A Self-Managing Infrastructure for Ad-hoc Situation Determination

Graham Thomson [a], Graeme Stevenson [b], Sotirios Terzis [a] and Paddy Nixon [b]

[a] *Pervasive and Global Computing Group, University of Strathclyde, Glasgow, UK*
[b] *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. The architecture of an accompanying situation determination infrastructure is provided, which autonomously optimises and repairs itself in reaction to changes or failures in the environment.

Keywords. Situation-awareness, Pervasive computing, Agent-based systems

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, or 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 [3,4] 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 course level of granularity, which limits the scope of situation-aware applications. For example, in [3,4] 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 doctor or a patient in a consultation.

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, 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. 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 information required for situation-aware applications is rarely at the same level of abstraction as that provided by individual data sources. We provide an infrastructure that can obtain and process sensor data from a variety of disparate sensor technologies and deliver it to applications at the level of abstraction they desire. The infrastructure also monitors, optimises and repairs itself as changes or failures occur in the environment.

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 [1]. 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. An example symbolic coordinates is 'Ward L10.01'. Both of these primitives are commonly supported by location systems [5]. In addition to this, we require location types of symbolic coordinates, similar to those employed by Look et al. [6]. These types indicate the category or function of the location. For example, the symbolic coordinate 'Ward L10.01' may have types 'Consultancy area' and 'Private ward'.

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,

```
role: warden                              situation: home visit
entities: p:Person, t:Time                roles:
expressions:                                wdn: warden
  p is warden of Dunaros Hospital           ptn: patient
  t is within working hours of p          expressions:
                                            wdn.cardinality = 1
                                            ptn.cardinality >= 1
role: patient                               wdn.p is warden for ptn.p
entities: p:Person                          ptn.p.location is within home of ptn.p
expressions:                                'Room' of wdn.p.location =
  p is patient of Dunaros Hospital             'Room' of of ptn.p.location
```

Figure 1. A simple 'home visit' specification.

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 situation in which a hospital warden is visiting a patient at home is given in Fig. 1. Two roles are defined - warden and patient. Each role lists the entities its expressions refer to, where an entity is a person, a device, or another role. The warden role is played by a person who is a warden for a particular hospital during working hours. The patient role is played by a person who is a patient of the particular hospital. A home visit is occurring when one or more patients and their designated warden are within the same room of the patients' home.

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. Existing situations can be customised for a particular environment, person, or application using specification customisation. Further details of these mechanisms can be found in [7].

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 involved in a home visit, that a person is either a warden or a patient. At a higher resolution still, we may report that a person or device is playing a more specific role, for example, different types of warden or patient may be defined.

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
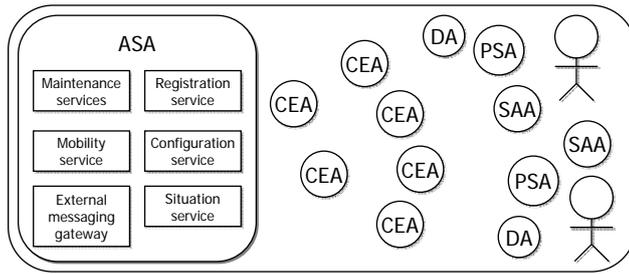
Figure 2. An example deployment.

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. Our system employs fuzzy logic based reasoning to incorporate and combine the confidence values of properties in a situation specification. This process is automated to its fullest extent, such that the uncertainty experienced within a smart environment can be managed effectively without being the burden of the specification author. Further details can be found in [7].

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.

Our proposed agent-based architecture is illustrated in Fig. 2. The rounded rectangle represents a bounded physical area, called a range, which will typical be a room. Each type of agent defined is described in turn in the next section.

3.1. Types of agent

An area server agent (ASA) performs situation determination for all of the people within a range. 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. The operation of an ASA calls upon several services, each of which is described in the next section.

A personal server agent (PSA) represents a person, who 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) 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 DA does not carry any additional specifications and customisations.

A situation-aware application agent (SAA) 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.

A context entity agent (CEA) represents a function that operates on the properties produced by a PSA, DA, or another CEA. Several flavours of CEA are provided. These include a fusion CEA which takes as input multiple properties of type T and produces a property also of type T whose quality has improved over that of the input (e.g. a more accurate estimation of the location of a person based on events produced by RFID and IR sensors), as well as an aggregator CEA which outputs a property of arbitrary type based on one or more inputs, also of arbitrary type (e.g. inferring which person is using a particular PDA based on who is logged on to the PDA and the position of the device and the person). Please refer to [8] for the full list of supported CEAs.

3.2. Area server services

Our system is comprised of a number of ranges which self-organise to form a partially connected overlay network. Each range is functionally equivalent and contains a set of services that are used for the management of the personal and device information available within the range as well recognition and dissemination of the occurring situations. Any agent or application which utilises the services provided by a range is referred to as being a part of that range. The infrastructure places no restrictions on the physical placement of range components within the network.

The overlay network is formed using a self-organising, self-repairing peer-to-peer protocol [9], and provides functionality for dealing with agents and applications which may move between ranges during their lifetime, and for managing the interactions required to obtain properties from DAs or PSAs in remote ranges.

