Situation Determination with Reusable Situation Specifications

Graham Thomson, Sotirios Terzis, and Paddy Nixon

Pervasive and Global Computing Group
Department of Computer and Information Sciences
University of Strathclyde
Livingstone Tower
Glasgow, UK, G1 1XH
E-mail: *firstname.lastname*@cis.strath.ac.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. In both cases, the situations are tailored to the specific environment, and are therefore not transferable to other environments. Furthermore, situations are 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.

Identifying the situation provides an essential context used by situation-aware applications to influence their operation, silently and automatically adapting the computing machinery contained within an environment to its inhabitants' behaviours, "invisibly enhancing the world" [1].

Current approaches to situation determination can be broadly categorised into the two following categories. Specification based approaches, where the situations are described by a specification of the events that occur within the situation, and learning based approaches, where sensor readings are automatically correlated to a set of situations by analysing many examples of each situation.

Specification-based approaches such as [2–4] experience the following drawbacks:

– 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.
– The specifications will suffer from the subjective bias of the expert who programmed them.
– Recognition is limited to the fixed number of cases programmed by the expert for the local environment.

Learning-based approaches such as [5, 6] experience the following drawbacks:

– A training period must be performed during which several examples of each situation that occurs is collected and analysed, before the situation determination system can be used.
– Due to a lack of a sufficient number of examples, it may be difficult, or even impossible to learn exceptional situations.
– The correlation of sensor data to situation is performed automatically. Therefore, there are no guarantees that the model produced provides meaningful information about the details of the events that occur within the situation.
– Introducing new situations or adapting existing ones may be expensive, as a sufficient number of examples will have to be collected.

The following drawbacks apply to both categories:

– Recognition of a situation does not adapt well to variation in the situation, that is, when the structure of the situation is uncertain.
– Each situation has often to be specified or learned from scratch.
– The situation descriptions are tailored to the specific environment, and are therefore not transferable to other environments.
– Situations are recognised at a low-level of granularity, which limits the scope of situation-aware applications.

In this paper, we present a novel specification-based approach to situation determination that attempts to overcome these issues.

Our approach is based on the key observation that a situation can be viewed as a collection of roles that are being played by the people and artefacts in the environment.

The situation specification is broken down into each of the roles it contains. A role specifies a smaller, meaningful part of the situation, providing a unit of recognition. These units group together sets of related sensor data. They can then be assembled to form more complex roles, and complete situation specifications. This divide and conquer approach helps alleviate the difficulty of correlating a large set of sensor to a large set of situations.

Roles specify expressions based on the properties of people and artefacts in the environment. The properties are defined in a standard ontology. This allows

general specifications to be created that can be supplied as standard library and deployed immediately in an environment that supports the ontology, without the need of an environment expert.

We introduce specification inheritance and customisation mechanisms that allow situation specification to be extended and customised easily. The set of situations is no longer fixed. Application developers can create new specifications using these mechanisms in addition to reusing specifications from the standard library, which can be deployed alongside new situation-aware applications.

Users can introduce new situations by performing simple adaptation to existing specifications, or customise situations, refining them with their own habits and preferences or those of their organisation, or specify exceptional cases.

Situation specifications may contain roles whose occurrence is variable, supporting recognition of situations whose structure is uncertain.

Specification inheritance provides a means of specifying situations at various levels of granularity. Situations-aware applications can then reason about the situation abstractly, as well as exploit the rich set of information provided by fine-granularity situation specifications.

We go on to explore each of these aspects of our approach in the following sections.

2 Example situations and applications

In our approach, the situation refers to the activity a single person or a group of people are conducting, or the collection of activities occurring in an environment.

Listed below are some examples of the kind of situations we wish to detect. Each of the situations would typically occur with members of a University department.

Travelling to work A person's location follows a path to their place of work.

Checking mail A person is working with mail reader, web browser, and document reader tools, with the mail reader tool being used most frequently.

Collecting from a printer A person collects a document they previously printed from a printer.

Project meeting People that are members of a project are assembled in a meeting room.

PhD meeting A PhD student and their supervisor are talking in the supervisor's office.

Presentation An audience is assembled in a meeting room, a projector is running, and presentation software is running. The host introduces the speaker(s), the speaker(s) present, and then the speaker(s) answer questions posed by the audience.

Coffee break The time is around either 11 or 4 o'clock, and a group of people are assembled in the staff room, drinking coffee.

After-work drinks In the late afternoon or evening, members of a department are gather in a bar, chatting over drinks.

