A Middleware for Pervasive Situation-Awareness

Graham Thomson, Sotirios Terzis

A Middleware for Pervasive Situation-Awareness

Graham Thomson and Sotirios Terzis

Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, UK
graham.r.thomson@gmail.com, sotirios.terzis@strath.ac.uk

Abstract. Situation-awareness is the ability of applications to adapt to the current situation of their users. For situation-awareness to be truly pervasive it should support the individual needs of every user, everywhere. We present a middleware for pervasive situation-awareness based on the idea of separating the features of a situation from the specification of how it should be recognised. The features of a situation can be seen as an interface that can be easily customised to satisfy individual user needs, while alternative specifications can be used to recognise a situation in different environments. The middleware views situations as collections of roles that individuals and devices play. Its implementation follows an agent-based architecture where collaborating agents acquire and reason over context data. We also show that the middleware can recognise a variety of highly customised situations using alternative specifications with performance that is acceptable for interactive situation-aware applications in realistic deployment sizes.

1 Introduction

Situation-awareness is considered a defining characteristic of pervasive computing systems [2]. It refers to the ability of applications to adapt their operation to the current situation of their users through the collection and interpretation of sensor data. More specifically, a situation is an interpretation of sensor data into a higher, domain relevant concept, understandable to users and applications [14]. Situations can be considered as semantic interpretations of sequences of contexts that describe particular aspects of the overall situation.

The main challenge for situation-awareness is how to recognise high-level concepts from the low-level sensor data. The challenge is confounded by the need for good appreciation of both the characteristics of the underlying sensing infrastructure, and the needs of user applications. To date there has been a lot of work to address this challenge, see [14] for a recent survey. Approaches take one of two approaches. They either utilise expert knowledge to explicitly specify situations (e.g. [3, 9, 4]), or learning algorithms to implicitly recognise situations (e.g. [8, 10, 1, 13]). However, due to the engineering effort required either to specify or learn a wide range of situations, both approaches are typically limited to quite generic situations. For example, they only recognise a generic meeting situation, instead of specific types of meetings that users may be interested in. They also

target specific sensing infrastructures, and are typically limited to a small number of environments. As a result, they fail to support the needs of every user, everywhere. They do not support pervasive situation awareness.

We believe that supporting the needs of every user across a wide range of environments can be best achieved by separating the description of situations and their features from the specification of how they are recognised. Our belief is based on the observation that in most domains users are interested in a wide range of specific variants of generic situations. In this context, variants of generic situations can be supported through customisation of the features of situations, while a situation could be specified in multiple ways to allow its recognition across different environments.

We have developed a middleware to support pervasive situation-awareness. The middleware provides a query-based interface that allows situation-aware applications to retrieve notifications about the occurrence of situations. To better deal with the volatility inherent in pervasive computing environments, our middleware follows a loosely-coupled agent-based architecture, with agents collaborating over a publish/subscribe substrate. Each agent is responsible for a particular aspect of the situation recognition process with reasoning over context data supported by a rule engine based on the Rete algorithm.

To demonstrate the expressive power of the middleware, we show that it is able to support recognition of a number of situations in a university environment through a range of specifications that can be easily customised to produce a large number of variants. To demonstrate that its performance is practical for interactive situation-aware applications we focus on the round-trip time of a situation recognition request, basically a summary of the overall performance of the system, and we show that by appropriately distributing the recognition load between agents the performance scales reasonably as the number of entities involved in the situation, and the number of situations to be recognised increases.

In the next section we look at previous work on situation-awareness. We then present our model for the description and specification of situations, followed by a description of our middleware architecture and implementation. This is followed by the evaluation of our middleware. Finally, we draw some conclusions and present some directions for future work.

2 Related work on situation-awareness

A key requirement for pervasive situation-awareness is the ability to support a large number of situations with limited engineering effort. In this respect, learning-based approaches are typically considered more appropriate as they are better suited for settings where there is a wide range of fairly inaccurate sensors to recognise a variety of situations [14]. In practice, both specification and learning-based approaches require significant engineering effort.

