A Model and Architecture for Situation Determination

Graham Thomson, Sotirios Terzis
and Paddy Nixon

Pervasive and Global Computing Group
University of Strathclyde
Glasgow, UK

Abstract

Automatically determining the situation of an ad-hoc group of people and devices within a smart environment is a significant challenge in pervasive computing systems. Current approaches often rely on an environment expert to correlate the situations that occur with the available sensor data, while other machine learning based approaches require long training periods before the system can be used. Furthermore, situations are commonly recognised at a low-level of granularity, which limits the scope of situation-aware applications. This paper presents a novel approach to situation determination that attempts to overcome these issues by providing a reusable library of general situation specifications that can be easily extended to create new specific situations, and immediately deployed without the need of an environment expert. A proposed architecture of an accompanying situation determination middleware is provided, as well as an analysis of a prototype implementation.

1 Introduction

Automatically determining the situation of an ad-hoc group of people and devices within a smart environment is a significant challenge in pervasive computing systems. Situation identification provides essential context information used by situation-aware applications to influence their operation, silently and automatically adapting the computing machinery contained within an environment to its inhabitants' behaviours.

Current approaches to situation determination can be broadly categorised as either specification based, where the situations are described by a specification of the events that occur, and learning based, where sensor readings are automatically correlated to a set of situations. For specification-based approaches such as [6, 8], an expert of the local environment is required to specify the correlation of the available sensor data with the situations that occur, often in an ad-hoc manner. As the amount of available sensor data and number of situations increases, it becomes increasingly difficult for an expert to decipher and specify correlations. With learning-based approaches such as [21, 18] a training period must be conducted, during which several examples of each situation are collected and analysed, before the system can be used. These factors impede swift adaptation to the evolving set of situations that will occur in an environment over time.

Situations are commonly recognised at a coarse level of granularity, which limits the scope of situation-aware applications. For example, in [21, 18] only a general 'meeting' situation may be recognised, which prevents applications from tailoring their behaviour to the many different types of meeting that a user may attend. Furthermore, at this level of granularity we are limited to determining whether or not a person or device is involved in a situation. This prevents applications from tailoring their behaviour to the role a person or device is playing within a situation, such as whether a user is a speaker or an audience member in a presentation.

In this paper, we present a novel specification-based approach to situation determination that attempts to overcome these issues. The essence of our approach is that situations are viewed as a collection of roles, where a role is a unit of recognition of a situation based on the observable properties

of people and devices in the environment. The properties are identified with common names defined in a standard ontology.

A standard library of situation specifications can then be provided. Situations from the library can be deployed immediately in an environment without the need for an environment expert. These situations enable various levels of granularity, including 'group meeting' and 'PhD progress meeting' in addition to a 'meeting' situation, as well as recognition of the distinct role a person or device is playing within the situation. New situations particular to an environment can be created as simple variations of those in the library. We also provide the ability for users to customise situation specifications to their particular habits, such as a particular supervisor only holds PhD progress meetings in his office. Furthermore, the roles and situations defined in the library can be re-used by application developers to construct new situation specifications by assembling these high-level components, rather than specifying new situation specifications from scratch.

The structure of this paper is as follows: Section 2 presents our approach to situation specification, Section 3 describes how specifications are extended to incorporate uncertainty, Section 4 proposes an architecture for situation determination and a description and analysis of a prototype implementation are given in Section 5, Section 6 surveys related work and Section 7 presents conclusions and future work.

2 Situation specifications

In our approach, the situation refers to the activity a single person or a group of people are conducting. A situation is characterised by the properties of the people involved in the situation and the properties of the tools, or devices, they are using.

A role is the basic building block of a situation specification, and describes a part of the overall situation we wish to recognise. A role contains a set of Boolean expressions based on the observable properties of people and devices. All of the expressions in the role hold when the part of a situation it describes is occurring in the environment. A full situation specification can be built up by assembling a collection of roles.

Location information is commonly regarded as essential for describing situations [11]. A location property is defined for people and devices. Our approach requires that an underlying location infrastructure is available and can provide the distance between two objects, and the symbolic coordinates of the location of an object. Example symbolic coordinates include 'Room L10.01' and 'Ric's desk'. Both of these primitives are commonly supported by location systems [7]. In addition to this, we require location types of symbolic coordinates, similar to those employed by Look et al. [17]. These types indicate the category or function of the location. For example, symbolic coordinate 'Room L10.01' may have types 'Meeting area' and 'Room', while 'Ric's desk' would have the type 'Desk area'.