In addition, examples of situation-aware applications are listed below. These applications shall be used to illustrate various aspects of our situation determination approach throughout this paper.

Availability checker This situation-aware application reports the current situation of a person or a room, allowing the user to make decisions more effectively. For example, a secretary may choose to not to forward a call, as although the recipient is in their office, the application reports that they are currently involved in a meeting.

Mode manager When a particular situation occurs, this situation-aware application sends a message to a set of devices, requesting that they switch into a particular mode of operation. For example, a user may configure the application to detect when they are the speaker in a presentation, and to switch their mobile phone into a silent mode, and the laptop they are using to present slides with to disable its screen saver and power saver modes.

Situation enhanced file search This situation-aware application allows the user to organise their files by the situations in which their were created, accessed, or modified. For example, the user may search for all files they accessed during a particular project meeting.

From considering the situations presented above in addition to other situations from work, domestic, and social scenarios, we propose that a situation can be robustly characterised by the combination of four main aspects:

- The time at which the situation occurs, as well as its duration.
- Where the situation occurs, and the properties of these locations.
- The attributes of the group of people that are present.
- The artefacts that are present, and the manner in which they are being used.

Furthermore, we make the observation that a situation can be viewed as a collection of roles that are being played by a group of actors. For example, the presentation situation described above comprises the roles of 'host', 'speaker', and 'audience member' where people are the actors, and the role of 'presentation equipment' where the projector and the presentation software are the actors. The coffee break situation comprises the roles of 'coffee break venue' where the staff room is the actor, the role 'coffee break time' where the current time is the actor, and 'drinking coffee' role where the people and their coffee cups are the actors.

We go on to look at the details of how roles are specified in the following section.

3 Situation specifications

A situation specification provides a description of a situation in a form that enables an occurrence of the situation in the environment to be recognised automatically by a computer.

A situation specification consists of a collection of roles. A role is a description of a smaller part of the overall situation we wish to recognise. A role describes a set of conditions on the observable properties of things in the environment that hold when the role is occurring. As such, a role can be thought of as a unit of recognition. A situation is occurring when all of the roles in its specification are occurring simultaneously.

The things in the environment to which the conditions of a role refer are called actors. An actor may be the current time, a location, a person, or an artefact. Each actor has a set of properties that describe its current state. For example, the properties of a person may include their name, their occupations, and the groups they belong to, and whether or not they are talking.

A location may appear as an actor, or as a property of a person or artefact. Our approach does not demand a specific location infrastructure. For a review of different location infrastructures currently proposed in the literature, see Becker et. al [7]. However, it does require that the underlying location infrastructure supports the following three primitives:

1. It can provide the distance between two objects.
2. It can provide the symbolic coordinates of the location of an object. Example symbolic coordinates include 'Room L10.01d' and 'Graham's desk'.
3. It can provide the types of the symbolic coordinates of the location of an object. The types of a symbolic coordinate indicate the category or function of the location. For example, symbolic coordinate 'Room L10.01d' may have types 'Meeting area' and 'Room', while 'Graham's desk' would have the type 'Desk area'.

Figure 1 shows the situation specification for a meeting. The specification contains a lot of information, and we shall look at each part in turn.

```
role: meeting attendee
actors: p:Person
roles: empty
expressions:
  p.location contains symbolic coordinate ?s with type 'Meeting area'

situation: meeting
actors: loc:Location
roles:
  attendees: meeting attendee
expressions:
  loc has symbolic coordinate attendees.?s
  attendees.cardinality >= 2
```

Fig. 1. An example specification of a meeting situation.

From Fig. 1 we can see that structure of a specification has four parts - a name, a set of actors, and set of dependant roles, and a set of Boolean expressions.

The name provides a label for the specification. The prefix 'role:' indicates that this specification describes a role, and will form part of a larger situation specification. The prefix 'situation:' indicates that this is a situation specification.

The set of actors lists all of the actors that are required to play the role. For each actor, a label and its type are specified.

The set of dependant roles lists other roles that must occur for the role to occur. When a dependant role is used in this way, a cardinality property is defined for it. The cardinality indicates how many instances of the role are occurring simultaneously in the environment.

It is the situations that are taking place at a particular location, or that a particular person is involved in that is reported by the system to situation-aware applications. The situation frames the roles it contains. Therefore, a situation specification will be at the top level of a specification and as such, situation specifications may not appear as dependant roles.

Each role has a duration property implicitly defined for it, with the label duration. The property indicates the length of time the role has been occurring in the environment. Similarly, each role has the current time actor implicitly defined, with the label time.

