Situation Determination with Reusable Situation Specifications

Graham Thomson and Sotirios Terzis
Pervasive and Global Computing Group
University of Strathclyde
Glasgow, UK

Paddy Nixon
Systems Research Group
University College
Dublin, Ireland

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. 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 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 [1, 2], 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 [4, 3] a training period must be performed during which several examples of each situation are collected and analysed, before the system can be used. Furthermore, situations are recognised at a low-level of granularity, which limits the scope of situation-aware applications. For example, in [4, 3] 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.

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 recognition of a situation based on the observable properties of people and devices in the environment, that 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 already support various levels of granularity, including 'group meeting' and 'PhD progress meeting' in addition to a 'meeting' 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.

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 specifi-

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE

cation, 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 essential for describing situations. A location property is defined for people and devices. Our approach requires that the underlying location infrastructure 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. In addition to this, we define location types for the symbolic coordinates. 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 duration, which indicates how long a role has been occurring 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, the current time, and the duration and cardinality properties of other roles. They may include the standard comparison operators, the Boolean operators $\wedge$, $\vee$, and $\rightarrow$, as well as other type specific operators.

An example specification of a presentation situation is given in Fig. 1. Three roles are defined - speaker, audience member, and presentation. 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 presentation is occurring when a speaker, three or more audience members, and a computer running presentation software are in the same room.

Different levels of granularity of a situation can be expressed through specification inheritance, providing a simple way to create new specifications as variations of another, and interpret the same situation at different levels of abstraction. Existing situations can be refined to a particular environment or person using specification customisation. Further details of these mechanisms can be found in [5]. Taken together, the constructs described above provide a sufficiently expressive language to define situation specifications.

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

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

role: presentation
entities: s:speaker, a:audience member, c:Computer
expressions:
  s.cardinality = 1, a.cardinality >= 3
  presentation software is active application on c
  symbolic coordinate of type 'Room' of s.p.location =
    symbolic coordinate of type 'Room' of a.p.location
  symbolic coordinate of type 'Room' of s.p.location =
    symbolic coordinate of type 'Room' of c.location
```

Figure 1. A presentation specification.

3 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.

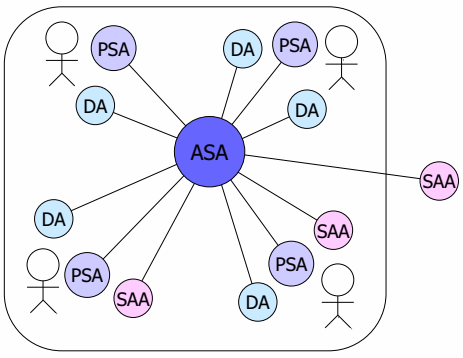

Figure 2. An example deployment.

Our proposed agent-based architecture is illustrated in Fig. 2. The rounded rectangle represents the physical area the ASA governs. 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 as 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.

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. A DA is hosted on the device and has knowledge of the device's properties. A general DA is defined that allows devices to communicate with the agent infrastructure. Specific DAs are defined that extend the general DA and can access the properties of a specific device or platform.

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. A SAA may run on any appropriate device.

Agents can discover and communicate with each other via an agent platform substrate. Agents may be discovered by their identity (white-page look up) or by the services they provide (yellow-page look up). In our architecture, agents can connect to the agent platform either through a wired or wireless network. The area server machine is connected to a wireless network on which it advertises its presence using a well-known name. PSAs and DAs can then discover the area server when they come in range, and connect to the agent platform.

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 then monitors the required 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 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 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. These advantages outweigh the typical disadvantages of a centralised architecture, where the ASA is a single point of failure and may be a communication bottleneck. The architecture 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.

4 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.

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 (http://herzberg.ca.sandia.gov/jess/). 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 (http://jade.tilab.com/). 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 results 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.

Furthermore, we calculated the complexity of an active situation set. 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, the complexity is $O(2R^2E + 2RPE)$ which is quadratic on the number of roles. This 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. Due to the centralised nature 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. 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. Further details about the above results and analysis can be found in [5].

5 Conclusions and future work

In this paper, we have presented a novel approach to situation determination that provides 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.

In the future, we plan to extend situation recognition to incorporate uncertainty, as many sensing technologies produce uncertain readings. We also intend to improve the mobility of the system by having a PSA perform lightweight situation recognition when it is outwith the range of an ASA. 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 evaluate the middleware on a larger deployment, covering an extended set of situations and types of device agent.

References

[1] D. S. Anind K. Dey and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI Journal*, 16(2-4), 2001.

[2] 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.

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

[4] 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.

[5] G. Thomson, S. Terzis, and P. Nixon. Situation determination with reusable situation specifications. Technical report, University of Strathclyde, 2005. Available at: http://smartlab.cis.strath.ac.uk/Publications/techReports.htm.