Two properties are defined for roles themselves. These are a timestamp, that indicates the time at which the role started to occur in the environment, and cardinality, which indicates how many occurrences of the role are happening simultaneously in the environment.

Expressions within a role may refer to the properties of people and devices. They may include the standard comparison operators, the Boolean operators $\neg$, $\wedge$, $\vee$, and $\Rightarrow$, as well as other type specific operators.

A situation specification is similar in structure to a role. Its expressions are based on a collection of roles and may refer to their timestamp and cardinality properties as well as the properties of the people and devices they specify, and also the current time. The expressions may include the same set of operators as a role. All of the expressions in the specification hold when the situation is occurring in the environment.

An example specification of a presentation situation is given in Fig. 1. Three roles are defined - speaker, audience member, and presentation equipment. Each role lists the entities its expressions refer to, where an entity is a person, a device, or another role. The speaker role is played by a person who is located in a speaker area. The audience member role is played by a person who is located in the audience area.

A computer is playing the presentation equipment role when it is running presentation software as the active application. A presentation is occurring when a speaker, three or more audience members, and one or more computers that have presentation software active are in the same room.

2.1 Specification inheritance

A situation can be expressed at different levels of abstraction through specification inheritance. This provides a simple way to create new specifications as refinements of another, and to allow situation-aware applications to interpret the same situation at the appropriate level of abstraction.

Figure 2 provides an example of specification inheritance. Three situations are described at increasing levels of abstraction. These are SmartLab group meeting, group meeting, and a general meeting situation. A meeting is occurring when two or more people are gathered in the same meeting area. A group meeting is occurring when three or more people are gathered in the same meeting area and at least 70% of them are members of the same group. A SmartLab group meeting is occurring when a group meeting is occurring for the group 'SmartLab'.

When a specification inherits from another, all of the expressions from both specifications must hold when the role or situation is occurring. For example, when a group meeting is occurring, all of the expressions in the group meeting specification and the meeting specification will hold. Furthermore, the expressions of the role or situation may refer to the entities from either specification. For example, the SmartLab group meeting specification can refer to the entities of the group meeting specification.

2.2 Customisations

Existing situations can be refined to a particular environment or person using specification customisation. Figure 3 provides an example of this. A PhD meeting situation is defined that occurs when a PhD student and one or more of their supervisors are gathered in the same meeting area. In addition, a customisation is defined that states that a particular PhD supervisor John only holds PhD meetings in his office L10.01.

When a specification is customised, all of the expressions from the role or situation specification and the customisation specification must hold when the role or situation is occurring. For example, when the John PhD meeting customisation has been defined and a PhD meeting is occurring, all of the expressions in the PhD meeting specification and the customisation specification will hold. Furthermore, the expressions of the customisation may refer to the entities from either specification. For example, the John PhD meeting customisation can refer to the entities of the PhD meeting specification.

2.3 Specification resolution

The resolution of a situation reflects the level of detail to which we can tell that a person or a device is involved in that situation. For example, at a low resolution we may only be able to report whether a person or a device is involved in a situation or not. At a higher resolution we may be able to report which role they are playing. In the presentation example in Fig. 1, we can tell that in addition to being an attendee of a presentation, either a person is a speaker or an audience member. At a higher resolution still, we may report that a person or device is playing a more specific role, for example, different types of audience member may be defined.

The specification in Fig. 1 is of high resolution, in that it reports whether a person is a speaker or an audience member. Let's consider the speaker role in more detail. A person is playing the role of a speaker if they are located within the speaker area. To detect this role, a person's location reported at room level resolution is not sufficient, we require the finer resolution of whether they are within the speaker area of the room. Alternatively, a speaker could be identified by detecting the fact that they have been speaking for most of the time. However, in this case we require high-level context information inferred from basic audio data. In order to deal more appropriately with either the lack of high-resolution context information or high-

```
role: speaker
entities: p:Person
expressions:
  p.location has type ‘Speaker area’

role: audience member
entities: p:Person
expressions:
  p.location has type ‘Audience area’

role: presentation equipment
entities: c:Computer
expressions:
  presentation software is active
    application on c
```

```
situation: presentation
roles:
  s:speaker
  a:audience member
  e:presentation equipment
expressions:
  s.cardinality = 1
  a.cardinality >= 3
  e.cardinality >= 1
  ‘Room’ of s.p.location = ‘Room’ of a.p.location
  ‘Room’ of s.p.location = ‘Room’ of e.c.location
```