When a role is occurring in the environment, all of the Boolean expressions in the set expressions will hold. An expression may include variables (the variable of the meeting attendee in Fig. 1 is denoted as ?s) that are bound to specific values when the expression holds. An expression can include standard comparison operators $<$, $\leq$, $=$, $\geq$, and $>$, the standard Boolean operators $\wedge$, $\vee$, $\neg$, and $\rightarrow$, and as well as other type specific operators. An expression may refer to the properties of an actor in its role or in dependant roles, the implicit time actor, the duration property of the role or of dependant roles, the cardinality property of dependant roles, the variables defined previously in other expression in the set or the sets of dependant roles, and literal values.

The first role shown in Fig. 1 is meeting attendee. Its name prefix is role: indicating that this is a role specification. It's set of actors contains a single person labelled p. Its set of dependant roles is empty. Its set of expressions contains a single Boolean expression that holds when the current location of person p contains a symbolic coordinate that has the type 'Meeting area'. The variable ?s will be bound to the matching symbolic coordinate. Therefore, when a person's location contains a symbolic coordinate with type 'Meeting area', they are playing the role of meeting attendee in the location bound to ?s.

The next role shown in Fig. 1 is meeting. Its name prefix is situation: indicating that this is a situation specification. Its set of actors contains a single location labelled loc. It set of dependant roles contains the role meeting attendee labelled 'attendees'. Its set of expressions contains an expression that holds when the location loc has the symbolic coordinate bound to ?s, and the cardinality of attendees is greater than or equal to 2. Therefore, when there are two peo-

ple playing the role of meeting attendee in the same meeting area, a meeting is occurring at location loc.

3.1 Specification inheritance

We will now go on to look at the specification for a project meeting, shown in Fig. 2. As a project meeting is similar to meeting, it is undesirable to have to duplicate parts of the specification that are common to both the meeting and project meeting specifications. This duplication can be avoided by using specification inheritance.

```
role: group member attendee
inherits: meeting attendee
actors: empty
roles: empty
expressions:
  p is a member of group ?g

role: other attendee
inherits: meeting attendee
actors: empty
roles: empty
expressions:
  p is not a member of group ?g

situation: group meeting
inherits: meeting
actors: empty
roles:
  gma: group member attendee
  oa: other attendee
expressions:
  gma.?g = oa.?g
  gma.cardinality >= 3
  oa.cardinality >= 0
  gma.cardinality >= (gma.cardinality + oa.cardinality) * 0.7
```

Fig. 2. An example specification of a group meeting situation.

Specification inheritance introduces an inherits part into the specification structure. The prefix inherits: indicates that this role inherits another role. A role may inherit one or more other roles. As situation specifications must appear at the top level of a specification, role specification may only inherit other role specifications, and situation specifications may only inherit other situation specifications.

When a role that inherits another is occurring in the environment, all of the expressions in the role and of the inherited role must hold. Expressions in the role can additionally refer to the properties and variables to which the inherited role can refer.

From Fig. 2 we can see that the role group member attendee inherits from the role meeting attendee. Only an additional expression is specified, requiring that person p is a member of the group ?g. Therefore, when a person's location contains a symbolic coordinate with type 'meeting area' and the are a member of the group ?g, they are playing the roles of both meeting attendee and group meeting attendee. The role other attendee is similar, though it specifies the additional expression that person p is not a member of the group ?g.

The group meeting specification shown in Fig. 2 extends the meeting specification from Fig. 1. It specifies two dependant roles. The first is on the group member attendee role labelled gma, and the second is on the other attendee role labelled oa. The expressions of this specification require that the group ?g bound in each instance of dependant roles gma and oa match, that there are at least three group members present, that there are zero or more other attendees present, and that of all attendees, at least 70% of them are group members.

From Fig. 2 we can see that only details bespoke to the group meeting situation had to be specified. The details of the more general meeting situation were reused.

3.2 Customisations

In additional to extending situation specifications through inheritance, specifications may be refined through customisation.

Situation specifications provided by the standard library are necessarily general or domain specific. It is desirable to tailor specifications to the particular environment of an organisation and to the particular preferences of an individual person to achieve higher situation recognition accuracy. This can be done using situation customisations.

An example customisation is shown in Fig. 3 along with the specification for the situation PhD meeting.

Customisation is similar to inheritance in that when a customised role is occurring in the environment, all of the expressions in the customisation and of the original role must hold, and that expressions in the customisation can additionally refer to the properties and variables to which the original role can refer. Unlike inheritance however, a customisation does not create a new specification, but alters an existing specification.