Specification-based approaches require domain experts to provide the logic-based or ontological specifications for each situation of interest. Learning-based approaches are mostly based on supervised learning techniques. These require

the annotation of the examples needed for the learning to take place. In recent years, there has been some work to try and minimize the number of examples, like [1, 13], by exploiting additional structure and knowledge to simplify the learning task. However, these approaches just trade the effort required for the annotation of the examples, with work by domain experts for the specification of the necessary structure and knowledge. In this context unsupervised learning techniques like [8, 10] would offer a significant advantage. However, they tend to be limited in scope and restricted to a small set of situations using a specific sensing infrastructure. Of particular note are techniques like [11] that have shown some promise in the identification of user activities such as walking, or sitting down. Such techniques could be utilised as part of the recognition of higher-level user activities.

It would seem that both specification and learning-based approaches are not really appropriate for pervasive situation-awareness. However, as often, most of the situations that need to be recognised are just variants of a small number of generic situations, the key point in keeping the overall engineering effort manageable is to make it easy to produce variants. In this respect, specification-based approaches have an advantage as they are more transparent to applications and users [14]. This makes the individual situation elements visible, making them easier to vary. This is particularly the case in approaches such as [3], which provide a well-defined structure for situations.

Another key requirement for pervasive situation-awareness is operation across various environments, i.e. recognition of the same situation in different ways, each aimed for a particular environment. Some specification-based approaches already support this [9, 4]. However, as they place limited structure over specifications, they make it quite difficult to produce variants of the specified situations, because one needs to fully understand the whole specification. At the same time, ontologies have been used to provide a common vocabulary for and facilitate sharing of situations, e.g. CobraOnt[1]. However, they also make it difficult to produce variants of the specified situations, as this requires good appreciation of ontological reasoning.

In conclusion, pervasive situation-awareness requires a specification-based approach that provides a well-defined structure for the specifications, is flexible enough to accommodate promising learning-based approaches, and supports multiple specifications for the same situation.

3 A model for pervasive situation-awareness

We define situations as semantic interpretations of context information (adopted from [14]) that represent the high-level activity of an individual or a group of people and the devices they use. The recognition of a situation involves information at the following levels of abstraction: sensor data, context information and situations. Sensor data represent the raw output from sensors. Context information is a structured representation of sensor data. This is a typed relation

[1] http://cobra.umbc.edu/ontologies.html

that represents some property of an entity derived from sensor data and/or other context information. Pieces of context information are combined and interpreted to deduce the situation that users or devices are currently involved in. This classification of the information is similar to that followed by [13], and allows us to accommodate learning-based techniques for the recognition of user activities like [11] within a specification-based approach.

A situation model comprises of three parts. First is the description of the situation itself, which identifies the features it contains. Features are externally observable properties that characterise some aspect of the situation. They could refer either to an individual entity or a collection of entities taking part in the situation. They describe *what* is recognised without any reference to *how*, providing a common interface to situation-aware applications and the other parts of the model.

Second, there are customisations, which place simple constraints on the features of the situation. The constraints are Boolean expressions that refer to either a single feature or a set of features. As they are specified with respect to the features of a situation, they can be applied to any situation with the necessary features allowing it to be easily tailored to address the particular needs of users. We can view the relationship between a situation and its customisations as a form of restriction inheritance [7].

Third, there are one or more specifications for the situation. These are complex documents describing the entities, properties, locations and constraints that are used to determine whether the situation is occurring. It is the specification that maps the context information, derived from low-level sensor data to the high-level features of the situation through a set of feature bindings. We adopt a specification-based approach as the transparency it provides facilitates these mappings. Taking inspiration from [3] we introduce structure into specifications through the specification of a set of roles played by individual entities involved in the situation. An entity (user, device, or location) is involved in a situation if it plays one or more of its roles, when it is occurring. An expression describes a series of constraints and relationships that need to apply on the roles. Each role in turn includes a similar expression that places constraints on any entity playing it. A situation is considered to be occurring when all of the relations and constraints included in its specification hold. Following a similar approach to [9], both roles and situations have a specific type that is defined with respect to a domain-specific ontology, providing a common vocabulary across environments.

The model also incorporates three types of inheritance between model parts that enable certain abstract reasoning over recognised situations. The first is situation inheritance, a form of extension inheritance [7], that allows a situation to introduce additional features to the inherited ones in its description. The second is feature inheritance, a form of subtype inheritance [7], that allows the introduction of subclasses of particular inherited features. The third is customisation inheritance, a form of restriction inheritance [7], that allows a customisation to be further customised with the introduction of additional constraints.