Figure 1: A presentation specification.

```
role: meeting attendee
entities: p:Person
expressions:
  p.location has type ‘Meeting area’

situation: meeting
roles: m:meeting attendee
expressions:
  m.cardinality >= 2
  ‘Meeting area’ of m.p.location =
    ‘Meeting area’ of m.p.location

role: group member attendee
inherits: meeting attendee
entities: g:Group
expressions: p is a member of g
```

```
situation: group meeting
inherits: meeting
roles:
  gma:group member attendee
  ma:meeting attendee
expressions:
  gma.cardinality >= 3, ma.cardinality >= 0
  gma.g = gma.g, gma.p.id != ma.p.id
  gma.cardinality >= 0.7 *
    (gma.cardinality + ma.cardinality)
  ‘Meeting area’ of gma.p.location =
    ‘Meeting area’ of ma.p.location

situation: SmartLab group meeting
inherits: group meeting
expressions: gma.g = ‘SmartLab’
```

Figure 2: Various meeting situations defined using specification inheritance.

```
role: PhD supervisor
inherits: meeting attendee
entities: phd:Person
expressions: p supervises phd

role: PhD student
inherits: meeting attendee
expressions: p is a PhD student

customisation: John PhD meeting
customises: PhD meeting
expressions:
  sup.p.id = JOHN_ID ->
    ‘Room’ of sup.p.location = L10.01
```

```
situation: PhD meeting
inherits: meeting
roles:
  stu:PhD student
  sup:PhD supervisor
expressions:
  stu.cardinality = 1
  sup.cardinality >= 1
  stu.p.id = sup.phd.id
  ‘Meeting area’ of stu.p.location =
    ‘Meeting area’ of sup.p.location
```

Figure 3: An example customisation of a PhD meeting specification.

level inference capability we allow alternative situation specifications of varying resolution.

Example high and low resolution alternative presentation specifications are given in Fig. 4. The high resolution presentation specification requires that 'Speaker area' and 'Audience area' locations are available. The low resolution specification requires only that room level locations are available. When both versions of the presentation situation are detected, the system will prompt the user for their preference of which version is reported and remember their decision for future use.

2.4 Overlapping situations

In the specifications we have looked at so far, all have assumed that situations are independent. It is desirable to make situations independent so that 'smaller' situations can still be recognised within 'larger' situations. For example, a group of eight people may be recognised as being involved in a group meeting situation, while at the same time, three of these people are involved in a 'working at a whiteboard' situation.

However, there are cases where we may wish to avoid situations overlapping. Given the group meeting specification from Fig. 2 and the PhD meeting specification from Fig. 3, when a group meeting is occurring where the four group members present consist of two sets of a PhD student and one of their supervisors, it is impossible to tell whether it is a single group meeting that is occurring, or two PhD meetings. This confusion has arisen because the PhD meeting specification is incomplete. What is actually meant is that a PhD meeting is occurring when *only* a PhD student and one or more of their supervisors are gathered in a meeting area. To incorporate this into the specification, a lack of another person can be expressed as a 'someone else' role that has a cardinality of 0. In other cases where overlap should be avoided, situation specifications can be differentiated through user customisation of a specification with environment-specific details. In order to assist the user in this task we require that the system is able to explain why situations occurred, for example, which people and devices where considered to be in which roles.

3 Incorporating uncertainty

So far, we have only considered specifying situations using properties and expressions that have crisp Boolean values. In a pervasive environment, many properties shall be captured using sensors, which may be limited in their accuracy and reliability. This will affect the level of confidence we can have 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, we must interpret their level of confidence 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.

Fuzzy logic provides an appropriate framework to incorporate and combine the confidence values of properties in a situation specification. An alternative approach to incorporating uncertainty is to use probabilistic techniques such as Bayesian networks [21, 18, 19]. In this work, we have chosen to employ fuzzy logic over probabilistic techniques for the following practical reasons:

Ease of translation As we will demonstrate 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. Our attempts to create a probabilistic representation of situation specifications have resulted in Bayesian networks whose topology

```
role: speaker
entities: p:Person
expressions:
  p.location has type 'Speaker area'

role: audience member
entities: p:Person
expressions:
  p.location has type 'Audience area'

role: presentation equipment
entities: c:Computer
expressions:
  presentation software is active
    application on c

role: presentation attendee
entities: p:Person
expressions:
  p.location has type 'Meeting area'
```

```
situation: presentation
roles:
  s:speaker, a:audience member
  e:presentation equipment
expressions:
  s.cardinality = 1, a.cardinality >= 3
  e.cardinality >= 1
  'Room' of s.p.location =
    'Room' of a.p.location
  'Room' of s.p.location =
    'Room' of e.c.location

situation: presentation
roles:
  a:presentation attendee
  e:presentation equipment
expressions:
  a.cardinality >= 4
  e.cardinality >= 1
  'Room' of a.p.location =
    'Room' of e.c.location
```