The PhD meeting specification in Fig. 3 states that when a PhD student and one or more of their supervisors are in the same meeting area, a PhD meeting is occurring. This specification would be supplied by the standard library.

A customisation of the PhD meeting specification is shown in Fig. 3, labelled sotirios supervisor. The customisation: prefix indicates that this specification is a customisation. Specification customisation introduces a customises part into

```
role: phd supervisor
inherits: meeting attendee
actors: empty
roles: empty
expressions:
  p supervises ?phd

situation: phd meeting
extends: meeting
actors: phd:Person
roles:
  supervisor: phd supervisor
expressions:
  supervisor.cardinality >= 1
  phd = ?phd

customisation: sotirios supervisor
customises: phd meeting
actors: empty
roles: empty
expressions:
  supervisor.p.id = SOTIRIOS_ID -> ?s = L10.01a
```

Fig. 3. An example specification of a PhD meeting situation.

the specification structure. The prefix customises: indicates which specification it customises. A customisation can be applied to a single specification only.

The sotirios supervisor customisation is defined for the PhD supervisor Sotirios. It refines the phd meeting specification by stating that a PhD meeting in which he is the supervisor, can only occur in his office, room L10.01a.

3.3 Temporal operators

In the specifications we have looked at so far, the expressions have contained conditions based on the current values of an actor's properties, and the roles that are currently occurring. Some situations will be characterised by the changes in value of an actor's properties and which roles are occurring over time. In this section we present operators that a specification can use to describe these changes.

Two temporal operators are defined for use in expressions. The first is a sequence operator that matches the history of a property's value over specified length of time against a given sequence. An example role using this operator is given in Fig. 4. The second is a proportion operator that holds when the history of a property's value matches a given value for a certain proportion over a specified length of time. An example role using this operator is also given in Fig. 4.

```
situation: taking a route
actors: p:Person
roles: empty
expressions:
  p.location has sequence ?s1

role: active application
actors: app:Application
roles: empty
expressions:
  app.isActive has proportion of value true for
    more than 80\% over the last 30 seconds
```

Fig. 4. Example specifications that make use of temporal operators on an actor's properties.

Three temporal operators are defined for use in roles. The first is in order, which requires that the specified group of roles occur in order. Figure 5 shows the in order operator being used with label r. It specifies that the roles labelled r1, r2, and r3 happen in that order. When the in order operator is used in this way, a duration property is defined for it, with the label duration. It indicates how much time has passed since the first role in sequence occurred.

The second operator is collection, which requires that all of a specified group of roles occur, but in no particular order. Figure 5 shows the collection operator being used with label r. It specifies that the roles labelled r1, r2, and r3 must occur. When the collection operator is used in this way, a duration property is defined for it, with the label duration. It indicates how much time has passed since the first occurrence of a role in the collection.

The third operator is repeat, which indicates that a role occurs a number of times. Figure 5 shows the repeat operator being used with label r. It specifies that the role labelled r1 occurs repeatedly. When the repeat operator is used in this way, a repetitions property is defined for it, with the label repetitions. It indicates how many repeats of the role have occurred.

These operators may be used in combination. For example, a set of roles may occur in order, and is repeat four times.

3.4 Recognising situations

The process of recognising an occurrence of a situation in environment is searching for a set of actors whose properties satisfy all of the expressions contained within the situation's specification.

However, it is possible that the expressions of a specification for one situation will be a subset of the expressions of a specification for another situation. This will always be the case with a specification that is inherited by another, but it is also possible that the subset relation will exist between two specifications that are not connected through inheritance.

```
role: in order example
actors: empty
roles:
  r: in order:
    r1: role1
    r2: role2
    r3: role3
expressions:
  r.duration < 20 minutes

role: collection example
actors: empty
roles:
  r: collection:
    r1: role1
    r2: role2
    r3: role3
expressions:
  r.duration < 20 minutes

role: repeat example
actors: empty
roles:
  r: repeat:
    r1: role1
expressions:
  r.repetitions <= 5
```

Fig. 5. Example specifications that make use of temporal operators on roles.

The system orders specifications on this subset relation. When two situations are occurring simultaneously and one is subset of the other and they are not connected through inheritance, the subset situation is suppressed and not reported by the system. This is because the subset specification is a false positive match, and suppressing it gives greater situation recognition accuracy.