Figure 1 shows an example of two situation descriptions a 'Meeting' and a more specific 'Group Meeting' represented by dark blue rectangles. Each description includes a number of features, represented by light blue ovals. So, a meeting is characterised by the time and place it occurs and its attendees, while a group meeting in addition to these is also characterised by the group, and two types of attendees, group members and others. The solid arrow connecting the two situation descriptions indicates an instance of situation inheritance, while the dotted arrows indicate an instance of feature inheritance.

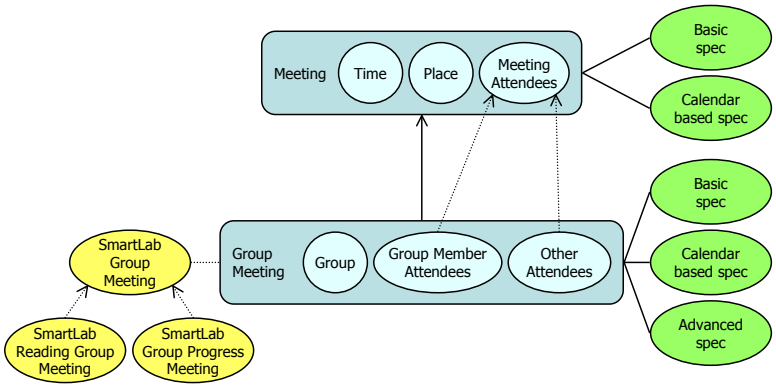

Fig. 1. The 'Meeting' and 'Group Meeting' situations, their specifications and customisations.

Customisations are shown as yellow ovals. For example, the 'SmartLabGroupMeeting' customisation constrains the group of a 'GroupMeeting' situation to the SmartLab group, with the dotted line connecting them indicating this relationship. More specific customisations can be created by placing additional constraints. For example, the 'SmartLab Reading Group Meeting' and 'SmartLab Group Progress Meeting' customisations constrain the 'Time' and 'Place' of the 'Group Meeting' in addition to its 'Group'. The dotted arrows indicate that these are instances of customisation inheritance.

Specifications are shown with green ovals and a solid line relates them to the situation they specify. A situation may be specified in a number of ways, each requiring different context information to recognise it. For example, a 'Group Meeting' can be recognised based on the time and the location of the attendees, 'Basic spec', or on calendar information and the location of the attendees, 'Calendar-based spec', or on the group membership of the attendees, 'Advanced spec'. Multiple specifications allow recognition to adapt to the currently available context information. Different specifications can be also created to suit different deployments. For example, light weight ones can be used when deployed on

resource-constrained mobile devices and more sophisticated ones when dedicated infrastructure is available. In addition, different specifications can be fused to increase confidence in the results.

4 A middleware for pervasive situation-awareness

The middleware needs to be open, as it must incorporate a variety of users and heterogeneous devices, whose number and identity are not known in advance and will change over time. The data describing the properties of users and devices, and new and customised situation specifications, are inherently distributed. Recognition of situations is a responsive process that must continually monitor changes in the environment and report the situations occurring. Situation-aware applications are adaptive, tailoring their behaviour to the current situation. Both recognition of situations and adaptation of application behaviour must be performed autonomously. Agent-based architectures with agents responsive to changes in their environment, that proactively act, and collaborate with other agents, support the flexible autonomous action necessary to satisfy these requirements [5], a fact that has already been recognised by others [6, 9].

The situation determination agent (SDA) is the agent that actually recognises situations. It continually attempts to match situation specifications and report instances of situations when they occur. There could be a number of SDAs deployed in an environment. To collect the context information necessary for the SDA to perform its function, a number of context entity agents are used (CEAs). The idea is that a number of different SDAs may be interested in the same kind of context information, so by having collections of agents that are managed by the middleware perform the context processing tasks, we can optimise the use of available resources. There are two flavours of CEAs - data (DCEAs) that detect and report individual properties of entities, and compute (CCEAs) that perform calculations over the properties and detect patterns of their values over time. The exchange of context information between CEAs and SDAs as well as between DCEAs and CCEAs is done over a publish/subscribe substrate. The CEA manager (CEAM) selects the most appropriate set of CEAs to provide the required context information taking into consideration the quality of the produced context information and the available resources.