Figure 4: Alternative presentation specifications at high (top) and low (bottom) resolution.

must be restructured at runtime to incorporate the varying number of people and device 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 [12]. Rete-based fuzzy rule engines have mature tool support and are freely available [3].

Improved scalability 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. Let us assume that a party situation is defined as twenty or more people gathered in a specific room. Furthermore, let us 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 ≈ 0.04. As the number of people increases, the maximum probability we can have 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 we could have 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.

To illustrate how fuzzy logic can be incorporated, consider again the presentation example in Fig. 1. A person is 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 [9].

If we take ec to be the confidence that person p's location contains the type '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', we can construct the fuzzy rule:

```
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 Fig. 5. The confidence that person p's location contains the type 'Speaker area' is 0.9. The results 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 role in Fig. 1 that have several expressions will be represented by a fuzzy rule that has several antecedents. Therefore, we must 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 expressions in the presentation specification will be combined by fuzzy intersection. Expressions that do involve uncertainty, such as the cardinality expressions, are treated as having a confidence value of 1 when true and 0 when false. Figure 6 illustrates how the confidence values of the presentation specification are combined.

```
situation: presentation
roles:
  s:speaker
  a:audience member
  e:presentation equipment
expressions:
  s.cardinality = 1, s.confidence >= 0.8
  a.cardinality >= 3, a.confidence >= 0.8
  e.cardinality >= 1
  'Room' of s.p.location = 'Room' of a.p.location
  'Room' of s.p.location = 'Room' of e.c.location
```

Figure 7: The presentation specification including expressions concerning confidence.

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 $\geq t$".

As an example, the presentation role shown in Fig. 1 is restated in Fig. 7 including the necessary additional expressions required to incorporate reasoning with confidence. For this example, we can see that the only additional expressions required are those to specifying the desired confidence threshold for the speaker and audience member roles. All other mechanisms for incorporating uncertainty are handled automatically and do not burden the specification author.

Taken together, the constructs described in this and the previous sections provide a sufficiently expressive language to define situation specifications for ad-hoc groups of people and devices. Complex situation specifications can be simply assembled as a collection of roles. Constraints can be placed on a role's cardinality, making it easy to specify groups of people and devices that are involved in a situation. New situations can be easily created as a refinement of another using specification inheritance. Existing situations can be customised to

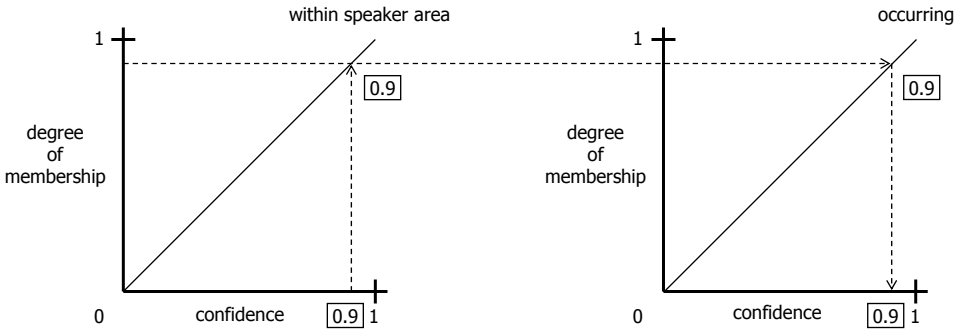

Figure 5: An example of determining the confidence value of a consequent fuzzy set using monotonic selection.

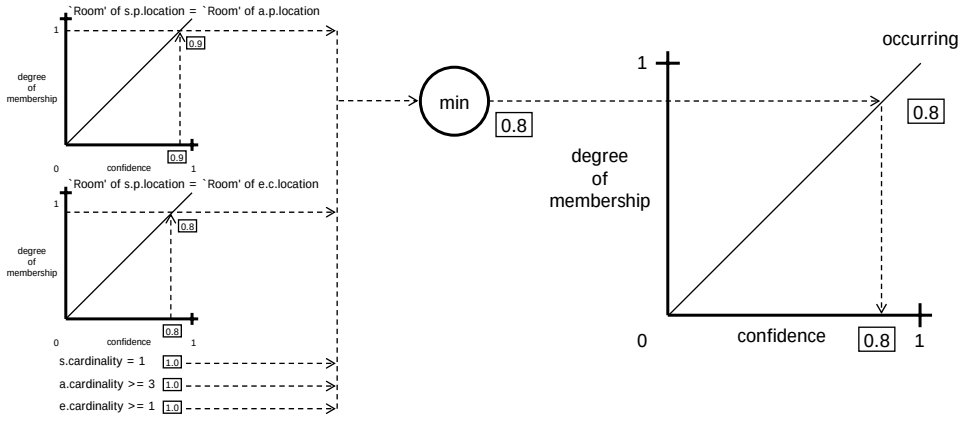

Figure 6: An example of combining confidence values.

a specific environment or the habits of a particular person in a straightforward manner. High-resolution situations can be supported that in addition to identifying which situation a person or device is involved in, identify which role the person or device is playing within the situation. Moreover, alternative specifications can be described such that situation recognition can adapt to the information available in the environment. Uncertainty associated with sensed properties can incorporated naturally into a situation specification, requiring only a few additional expressions.

4 A situation determination architecture

A situation determination system has several distinct characteristics that must be supported by an 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 the most appropriate [15].

Our architecture is designed to support situation determination within a small physically bounded space, such as a single room or the rooms within a floor of a building. The architecture is illustrated in Fig. 8. The following types of agents are defined:

An area server agent (ASA) An area server agent performs situation determination for all of the people and devices within a bounded physical space, such a room. It runs on a dedicated server. The ASA will have knowledge of all library situation specifications, as well as any additional specifications and customisations particular to the space it governs.