Two broad categories of situation-aware applications can be defined based on the manner in which the reported situation information is used. The first category is reactive applications. Applications in this category use information they receive about a situation to influence their operation throughout the situation as it is occurring. The availability checker and mode manager applications fall into this category. The second category is archival applications. Applications in this category use the information they receive about a situation to influence their operation once the situation has ceased to occur. The situation enhanced file manager application falls into this category.

As reactive applications require to be notified of a situation occurring as soon as possible. This requires that the temporal operators must hold as soon as possible. Let's return to the example situation taking a route, given in Fig. 4. If the sequence operator were to hold only when a match on the full sequence of locations were made, the situation would be reported as occurring at the point when it ceased to occur. This is of no use to reactive applications, as they will have missed the occurrence of the situation. To overcome this, the sequence operator must hold as soon as the person's location begins to match the specified sequence. This way, reactive applications receive notification of the situation when it begins. Similarly, the in order and collection operators must hold as soon as the first role occurs.

For situation specifications that differ only by the use of a temporal operator, it will be impossible to distinguish between them until some discriminating condition expressed in the temporal operator holds.

For example, let's say situation A specifies that role R_a occurs with a duration of 30 minutes, and situation B specifies that role R_a occurs with a duration of twenty minutes, and then role R_b with a duration of 10 minutes in order. In the environment, role R_a has been occurring for 10 minutes. At this point, it is impossible to say whether situation A or situation B is occurring. For reactive applications we may assume that both situations are occurring. At twenty minutes, role R_b occurs in the environment, and it is now clear that situation B is occurring. Archival application can note this and suppress situation A in the archive. Reactive applications can now identify the situation B is occurring, but cannot avoid incorrectly identifying situation A for the first 20 minutes.

This is an unavoidable trade-off between responsiveness and recognition accuracy. Situation specifications can alleviate this slightly by specifying a minimum duration of the situation, but that means that reactive applications cannot react to the situation until the minimum duration has passed.

This results in different situation specifications being better suited to use by different situation-aware applications. Reactive applications that must react immediately to a situation and cannot tolerate reduced recognition accuracy are

best to use only situation specifications that do not use temporal operators, or that have a short minimum duration. Archival applications and reactive applications that can tolerate reduced recognition accuracy may use specifications that do use temporal operators.

This is not as restrictive as it may first appear. From the specifications we have constructed, we have found that the majority of situations of interest can be defined without temporal operators.

3.5 Ontology

The ability to produce the values of a person's or artefact's properties is not provided by the system. The means of doing this will be specific to the particular device used to detect the properties of a person or artefact, and so must be supplied by the devices themselves.

These devices may originate from many different sources. In order for the system to recognise its properties, they must have a common, consistent set of names. As such, we define an ontology describing the properties that each type of actor is expected to posses. This allows information to be exchanged and interpreted correctly between many different area, person, and artefact agents, and the information to be translated to a different ontology that may be used in other environments.

4 A situation determination architecture

This section proposes an architecture for our situation determination approach.

The architecture is agent based and is illustrated in Fig. 6. The following types of agents are defined:

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

A personal server agent (PSA) This agent represents a person. Each person is assumed to wear or carry with them a device that hosts this agent. Typically this device would be a PDA, a mobile phone, or a Personal Server [8]. A PSA will have knowledge of the person's properties, as well as any situation specifications and customisations particular to the person.

An artefact agent (AA) This agent represents an artefact. An AA will have knowledge of the artefact's properties and is hosted on artefact itself.

A situation-aware application agent (SAA) This agent represents an application that shall use situation information to influence its operation. A SAA may run on any appropriate device.

As noted in the previous section, situation specifications from the standard library may be customised or extended. These additional specifications may be

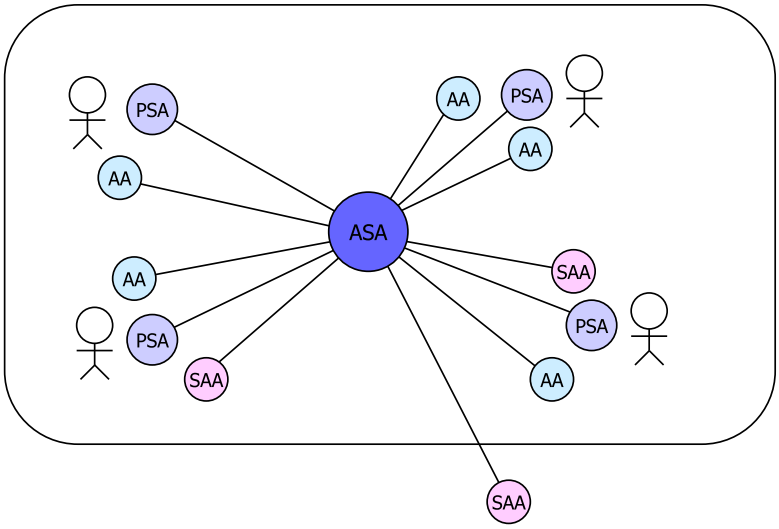

Fig. 6. An example deployment showing the architecture of the situation determination system. The rounded rectangle represents the physical are the ASA governs.