Specification repository agents (SRAs) store collections of specifications and customisations. SRAs can either be hosted in the environment or on users' personal devices in order to ensure that the particular needs of users are satisfied everywhere. Situation-aware application agents (SAA) provide the interface to the middleware for situation-aware applications. They manage the applications' situation recognition requests and report to them occurrences of the requested situations. They are typically hosted on the device that hosts the situation-aware application. An SDA manager (SDAM) coordinates the interaction between SDAs and SRAs and selects the most appropriate SDA for each situation recognition request based on the resource requirements of the situation specification and the available resources at the device hosting the SDA.

In order to simplify the situation recognition process, each situation is considered to include an index. This is the entity from whose perspective the situation is viewed. Situation recognition requests are expressed with respect to a particular situation index. This turns situation discovery into an entity discovery process. To facilitate this process every entity in the system is represented by an index server agent (ISA) that keeps track of the SAAs that have requested situations involving this entity, and can additionally filter the information provided about an entity depending on the agent requesting it. An index locator agent (ILA) is responsible for the discovery of ISAs and is typically deployed alongside each SAA. Figure 2 illustrates the situation recognition process.

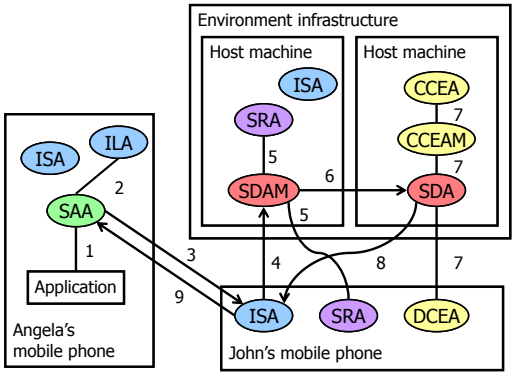

Fig. 2. An illustration of how the different types of agents cooperate to carry out the situation recognition process.

The SAA presents a simple situation query interface to situation-aware applications with three operations. The first is `getSituations` that allows situation-aware applications to discover which situations, customisations and specifications are supported in the local environment, particularly useful for runtime situation discovery. It can be parameterised to discover situations, customisations, specifications or any combination of these. The second is `requestSituations` that allows applications to request notifications for occurring situations. Its parameters include: the situation index, the target situation/customisation to be recognised, the required confidence by which the target situation must be recognised, the level of detail to be reported ranging from short (just the situation and its index), to features (the situations and all each features), to full (the situation, all its features, and all available details of the recognition process, including the specification used, the agents that were involved, etc), the lease period (i.e. the amount of time for which the query should remain active). The third is `unregister` that allows applications to deactivate a situation recognition request before its lease expires.

4.1 Middleware implementation

Situation descriptions, customisations and specifications are defined in a semantically rich manner with reference to relevant domain ontologies by using OWL documents[2]. OWL enables interoperability across different environments and is supported by a rich set of tools that can check the validity of documents, graphically edit and manipulate documents, and even reason about the relationships of concepts across different ontologies.

The agent-based architecture is implemented using the Java Agent DEvelopment Framework (JADE)[3]. It supports suitable abstractions for the development of agents and agent behaviours. It provides support for agent discovery, and communication over both wired and wireless networks. It is able to handle networking volatility through store and forward techniques, and supports multiple device platforms, from server-class machines to mobile phones.

The situation determination agent (SDA) implementation is based on the Jess Rule Engine realisation of the Rete network algorithm[4]. The SDA is also responsible for transforming situation specifications into Jess code and determining the required entity properties. There are three main steps to the translation. The first is to identify all the properties that are referenced in the specification's expression and feature bindings. The set of properties determines which DCEAs and CCEAs are required to recognise the specification. The second step is to identify the roles that appear in the specification, and extract any cardinality constraints that are placed upon them as these require special processing. The third step is to generate Jess code for the specification. Each Jess rule comprises of an 'if' part that lists all of the patterns that must match for the rule to fire, and a 'then' part that lists all of the actions that are executed when the rule is fired. In this case the focus is on the 'if' part that concatenates (a) expressions that match against each of the roles present in the specification, (b) a list of the tuples and expansions required to access the necessary properties, and (c) the expressions of the specification. A very similar process is used for the transformation of role specifications. The full details of the process including an example transformation can be found in [12].