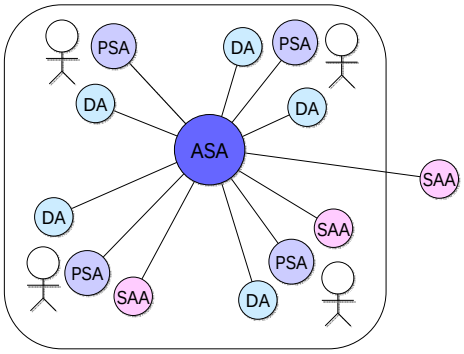

Figure 8: An example deployment showing the architecture of the situation determination system. The rounded rectangle represents the physical are the ASA governs.

A personal server agent (PSA) This agent represents a person. Each person is assumed to wear or carry a device that hosts this agent. Typically this device would be a PDA or a mobile phone. A PSA will have knowledge of the person's properties, as well as any situation specifications and customisations particular to the person.

A device agent (DA) This agent represents a device and has knowledge of the device's properties. For devices with sufficient capability, the DA is hosted on the device itself. For devices with limited resources, the DA will be hosted on the area server or another appropriate device, and act as a proxy. A general DA is defined that provides facilities to communicate with the agent infrastructure that are common to all DAs. The code required to access properties of a device must often be customised to a specific device or platform. Therefore, device and platform specific DAs are defined that extend the general DA that can access the properties of the device. A DA does not carry any additional specifications and customisations.

A situation-aware application agent (SAA) This agent represents an application that uses situation information to influence its operation. It allows applications to communicate with the ASA to request and receive notifications about occurring situations, which may include additional specifications and customi-

sations that are particular to the application. A SAA may run on any appropriate device.

Agents can discover and communicate with each other via an agent platform substrate. In our architecture, agents can connect to the agent platform either through a wired or wireless network. The agent platform is advertised on both networks using a well-known name and is discovered through an ad-hoc network discovery protocol. Within the agent platform, agents may be discovered by their identity (white-page look up) or by the services they provide (yellow-page look up).

Once connected to the agent platform, a PSA discovers the ASA using yellow-pages lookup. It will then carry out the following conversation with the ASA: 1) the PSA sends a message to the ASA identifying which type of agent it is, as well as the description of any additional or customised specifications it has, 2) upon receiving this message, the ASA adds any new specifications to the active specification set, and then analyses this set to discover which roles of that agent type it requires to be informed of, and sends these back in a reply to the PSA, 3) the PSA analyses this set of roles to determine which of them it can support, given the set of properties it has available and their supported level of resolution, 4) the PSA then monitors the supported roles and sends a message to the ASA when they hold or cease to hold. If the set of roles required by the ASA changes, for example in response to new or customised situations being introduced, the ASA sends the PSA a message informing it of these changes and the PSA filters these and updates its monitoring appropriately.

Both a DA and SAA may connect to the agent platform through either the wired or wireless network depending on whether they are hosted on a fixed or mobile device. In both cases, the ASA is again discovered using yellow-pages look up. When a DA discovers the ASA, it will conduct a conversation similar to that of the PSA, with the exception that it shall not send any additional or customised specifications.

When a SAA discovers the ASA, it carries out the following conversation: 1) the SAA sends a message to the ASA informing it of the situations it wants to be notified about, 2) the ASA then monitors the situations of interest and sends a message to the SAA when they occur and when they cease to occur. If the situations that the SAA is interested in change, the SAA sends a message to the ASA informing it of the changes and the ASA updates its monitoring appropriately.

The ASA supports an 'explain' request that may be issued by other agents in the system. In response, the ASA sends a message containing the specifications, customisations, and values of properties of the situations that it believes are currently occurring. The request may specify a filter on the situations for which an explanation is required, such as only the situations a particular person is involved in, or only the situations that are occurring in a particular room. This information is not only useful for debugging the system, but also helps to identify suitable properties that can be used to customise and refine situation specifications to a particular environment.