particular to an area, for example throughout an origination's building, or be particular to a person. Additional specifications particular to the area will be known to the ASA as well as the standard library specifications. Additional specifications particular to a person are stored by their PSA, and so must be made available to the ASA when the person is within range of it. The total set of situation specifications known to the ASA and which it will attempt to recognise will change as different people, and their own additional specifications, enter and leave the range of the ASA. This set is referred to as the active situation set.

The architecture's star topology offers the advantages that redundant determination effort is eliminated as the situation for all of the people and artefacts in the environment is performed once, all of the customised specifications from each PSA can be combined to give greater situation recognition accuracy, the ASA is likely to be more powerful than a PSA and so can perform the determination more quickly, and it reduces the drain on the battery power of each PSA's mobile host. These are seen to outweigh the typical disadvantages of a centralised architecture, in that the ASA is a single point of failure and that it may be become a communication bottleneck.

An agent has a set of behaviours and executes each behaviour in the set in turn repeatedly. When a behaviour is terminated, it is removed from the set. Behaviours may be added to or removed from the set as the result of executing another behaviour. A PSA has the following behaviours:

ASA discovery :

– When this behaviour is active, the PSA will carry out the following steps:

1. If as ASA has not currently been found, attempt to discover one. This will be repeated periodically until an ASA is found.
2. Add the person state update behaviour and terminate this behaviour.

Person state update :

- When this behaviour is active, the PSA will 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 the additional and customised specifications to the active specification set, and then analyses the set to discover which conditions on which properties of that agent type it requires to be informed of. This information is sent back in a reply to the PSA.
  3. The PSA then monitors the required conditions and sends a message to the ASA when they hold or cease to hold.
- If the set conditions required by the ASA changes, for example in response to a change in the active situation set, the ASA sends the PSA a message informing it of these changes and the PSA updates it monitoring appropriately.
- If the connection with the ASA is lost, the PSA will add the ASA discovery behaviour, and terminate this one.

The behaviours of an AA are similar, with the exception that an AA will not have additional or customised specification.

An SAA has the following behaviours:

ASA discovery :

- Similar to that for PSA, but a query ASA behaviour is added rather than a person state update behaviour.

Query ASA :

- When this behaviour is active, the SAA will carry out the following conversation with the ASA:
  1. The SAA sends a message to the ASA informing it of which situations it want to want to be notified as well as any conditions on those situations. For example, is may wish to know that a group meeting is taking place, but only if it is in a specific room.
  2. The ASA then monitors the situations of interest and sends a message to the SAA when they occur matching the associated conditions, and when they cease to occur.
- If the situations or their conditions that the SAA is interested in change, the SAA sends a message to the ASA informing it of these changes and the ASA updates it monitoring appropriately.
- If the connection with the ASA is lost, the SAA will add the ASA discovery behaviour, and terminate this one.

The behaviours of an ASA are simply the complement of each of the behaviours described above.

The architecture facilitates situation determination for a large number of situations and actors, while defining only a small set of agents with simple sets of behaviours.

5 Prototype implementation

We have constructed a prototype implementation of our situation determination system and the availability checker application.

We developed an area, person, artefact, and situation-aware application agent. The artefact agent was an application monitor report properties of current set of applications running on a host computer.

Location information is currently self-reported by a user through their PSA. The user identifies their location by selecting its symbolic coordinate from a drop-down list.

Each PSA stores a map of locations to the corresponding host address of the agent platform. Once connected to the platform, an agent discovers the ASA by using a well-known name.

Specifications were created for the following nine situations:

Using a specific application This is a general situation that occurs when a person is working with a computer, and is using a specific application for more than 70% of time over the last 30 seconds.

Checking e-mail This situation inherits from Using a specific application, and specifies that the application is an e-mail client.

Surfing the web This situation inherits from Using a specific application, and specifies that the application is a web browser.

Reading This situation inherits from Using a specific application, and specifies that the application is a document reader.