The middleware implementation also includes a number of optimisations to better utilise available resources. First, it distinguishes between generic situation and role specifications, and grounded ones, i.e. those that rely on entities properties whose value is produced by agents residing on the same device. For the latter it is wasteful to assign their recognition to a remote SDA, so an SDA is preferably deployed on the same device in order to reduce network communication. Second, SDAs classify specifications of interest as either active or inactive. The former are those for which all required entity attributes are currently available through corresponding CEAs, and can thus be identified. As entities appear and disappear from the environment so do their associated CEAs resulting in the activation and de-activation of the corresponding specifications.

[2] http://www.w3.org/ TR/2004/REC-owl-features-20040210
[3] http://jade.tilab.com
[4] http://herzberg.ca.sandia.gov/jess

5 Middleware evaluation

Our evaluation first focuses on showing that the middleware is expressive enough to easily support multiple specifications and a wide range of customisations for a set of situations, and thus is able to support the needs of individual users across potentially different environments. Then we focus on showing that the middleware performs well. In this respect, the quality of the recognition process is not really relevant as it depends on the specific situation specification used and the quality of the underlying sensing infrastructure, both of which are not determined by the middleware itself. Instead, we focus on its ability to support interactive situation-aware applications as the number of entities involved in the situation, and the number of situations to be recognised increases. The roud-trip time of a situation recognition request provides a summary of the overall performance of the system.

5.1 Expressive power

To demonstrate the expressive power of our middleware we focus on meeting, presentation and lecture situations in a university setting. The blue ovals represent situations, the yellow ovals customisations, the green ovals specifications, and the solid arrows indicate situation inheritance relationships. The capital letters in the specification ovals indicate the types of CEAs used to implement the specification (see [12] for more details).

Figure 3 shows a generic 'Meeting' situation extended by a 'Group meeting' that recognises meetings of particular research groups, a 'PhD meeting' that detects a meeting between a PhD student and his/her supervisors, and a 'Demonstrator meeting' that recognises a meeting held between the lecturers and demonstrators of a particular class. Each extended meeting situation has at least three alternative specifications. The first two are the ones we saw in figure 1. The third is tailored to the particular type of meeting, for example part of this specification for the group meeting required that more than 75% of the attendees are members of the same group, while for the PhD meeting, part of this specification detects PhD supervisor and student relations between the attendees.

Figure 4 shows a presentation situation with five different specifications, two specifications similar to those described above, one based on a fixed time, place and attendees' locations, and another based on attendees' locations and a calendar entry. There is also a coarse-grained specification that can only recognise presentation attendees, and two fine-grained specifications that could differentiate between speakers and audience members of a presentation, one based on fine-grained location information (i.e. based on the area of the room a person is located), and the other based on detecting whether a person is speaking or not. A lecture situation is also identified by five specifications similar to the presentation ones, with the different attendees now being the lecturer and the students of a class.

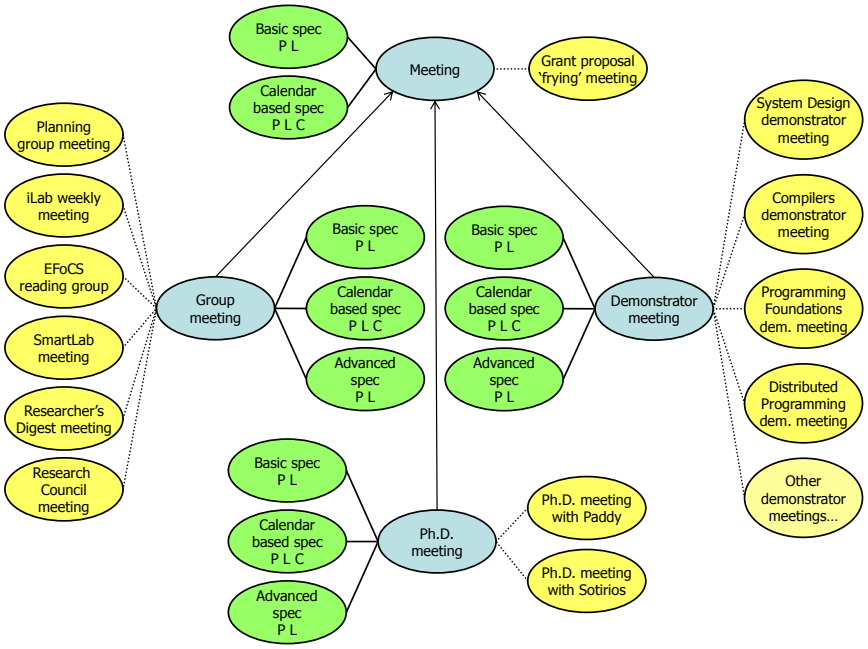

Fig. 3. University meeting situations.