The architecture's star topology offers the following advantages: a) redundant determination effort is eliminated as the situations for all of the people and devices in the environment is performed once, b) all of the customised specifications from each PSA can be combined to give greater situation recognition accuracy, c) the ASA is likely to be more powerful than a PSA and so can perform the determination more quickly, and d) it reduces the drain on the battery power of each PSA's mobile host. Given that an ASA is hosted on powerful computer and governs a small physical space, we consider these advantages to outweigh the typical disadvantages of a centralised architecture, where the ASA is a single point of failure and may be a communication bottleneck. In cases where the physical area is large, or when only limited computing power is available, a hierarchical deployment of ASAs can be used. In this deployment, the physical space is divided into smaller sub-areas, each with its own ASA. The situation determination process is then coordinated between all ASAs. We recognise this as an area of future work.

The architecture presented in this section facilitates situation determination for a large number of situations, people and devices, while defining only a small number of agents with a

simple set of behaviours.

5 Prototype implementation and analysis

We have constructed a prototype implementation of our situation determination system and an initial test application. We developed the person and area server agents and the general device and situation-aware application agents. Two extended device agents were also developed, one for desktop or laptop computers running Windows, and one for Pocket PCs running Windows Mobile. The person agent reported location, identity, group membership, and occupation properties for a person. The device agents were able to report which applications were running on the device, the currently active application, and the person using the device. Location information was self-reported by the user. These properties were chosen as they do not involve uncertainty which simplified the development of the initial test implementation, and formed an adequate basis for a sufficiently interesting and complex set of situations.

The following situations were defined: checking e-mail, surfing the web, reading, and coding which extended a general 'using an application' specification, and group meeting, PhD meeting, and demonstrators' meeting which extended a general 'meeting' specification.

Recognition of situations is performed by the ASA using the JESS rule engine [5]. Situation specifications are translated offline into a set of JESS rules. The ASA maintains a record of all the situations that the people and devices within its area are involved in. SAAs can then be notified immediately of the situations they request. Our implementation of the architecture is based on the JADE agent framework [4]. JADE provides all of the standard agent functionality required by our system.

As an initial test of the system, we developed an availability checker application. This presents a list of the current situations for a particular person who the user selected from a drop-down list. The application was simple to write, requiring only a single agent class and GUI to be written. The agent itself was very simple, it extended the SAA agent configuring it such that a single, immediate update of all the current situations was received. Each of the situations listed above were identified accurately by the availability checker.

The code footprint of the person agent and the availability checker application is small and can be accommodated comfortably on a mobile device. The total size of a PSA is under 7KB, and the availability checker application is under 6KB for both the agent and GUI. The average size of the situation specifications used in this analysis requires approximately 2KB of JESS code.

To test the responsiveness of our system a special PSA was created that received updates of its own situations. The time was measured from when the PSA sent a message to the ASA with an update that changed the situation of the PSA's representative, until the PSA received the message informing it of the change to its situation. The PSA was hosted on a HP Pocket PC h5500 with a 400 MHz Intel XScale processor and 128 MB RAM. The ASA was hosted on a desktop PC with a 1.8GHz Intel Pentium 4 processor and 512 MB RAM. The PSA and ASA were connected via a WEP encrypted 802.11 wireless network. The results are given in Fig. 9 and show that the round trip situation update time increases proportionally to the number of active situations and PSAs.

We also measured the average cumulative CPU time for a single situation update on the ASA over all nine situations with eight PSAs. JESS, that updates the record of which situations are occurring, consumed 38.66% CPU time, JADE, which sends, receives, and decodes agent messages, consumed 58.06% CPU time, and the custom situation determination code, that inserts updates into the JESS engine, consumed 3.28% CPU time.

