices. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 2–11, New York, NY, USA, 2004. ACM Press.

[116] Dushyanth Narayanan and M. Satyanarayanan. Predictive resource management for wearable computing. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 113–128, New York, NY, USA, 2003. ACM Press.

[117] Michael Wooldridge. *Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

[118] JADE - Java Agent DEvelopment Framework. See: http://jade.tilab.com.

[119] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[120] Peter R. Pietzuch and Jean Bacon. Hermes: A distributed event-based middleware architecture. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 611–618, Washington, DC, USA, 2002. IEEE Computer Society.

[121] Susan McKeever, Juan Ye, Lorcan Coyle, and Simon A. Dobson. Using dempster-shafer theory of evidence for situation inference. In Payam M. Barnaghi, Klaus Moessner, Mirko Presser, and Stefan Meissner, editors, *EuroSSC*, volume 5741 of *Lecture Notes in Computer Science*, pages 149–162. Springer, 2009.

[122] IP Mobility Support for IPv4. See: http://www.ietf.org/rfc/rfc3344.txt?number=3344.

[123] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM*

*International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[124] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[125] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[126] B. Y. Zhao, Ling Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

[127] Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components. See: http://www.itu.int/ITU-T/studygroups/com17/oid.html.

[128] OWL Web Ontology Language Overview. See: http://www.w3.org/TR/2004/REC-owl-features-20040210.

[129] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2), 2007.

[130] The FaCT++ OWL-DL Reasoner. See: http://owl.man.ac.uk/factplusplus.

[131] RacerPro: An OWL reasoner and inference server for the Semantic Web. See: http://www.racer-systems.com/index.phtml.

[132] Jena: A semantic web framework for Java. http://jena.sourceforge.net.

[133] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The evolution of Protégé: An environment for knowledge-based systems

development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[134] Cobra-ont. See: http://cobra.umbc.edu/ontologies.html.

[135] Jess, the Rule Engine for the Java Platform. See: http://herzberg.ca.sandia.gov/jess.

[136] Mauro Caporuscio, Antonio Carzaniga, and Alexander L. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. Technical Report CU-CS-944-03, Department of Computer Science, University of Colorado, January 2003.

[137] An Introduction to the Service Location Protocol (SLP). See: http://openslp.org/doc/html/IntroductionToSLP/index.html.

[138] OpenSLP. See: http://openslp.org.

[139] jSLP. See: http://jslp.sourceforge.net.

[140] FreePastry. See: http://freepastry.rice.edu.

[141] Google Calendar APIs. See: http://code.google.com/apis/calendar.

[142] File watchers utilities. See: http://fwutilities.sourceforge.net.

[143] Tim Kindberg and Armando Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81, 2002.

[144] JADE Performance Tests. See: http://jade.tilab.com/papers-2004.htm.

[145] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS VIII)*, pages 75–80, Schloss Elmau, Germany, May 2001.

[146] Marc Langheinrich. Privacy by design – principles of privacy-aware ubiquitous systems. In Gregory D. Abowd, Barry Brumitt, and Steven A. Shafer, editors, *Proceedings of the Third International Conference on Ubiquitous Computing (UbiComp 2001)*, number 2201 in LNCS, pages 273–291, Atlanta, USA, 2001. Springer-Verlag.

[147] Marc Langheinrich. A privacy awareness system for ubiquitous computing environments. In Gaetano Borriello and Lars Erik Holmquist, editors, *4th International Conference on Ubiquitous Computing (Ubicomp 2002)*, number 2498 in LNCS, pages 237–245. Springer-Verlag, September 2002.

[148] See5/C5.0. See: http://www.rulequest.com/see5-info.html.

[149] .NET Class Libraries. See: http://msdn2.microsoft.com/en-gb/library/aa388745.aspx.

[150] Keyboard hooks. See: http://www.codeproject.com/dll/keyboardhook.asp.

[151] Mousey! Roll Over and Park: Controlling the mouse using Windows hooks. See: http://www.codeproject.com/dll/ParkMouse.asp.

[152] Pocket PC Developer Network: Changing the sound volume from a program. See: http://www.pocketpcdn.com/articles/soundvolume.html.

[153] How to Use SystemParametersInfo API for Control Panel Settings. See: http://support.microsoft.com/kb/97142.

[154] *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA*. IEEE Computer Society, 2004.

[155] Anind K. Dey, Albrecht Schmidt, and Joseph F. McCarthy, editors. *UbiComp 2003: Ubiquitous Computing, 5th International Conference, Seattle, WA, USA, October 12-15, 2003, Proceedings*, volume 2864 of *Lecture Notes in Computer Science*. Springer, 2003.

[156] Kenneth P. Fishkin, Bernt Schiele, Paddy Nixon, and Aaron J. Quigley, editors. *Pervasive Computing, 4th International Conference, PERVASIVE 2006, Dublin, Ireland, May 7-10, 2006, Proceedings*, volume 3968 of *Lecture Notes in Computer Science*. Springer, 2006.

[157] John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang, editors. *UbiComp 2007: Ubiquitous Computing, 9th International Conference, UbiComp 2007, Innsbruck, Austria, September 16-19, 2007, Proceedings*, volume 4717 of *Lecture Notes in Computer Science*. Springer, 2007.