The services provided by a range are managed by the ASA. There are six services in total, as shown in Fig. 2. The registration service records which agents are currently part of the range as well as which properties each agent can provide, and which situations SAAs wish to be notified about. 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 and is maintained by the situation service. The configuration service acts as a bridge between the situation service and the registration service. It composes and instantiates graphs of CEAs, called configurations, which are capable of providing the situation service with the properties it requires. The external messaging gateway is used to obtain properties from other ranges via the overlay network, whilst the maintenance services monitor the status of all the agents within a range, performing repairs to configurations as required. The mobility service is responsible for supporting applications relocating to other ranges.

3.3. Agent interaction

Agents can discover 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. The following interactions then take place: 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 passes any new specifications to the situation service and the agent type information to the registration service, 3) the situation service analyses the active situation set to determine which set of properties it requires, and passes this information to the configuration service, 4) the configuration service creates a new configuration, or modifies the current configuration, such that it provides the properties required by the situation service from the properties available in the registration service.

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 similar interaction to that of the PSA, with the exception that it shall not send any additional or customised specifications.

When a SAA discovers the ASA, the following interactions occur: 1) the SAA sends a message to the ASA informing it of the situations it wants to be notified about, 2) the ASA passes this information to registration service which then sets up appropriate notification from the situation service, 3) when situations of interest occur or cease to occur, the situation service notifies the appropriate applications.

3.4. Automatic path creation

The configuration service employs automatic path creation (APC) techniques in order to generate configurations that are capable of satisfying the requirements of the situation service. This section describes three aspects of this process: the APC mechanism itself, the techniques implemented to reuse existing configurations and CEAs where possible, and the process of maintaining configurations during their lifetime.

Restricting, for now, discussion of the resolution process to a single range, the process carried out by the configuration service is as follows. A property is represented by a type, e.g. Temperature, and the range of values that a particular data source can support, e.g. $0°C < value < 100°C$, and optionally a location. The configuration service searches for CEAs which match the desired type, range, and location of each property requested by the situation service. The properties supplied by each candidate are then compared to the situation service's requirements and are classified into one of four categories: no match, more general match, exact match, and more specific match. The no match category contains CEAs whose output does not match any properties from the situation service's request. The more general match category contains CEAs whose output is more general than that required by the situation service. The exact match category

includes CEAs whose output has an exact one-to-one correspondence with the situation service request. Finally, the more specific category contains CEAs whose output is more specific than that required by the situation service. If any exact match category contains at least one CEA, the next step is to examine each of their input requirements (if any) in turn, and determine if they can be satisfied (using this procedure). This is a recursive process which continues until physical sources of data are found, that is, a PSA or a DA. If there is a choice to be made among multiple CEAs, the one with the classification that provides the higher quality of data is chosen, e.g. fusion CEA > (PSA or DA) > aggregator CEA. If there are no exact matches, the next step is to examine the input requirements for any more general matches in a similar manner. If a complete configuration can be formed, a filter is automatically generated and configured to bridge the gap between the output of the configuration and the requirement of the situation service. Should the previous two groups fail to yield a positive result, the final option is to evaluate the group of CEAs in the more specific match category. The results of all successfully evaluated configurations can then be merged together to provide the situation service with the best possible match available.

Pervasive computing environments are dynamic with respect to the resources available within them at any one time. Furthermore, failure of computational devices should be treated as commonplace. To deal with these aspects, a suite of maintenance services is provided that: monitors agent failure; performs repairs to configurations where possible; and re-evaluates configurations when new resources become available [8].

The situation service supports an 'explain' request that may be issued by other agents in the system. In response, the service sends a message containing the specifications, customisations, and values of properties of the situations that it believes are currently occurring. 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.

We have constructed a prototype implementation of our situation determination system and several initial test applications, including an availability checker, a coffee break notifier, and a situation-aware to-do list application. We have performed both a performance and theoretical analysis of our initial prototype, which has shown our system to be promising approach to situation determination. Please see [7,8] for details of the analysis.

The architecture presented in this section facilitates robust 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. 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 also presented the architecture of a supporting agent-based self-managed infrastructure. 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 (e.g. fetching a patient's details on a warden's PDA when a home visit is occurring), and an automatic daily activity dairy application which records a patient's situations throughout the day and can be configured to monitor for particular situations that do or do not occur, as well as for unexpected or unrecognised situations occurring. 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.

References

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

[2] H. Chen, T. Finin, and A. Joshi. A context broker for building smart meeting rooms. In *Proc. Knowledge Representation and Ontology for Autonomous Systems*. AAAI, March 2004.

[3] 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.

[4] 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.

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

[6] Gary Look, Buddhika Kottahachchi, Robert Laddaga, and Howard Shrobe. A location representation for generating descriptive walking directions. In *Proc. Intelligent User Interfaces*, pages 122–129, New York, NY, USA, 2005. ACM Press.

[7] Graham Thomson, Sotirios Terzis, and Paddy Nixon. A model and architecture for situation determination. Technical report, University of Strathclyde, 2006. Available at: http://smartlab.cis.strath.ac.uk/Publications/techReports.htm.

[8] Graeme Stevenson. A Service Infrastructure for Change-Tolerant Context-Aware Applications. Master's thesis, University of Strathclyde, Glasgow, Scotland, 2006. (Submitted).

[9] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *LNCS*, 2218:329–350, 2001.