The complexity of a Rete-based rule engine such as JESS 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 LHS [12]. From this, we calculated the complexity of an active situation set. Each person and device is represented by a single fact. Each role produces an activation and deactivation rule. The number of patterns in the rule LHS will equal the number of expressions in

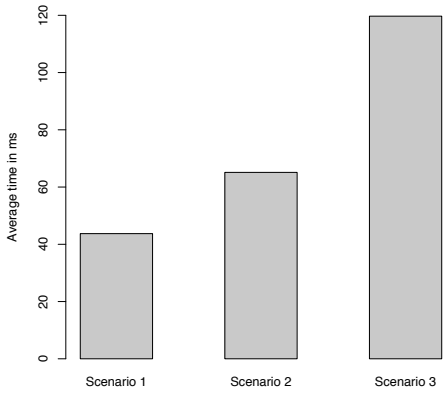

Figure 9: This figure shows the round trip situation update time between a single PSA and ASA. In scenario 1, the active situation set of the ASA contained only a single situation and only one PSA sending updates. In scenario 2, the active situation set contained all nine situations and eight PSA sending updates. In scenario 3, the active situation set contained all nine situations and sixty-four PSA sending updates. The average round trip situation update time is averaged over 100 situation changes.

the role. To keep track of cardinalities, an additional bookkeeping fact is required for each role, as well as an additional pattern in its LHS of the activation and deactivation rules. If we take R to be the number of roles, P to be the number of people and devices, and E to be the average number of expressions per role including cardinality bookkeeping, the complexity is $O(2R(P+R)E)$, or $O(2R^2E + 2RPE)$ which is quadratic on the number of roles.

As described in the previous section, this complexity can be shared amongst several ASAs in a hierarchical deployment. For example, if the load on the host of an ASA whose range is a floor of a building becomes too great, an ASA can be assigned to each room of the floor. For an area that experiences a sudden influx of a large number of people, the ASA could become a communication bottleneck. However, in a typical office environment for which this system has been designed, it is not expected that such a large, sudden influx of people will occur.

6 Related work

Several recent projects have shown success in automatically identifying fine-grained activities of people. Philipose et al. presented an approach where activities such as holding a telephone handset and adjusting a thermostat were recognised by tagging household objects with radio frequency identification (RFID) tags which were detected by a small RFID reader worn on the wrist when in close proximity [20]. In [19], Patterson described a system that inferred whether a person was travelling by bus, by foot, or by car, based on a series of GPS coordinates. Both of these approaches are based on Bayesian networks and the problems involved in using this representation to recognise the situation of ad-hoc group of people and devices have been described in Section 3. However, these projects are complementary to our work and may be used to provide properties of people and devices from which situations are inferred.

In [13, 14], Henricksen et al. presented a graphical context modelling framework that could be used to specify context information

requirements for context-aware applications. Dey et al. described a toolkit that supported the acquisition of low-level context information from sensor data that could be combined via a set of 'widgets' to form higher-level contexts [10, 6]. In both of these approaches a person's activity is provided as an atomic piece of context information rather than automatically inferred from other context. Moreover, our approach focuses on the provision of a reusable library of general high-level situation specifications that can be deployed and customised by a non-expert, rather than the low-level context requirements of a specific context-aware application.

A strand of the GAIA project [21] enabled recognition of low-resolution situations within a specific 'active space' by training a Bayesian network classifier on a fixed set of properties of the space. In [18], Oliver et al. presented a system that recognised low-resolution situations within a specific office environment. A layered approach was used in which a hidden Markov model (HMM) was trained on each category of sensor within the environment, e.g. audio sensors and video sensors, which then formed the input to a higher-layer HMM that recognised the situation. In both these cases, the situations are tightly bound to the static set of properties of the particular environment and are therefore not transferable to other environments. Other smart space projects include Georgia Tech's Aware Home [16], the University of Colorado at Boulder's Adaptive House [1] and Microsoft's EasyLiving [2]. Our approach seeks to provide transferable situation specifications for an ad-hoc group of people and devices, rather than those of a particular space, without requiring an explicit training phase.

7 Conclusions and future work