Coding This situation inherits from Using a specific application, and specifies that the application is an IDE.

Meeting This is the general meeting specification as described in Fig. 1. A meeting occurs when two or more people are gather in a meeting area.

Group meeting This is the group meeting specification as described in Fig. 2. A group meeting occurs when three or more members of a group are gathered in a meeting area. If other attendees are present, the group members must constitute 70% or more of all the attendees.

PhD meeting This is the PhD meeting specification described in Fig. 3. A PhD meeting is occurring when a PhD student and one or more of his/her supervisors are gathered in a meeting area. The customisation described in Fig. 2 is also defined. This states that a particular supervisor only holds PhD meetings in his office.

Demonstrator meeting A demonstrator meeting is occurring when the lecturer of a specific class and one or more of the demonstrators of the class are gather in a meeting area. This situation inherits the Meeting specification.

Recognition of situations is performed by the ASA using the JESS rule engine [9]. Situation specifications are translated offline into a set of JESS rules. Each role in the specification has two corresponding rules - an activation rule which fires when the role is occurring, and a deactivation rule that fires when role ceases to occur. When the activation rule for a situation specification fires, the ASA updates its internal model of each person in the situation showing that they are currently involved in that situation. Similarly, when the situation deactivation rule fires, or when a deactivation rule fires when a person is no longer playing a role in the situation, the situation is removed from the internal model of those people. In this way, the ASA maintains and up-to-date model of all the situations that are occurring within its environment. This allows situation queries from SAAs to be answered immediately.

Our implementation of the architecture is based on the JADE agent framework [10]. JADE provides all of the standard agent functionality required by our system, requiring that only the agents bespoke to our system be written.

As an initial test of the system, we developed the availability checker application. This allows it user to check the current situations of a particular person. The application can run on any platform supported by JADE-LEAP - a PC, a PDA, or a mobile phone. The user selects the person they wish to check the availability of from a drop-down list, and that person's current situations are displayed as a 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 'one-shot' update of the current situations was received, that is, we did not wish to monitor the situations over time, and that all situations for the particular person were to be reported. Each of situations listed above were identified accurately by the availability checker.

6 Analysis of approach and prototype implementation

In this section, we present an analysis of our situation determination approach and its initial prototype implementation.

The essence of our approach is that situations are viewed as a collection of roles, where a role is a unit recognition of a situation based on the observable properties people and artefacts in the environment, that are identified with common names defined in a standard ontology. This offers the following advantages:

- It allows the creation of a library of standard situation specifications that can be deployed immediately in an environment without the need of environment expert.
- It permits simple extension and customisation of these standard specifications to specific situations that occur in a particular environment, giving a higher level of recognition accuracy.
- It gives a discrete structure to the situation. This provides clear points for extended and customising its specification. Extended or customised specifi-

cations for a particular environment will often only have to bind particular values to variables it contains.
- Each role and situation has a distinct label. Using this label, graphical tools can present simple, meaningful GUIs to users to extend or customise the specification based on its semantics.
- For application developers that wish to create new situation specifications, roles provide a means of partitioning the potential very large set of sensor data contained within the situation into smaller, manageable sets of related sensor data.
- Using roles, application developers can create new specifications by assembling high-level constructs rather than manually correlating many individual sensor readings.
- Roles can reused across several specifications, reducing the overall specification effort required.
- It offers greater utility to situation-aware applications, as situation queries can refer to the roles within a situation. In other current approaches, application may only refer to the situation label.
- Situations are recognised at various levels of granularity. This expands the scope of situation-aware applications such as the mode manger, which can now partition the set of behaviours it applies to the most appropriate level of granularity, and the situation enhanced file manger application that can now associate files with the most specific situation in which it was used.
- Explicitly support for temporal operators is provided for situation specification, enabling a richer set of situations to be recognised.

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 availability checker application under 6KB for the agent and GUI. The average size of the situation specifications used in this analysis is approximately 2KB of text. A situation customisation is not likely to exceed 1KB of text.

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 updated that changed the situation of the PSA's representative, until the PSA received the message informing it of the change in 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. Figure 7 shows the average time measurements for increasingly complex scenarios. The results show that the round trip situation update time increases proportionally to the complexity of the scenario.