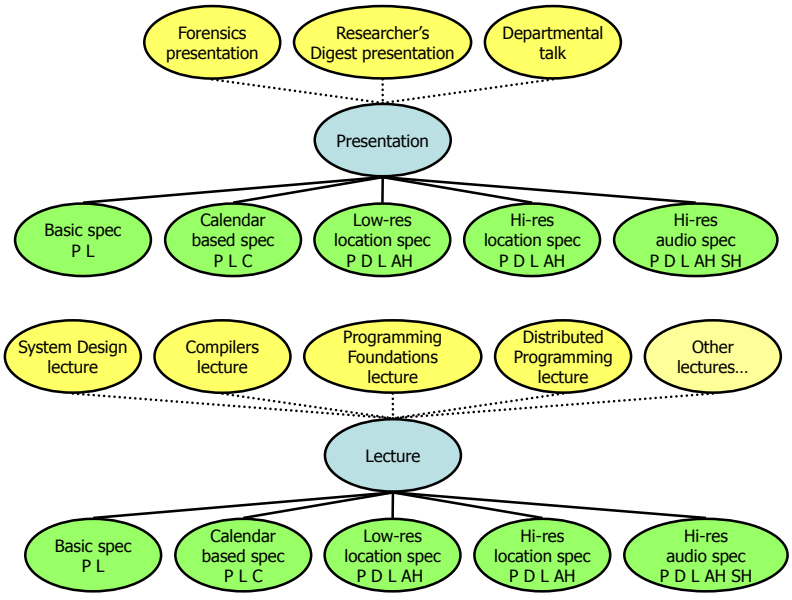

Fig. 4. University presentation and lecture situations.

Further customisation to the particular organisational setting is carried out through several customisations for each of the meeting, presentation and lecture situations. For example, customisations of the group meeting situation are created for each of the research and administrative groups within a particular university department, as well as customisations for each of the lectures, labs and tutorials for each course run by the department. This offers a large degree of flexibility in tailoring situations to the particular needs of the department's staff and students.

5.2 Performance

The performance measurements are based on a deployment of the middleware in a university environment. Three different classes of computers were used: (a) desktop PCs with a 3.4 GHz Intel Pentium 4 processor and 1 GB of RAM, running Debian GNU/Linux 3.1, (b) a Dell Inspiron 6000 laptop, with a 1.5 GHz Intel Pentium M processor and 1 GB of RAM, running Windows XP, and (c) a collection of HP Pocket PC h5500 handheld computers with a 400 MHz Intel XScale processor and 128 MB RAM, running Windows CE 4.2. The desktop PCs were connected to a 100 Mbps-BaseT Ethernet network, while the laptop and Pocket PCs were connected via a WEP-encrypted 802.11b wireless network.

The performance study focuses on the round trip time of a situation recognition request (RTT). It incorporates the time consumed by sending the situation recognition request out, the system gathering the necessary information required to recognise the situation, the cost of performing the recognition itself, and the delivery of the results back to the application. Each RTT measurement represents the mean value of 300 samples, which reflect a single situation recognition session where an application first joins the network, issues a situation recognition request, receives 10 situation reports, and reacts to the situation information. A total of 30 sessions were collected.

We first focus on the middleware performance for a set of situations as the number of entities involved in them increases from typical to larger sizes. The measurements were collected using a situation-aware application that could record the times that events such as sending situation recognition requests and receiving situation reports occur. One SDA was run on a desktop PC and another desktop PC hosted the other infrastructure agents. Pocket PCs were used to represent users, and hosted the CEAs that reported the user's location and personal profile information, as well as their ISA. The measurement application was also run on a Pocket PC.

The RTT measurements for each situation are shown in figure 5. The size values represent the number of users involved in the situation. For larger sizes a single Pocket PC represented more than one person. Overall, each of the situations shows an acceptable round trip time. Even at the largest situation size, all response times are under nineteen milliseconds. For each doubling in situation size, the round trip time increases by only 10% on average. This means the middleware is able to support interactive situation-aware applications.