In this paper, we have presented a novel approach to situation determination based upon a reusable library of situation specifications that can be deployed immediately by non-expert users. Situation specifications may be extended and customised to recognise fine-granularity situations of particular people and environments. We have also presented a supporting agent-based architecture and an initial prototype implementation. Preliminary experimentation and analysis demonstrated that our approach can accurately identify situations for ad-hoc group of people and devices with sufficient responsiveness for a large number of people, devices, and situations.

An extended evaluation of the current architecture which incorporates uncertainty and fuzzy reasoning is currently underway. Moreover, a fuller application-based evaluation of the system is planned, with the development of a mode-manager application in which the mode of operation of a device is automatically set to that most appropriate for its current situation, and a situation-enhanced file management application that allows users to search for files indexed by the situations in which they were used. Furthermore, we intend to continue our evaluation of the middleware onto a larger deployment, covering an extended set of situations and types of device agent.

About the Authors

Graham Thomson is a PhD student in the Department of Computer and Information Sciences at the University of Strathclyde. He received his BSc in Software Engineering from the University of Strathclyde. Contact him at the Univ. of Strathclyde, Dept. of Computer and Information Science, Livingstone Tower, 26, Richmond Street, G1 1XH Glasgow, Scotland; Graham.Thomson@cis.strath.ac.uk.

Sotirios Terzis is a lecturer in the Department of Computer and Information Sciences at the University of Strathclyde. He received his PhD in the Computer Science Department at Trinity College Dublin. His research interests include context-awareness and trust management in pervasive computing systems. He received his BSc and MSc with a specialization in distributed systems from the University of Crete. He is a member of the ACM, the IEEE Computer Society, and the British Computer Society. Contact him at the Univ. of Strathclyde, Dept. of Computer and Information Science, Livingstone Tower, 26,

Richmond Street, G1 1XH Glasgow, Scotland; Sotirios.Terzis@cis.strath.ac.uk.

Paddy Nixon Paddy Nixon is the Professor of Distributed Systems and head of the Systems Research Group (SRG) in the Department of Computer Science in the University College Dublin Ireland. Contact him at Paddy.Nixon@ucd.ie.

References

[1] The adaptive house. See: http://www.cs.colorado.edu/m̃ozer/house/.

[2] Easy living. See: http://research.microsoft.com/easyliving/.

[3] Fuzzyj toolkit. See: http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit2.html.

[4] Jade - java agent development framework. See: http://jade.tilab.com/.

[5] Jess, the rule engine for the java platform. See: http://herzberg.ca.sandia.gov/jess/.

[6] 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.

[7] Christian Becker and Frank Durr. On location models for ubiquitous computing. *Personal Ubiquitous Comput.*, 9(1):20–31, 2005.

[8] H. Chen, T. Finin, and A. Joshi. A context broker for building smart meeting rooms. In *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium*. AAAI, March 2004.

[9] 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.

[10] Anind K. Dey and Gregory D. Abowd. Cybreminder: A context-aware system for supporting reminders. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 172–186. Springer-Verlag, 2000.

[11] 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.

[12] Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artif. Intell.*, 19(1):17–37, 1982.

[13] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *PerCom*, pages 77–86. IEEE Computer Society, 2004.

[14] Jadwiga Indulska, Karen Henricksen, Ted McFadden, and Peter Mascaro. Towards a common context model for virtual community applications. In *2nd International Conference on Smart Homes and Health Telematics (ICOST)*, 2004.

[15] 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.

[16] 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.

[17] Gary Look, Buddhika Kottahachchi, Robert Laddaga, and Howard Shrobe. A location representation for generating descriptive walking directions. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 122–129, New York, NY, USA, 2005. ACM Press.

[18] Nuria Oliver, Ashutosh Garg, and Eric Horvitz. Layered representations for learning and inferring office activity from multiple sensory channels. *Comput. Vis. Image Underst.*, 96(2):163–180, 2004.

[19] Donald J. Patterson, Lin Liao, Dieter Fox, and Henry A. Kautz. Inferring high-level behavior from low-level sensors. In Anind K. Dey, Albrecht Schmidt, and Joseph F. McCarthy, editors, *Ubicomp*, volume 2864 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2003.

[20] 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.

[21] 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.