The proportion of time consumed by JADE, JESS, and the custom situation determination code required to process a single situation update on the ASA was measured using the Eclipse profiling tools [11]. Taking the average cumulative CPU time for a single situation update over all nine situations with eight PSAs

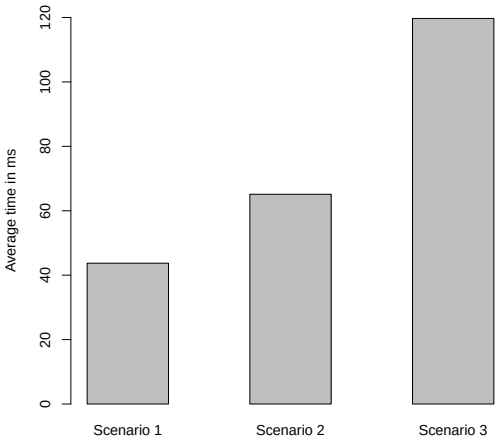

Fig. 7. This figure shows the round trip situation update time between a single PSA and ASA. In scenario 1, the active situation set of the ASA contained only a single situation and only one PSA sending updates. In scenario 2, the active situation set contained all nine situations and eight PSA sending 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.

showed that JESS consumed 38.66%, JADE 58.06%, and the custom situation determination code 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, P is the average number of patterns per rule LHS, and F is the number of facts in the knowledge base [12]. From this the complexity of an active situation set can be calculated. Each actor within range of an ASA is presented by a single fact in the knowledge base. Each role produces an activation and deactivation rule. The number of patterns in the rule LHS will equal the number of expressions in 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, A to be the number of actors, and E to be average number of expressions including cardinality bookkeeping, the complexity is $O(2R'(A + R')E)$, or $O(2R'^2E + 2R'AE)$ which is quadratic on the number of roles.

The complexity can be shared amongst several ASAs. For example, if the load on the host of an ASA whose range is a building becomes too great, an ASA can be assigned to each floor of the building. Similarly, if a large, public room that can play host to many different types of situations and potentially has a large number of roles in the ASA's active situation set, then the room could be assigned its own ASA.

Due the centralised of the situation determination process, for an area that experiences a sudden influx of a large number of people, the ASA could become a

communication bottleneck. In a typical office environment, for which this system has been designed, it is not expected that such large, sudden influx of people will occur.

7 Future work

In the current implementation of our system, all of the expressions on the states and properties of a person or tool are reported as a crisp Boolean value. However, many sensing technologies produce sensors readings with a degree of uncertainty. To incorporate these technologies fully, situation recognition must take this uncertainty into account. This may be done by replacing JESS with fuzzy logic or Bayesian network based reasoning.

As it is only the ASA that performs recognition of situations, PSAs can only operate when they are within range of an ASA, restricting their mobility. Its desirable to have a person's situations recognised wherever they are. That is, to have the person's PSA act as an ASA in the absence of a fixed ASA in the environment. We plan to extend the PSA implementation to incorporate this functionality.

Currently, the situation specifications have been translated into their representative JESS rules by hand. We plan to develop a tool that facilitates the graphical specification of the situations and automatic translation to JESS rules.

A fuller application based evaluation of the system is planned, with the development of several more situation-aware applications of greater complexity.

Furthermore, we plan to evaluate the middleware on a larger deployment, covering an extended set of situation and types of artefact agent.

8 Summary

In this paper, we have presented a novel approach to situation determination that offers several advantages over other state-of-art approaches. It provides a reusable library of situation specification that can be deployed immediately by non-expert users, which may be extended and customised to recognise fine-granularity situations of a particular environment.

We presented an agent-based architecture that supports our situation determination approach as well as a description of an initial prototype implementation. An analysis of our approach and prototype implementation was given, which demonstrated that our approach is viable, and can accurately identify situations for ad-hoc group of people and devices with sufficient responsiveness for a large number of people, devices, and situations.

References

1. M. Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991.

2. Daniel Salber Anind K. Dey and Gregory D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Anchor article of a special issue on context-aware computing in the Human-Computer Interaction (HCI) Journal*, 16(2-4), 2001.
3. 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.
4. Jakob E. Bardram. The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications. In *Pervasive*, pages 98–115, 2005.
5. 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.
6. 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.
7. Christian Becker and Frank Durr. On location models for ubiquitous computing. *Personal Ubiquitous Comput.*, 9(1):20–31, 2005.
8. Intel personal server project. See: http://www.intel.com/technology/techresearch/research/rs08031.htm.
9. Jess, the rule engine for the java platform. See: http://herzberg.ca.sandia.gov/jess/.
10. Jade - java agent development framework. See: http://jade.tilab.com/.
11. The eclipse test and performance tools platform. See: http://eclipse.org/tptp/index.html.
12. Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artif. Intell.*, 19(1):17–37, 1982.