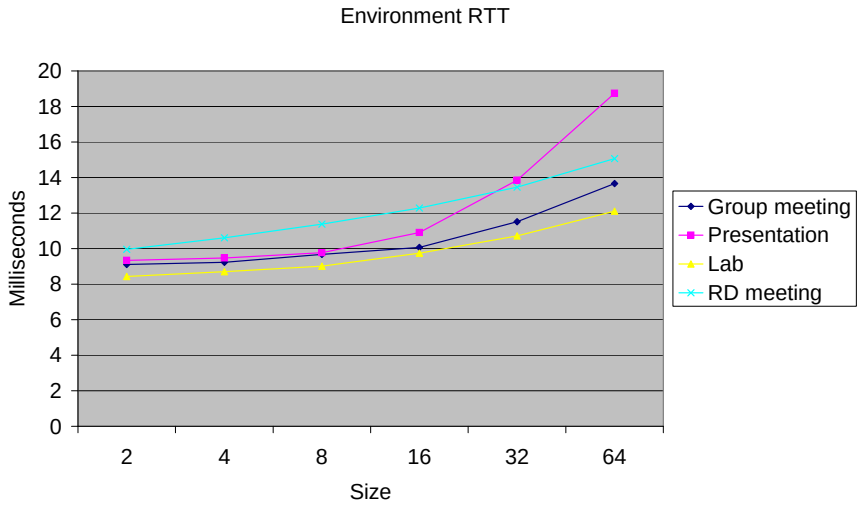

Fig. 5. Mean RTT for a selection of situations as the size of the situations increases.

We now focus on the performance of the middleware as the number of situations simultaneously recognised increases from light to heavy loads. The change in RTT is measured for a particular situation, while the number of instances of other situations being recognised in the environment increases. The 'Presentation' situation is chosen as it showed the steepest RTT scaling in figure 5 above. This is due to the fact that it has the greatest number of distinct roles. For these measurements, a slightly different set up was used. For the cases where the number of users involved exceeded the available Pocket PCs, we simulated additional users on desktop PCs. However, no simulation is involved in the 'Presentation' situation itself as the number of participants is limited to eight. Otherwise, the set-up is the same as above.

The RTT measurements are shown in figure 6. At scale 1, there was a single instance of a number of group situations that included a 'Lunch', 'Meeting', 'Group Meeting', 'Demonstrator Meeting', 'Lecture', 'Lab' and 'Tutorial' situations. Each instance included eight people, matching the size of the 'Presentation' instance. Each scale number reflects a multiple of this set of situations. At each scale, the single, realistic instance of the 'Presentation' situation was measured.

Figure 6 shows that RTTs increase more steeply as the number of simultaneously recognised situations increases, than when the size of the situation increased. For each doubling in the number of situations simultaneously recognised, the RTT more than doubles on average. However the level of performance is still acceptable. At the heaviest load, while recognising forty situations simultaneously with a single SDA, the average RTT was just 387 milliseconds. These times can be reduced by sharing the recognition load across multiple SDAs. Note

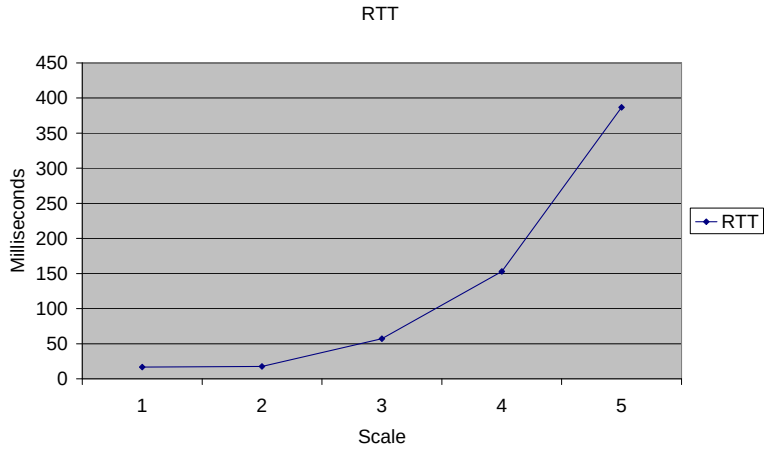

Fig. 6. Mean RTT for the 'Presentation' situation as the number of simultaneously recognised situations increases.

that load distribution is automatically managed by the middleware. To demonstrate this, the largest scale set up was redeployed, with the number of available SDAs increased, and the reduction in the RTTs was measured. Each SDA processes an equal share of the total number of situations. For 2 SDAs the mean RTT was 231.829 milliseconds, while for 4 SDAs the mean RTT was 144.112. These represent a decrease of the RTT by more than half, 61%, on average for each doubling of the available SDAs. So, employing multiple SDAs can help to maintain a low RTT in periods of heavy load.

6 Conclusions and future work

In this paper we presented a middleware for pervasive situation-awareness. The middleware is able to recognise the same situations across different environments by supporting multiple specifications, each targeting a particular sensing infrastructure. It is also able to accommodate the particular needs of individual users by supporting easy customisation of situations. As customisations are expressed in terms of the features of a situation, they can be applied across a variety of situation specifications and the resulting situations can be identified across different environments. We also showed through a number of situations that our middleware is expressive enough to easily support the definition of a wide range of situations, customisations and specifications. Moreover, we demonstrated that its performance scales reasonably to support practical deployment scenarios.

In the future we plan to extend our middleware along two dimensions. First, we plan to enhance our model by exploring more sophisticated forms of situation customisation beyond restriction inheritance. In this respect, we are looking

into the specification of sophisticated interpreters to apply more expressive customisations. Second, we plan to enhance the performance of our middleware implementation by exploring cloud-based deployment scenarios. In this front, we can off-load large parts of the rule matching involved in the situation identification process in a cloud platform like the Google App Engine, however this would require careful consideration of the performance trade-offs involved.

References

1. Albinali, F., Davies, N., Friday, A.: Structural learning of activities from sparse datasets. In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07). pp. 221–228 (2007)
2. Cook, D.J., Das, S.K.: Pervasive computing at scale: Transforming the state of the art. Pervasive and Mobile Computing 8(1), 22–35 (2012)
3. Coutaz, J., Crowley, J.L., Dobson, S., Garlan, D.: Context is key. Communications of the ACM 48(3), 49–53 (2005)
4. Henricksen, K., Indulska, J.: Developing context-aware pervasive computing applications: Models and approach. Pervasive and Mobile Computing 2(1), 37–64 (February 2006), http://dx.doi.org/10.1016/j.pmcj.2005.07.003
5. Jennings, N.R., Wooldridge, M.J.: Applications of intelligent agents. In: Agent Technology: Foundations, Applications, and Markets, pp. 3–28. Springer-Verlag: Heidelberg, Germany (1998)
6. Kumar, M., Shirazi, B.A., Das, S.K., Sung, B.Y., Levine, D., Singhal, M.: Pico: A middleware framework for pervasive computing. IEEE Pervasive Computing 02(3), 72–79 (2003)
7. Meyer, B.: The many faces of inheritance: A taxonomy of taxonomy. IEEE Computer 29(5), 105–108 (1996)
8. Patterson, D.J., Liao, L., Fox, D., Kautz, H.A.: Inferring high-level behavior from low-level sensors. In: Ubicomp. pp. 73–89 (2003)
9. Ranganathan, A., Al-Muhtadi, J., Campbell, R.H.: Reasoning about uncertain contexts in pervasive computing environments. IEEE Pervasive Computing 3(2), 62–70 (2004)
10. Roy, N., Roy, A., Das, S.K.: Context-Aware Resource Management in Multi-Inhabitant Smart Homes: A Nash H-Learning based Approach. In: Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2006). pp. 148–158 (2006)
11. Tapia, E.M., Intille, S.S., Larson, K.: Activity recognition in the home using simple and ubiquitous sensors. In: Ferscha, A., Mattern, F. (eds.) Pervasive Computing, Second International Conference, PERVASIVE 2004. pp. 158–175. Springer, Vienna, Austria (April 18-23 2004)
12. Thomson, G.: A Model and Architecture for Pervasive Situation Determination. Ph.D. thesis, University of Strathclyde (2010)
13. Ye, J., Coyle, L., Dobson, S., Nixon, P.: Using situation lattices in sensor analysis. In: PerCom. pp. 1–11 (2009)
14. Ye, J., Dobson, S., McKeever, S.: Situation identification techniques in pervasive computing: A review. Pervasive and Mobile Computing 8(1), 36–66 (2012)