On representing situations for context-aware pervasive computing: six ways to tell if you are in a meeting

Seng W. Loke
Caulfield School of Information Technology
Monash University, VIC 3145, Australia
swloke@csse.monash.edu.au

Abstract

Context-aware pervasive systems are emerging as an important class of applications. Such work attempts to recognize the situations of entities. This position paper notes three points when modelling situations: (1) there can be multiple ways to represent a situation; (2) a situation can be viewed as comprising relations between objects and so recognizing a situation boils down to determining if a prescribed set of such relations hold or not hold at that given point in time; and (3) situations can be represented in a modular manner, emphasizing an incremental approach and reuse when building a knowledge base of situations. We also present a declarative approach to building context-aware pervasive systems, where relations that are in a situation can be recognized.

1 Introduction

Context-aware pervasive systems are emerging as an important class of applications. Such work attempt to recognize the situations of entities and intelligently respond to them. These systems typically utilize a collection of sensors to perceive the physical world (or some aspect of it as described by a collection of context attributes) and sensor readings must be aggregated and interpreted appropriately. Applications range from context-aware mobile services, context-aware mobile phones and appliances to context-aware homes.

A natural way to program context-aware behaviour is to use rules which map a recognized situation to some given action. But a preceding question is how does one describe and represent the situations that such a system should recognize? For instance, if I were building a context-aware phone, I would like the phone to behave appropriately in certain situations - the phone could somehow detect a situation via some combination of sensors and then switch itself to an appropriate mode (e.g., see that I am in a meeting and put itself to silent mode). One could enumerate a set of typical situations (or situation types) which the phone can be in and have rules to act appropriately in those situations. There would be a need to have some formalism to represent these typical situations in terms of readings from sensors - we are in effect labelling a collection of sensor readings with an interpretation that they represent some situation.

In this paper, we explore an approach to recognizing and reasoning with situations from the perpective of knowledge engineering. We (as a domain expert) create explicit representations of situations and reason with them. Because situations are recognized via values for context attributes acquired via sensors, our representation describes situations by relating context attributes and sensors.

We outline our representation formalism for situations in Section 2. Then, Section 3 examines the case study of representing the situation of a meeting, and explores underlying issues. Here, we also point out that the same situation can be represented in many different ways, just as there are many different ways to tell if a particular situation is occurring (or if an entity is currently in a given situation). Section 4 concludes the paper.

We assume a basic knowledge of the logic programming language Prolog below.

2 Representing Situations: Situation Programs and LogicCAP

We use the operational (and arguably broader) definition of context from [3]: *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."* Related to the notion of context is the notion of the situation. Work such as situation theory [1] and the situation calculus consider the primacy of the situation abstraction and noted that an agent (e.g.,

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

human) is able to individuate a situation. According to [2], a situation is a

> *"structured part of reality that it (the agent) somehow manages to pick out",*

by "direct perception of a situation, perhaps the immediate environment, or thinking about a particular situation," and "individuation of a situation by an agent does not (necessarily) entail the agent being able to provide an exact description of everything that is and is not going on in that situation." The idea is that the situation abstraction allows one to effectively "carve the world up" into manageable pieces which a collection of sensors of a system might recognize and respond to. It might also be possible to compose such pieces to construct more complex models of situations.

Our way of representing situations take into account the structure of a context-aware system as comprising sensors at one level and inference procedures to reason with context and situations at another level. We also consider how to manipulate situations as first-class entities and how to reason with our representation of situations within a logic programming language. We clearly differentiate between sensor readings, context, and situation in our model. Our model is also declarative and based on logic programming ideas and was first put forward in [5], and idea of a situation as a collection of relations comes from [1].

There are many ways to describe a situation depending on the application. Here, we involve *sensors* in our definition of situation. We use a broad definition of sensor, which is taken to mean not only temperature, heat or motion sensors but any device or mechanism that is used to provide contextual information. This means that a positioning engine (that provides location information about a device and user) is also called a sensor. The clock in the computer with its associated operating system call can be considered a sensor if it used to return time which is used as contextual information in an application.

2.1 The Situation Program

Let each sensor be represented by a *sensor predicate* of the form:

⟨sensor_id⟩*(⟨inputs⟩,⟨output readings⟩).

The output from a sensor is represented by a variable, and inputs to sensors by parameters. Then, a situation program S is defined as a collection of rules (or a logic program), which we call a *situation program*, each rule of the form:

$$\mathcal{A} \texttt{ -e-> } \mathcal{G}$$

where "-e->" denotes "is a possible explanation for" (i.e., abductive rules), and $\mathcal{G}$ is given by

$$\mathcal{G} \ ::= \ \mathcal{A} \mid \mathcal{S} \mid (\mathcal{G},\mathcal{G}) \mid S\texttt{*>}E$$

$\mathcal{A}$ is an atomic goal formula (an ordinary Prolog-style term), $\mathcal{S}$ is a sensor predicate, "," denotes conjunction, S is a situation identifier and E is an entity (e.g. user, device, or software agent) identifier. We call the operator "in-situation" denoted by "*>". A goal of the form S*>E, read as a query "E in situation S?", is a meta-level goal which succeeds if *the contextual information about E is provable from S* in the way we describe later. Because S represents clauses (facts and rules that would hold) about the situation, the intuition of this operator is that E is in the real world situation represented by S if the contextual information about E holds in S. There is one distinguished rule (which we call the *situation rule*) whose premise is a predicate naming the situation and, optionally, have a parameter denoting the entity.

The rules of a situation program permit natural representation of a situation, i.e. if a situation occurs, then certain conditions and constraints should hold. As an example, we can define a `in_meeting_now` situation as follows. The sensor predicates are `location*(E,L)` which returns the location of an entity `E` in variable `L`, `diary*(E, Event, entry(StartTime, Duration))` which returns diary entries for entity `E` for a matching `Event`, `people_in_room*(L,N)` which returns the number of people at a location, and `current_time*(T)` which takes no inputs and returns the current time in a variable. The constraints the situation imposes on such sensors' readings can then be modelled by the following logic program:

```
situation program meeting1:
in_meeting_now(E) -e->
 with_someone_now(E),
 has_entry_for_meeting_in_diary(E).
with_someone_now(E) -e->
 location*(E,L),
 people_in_room*(L,N), N > 1.
has_entry_for_meeting_in_diary(E) -e->
 current_time*(T1),
 diary*(E,'meeting',
    entry(StartTime,Duration)),
 within_interval(T1, StartTime, Duration).
```

The program is viewed as a constraint in the sense that if the entity is in that situation, various relationships as specified above should hold.

The idea of the "-e->" goal is an abductive reading, where the situation is viewed as an explanation for sensors (in our broad view as any mechanism which returns context information, from accessing diary entries to smoke detectors or Berkeley motes) being observed to have certain values. The occurrence of a situation is viewed as causing, yielding, or explaining the sensory observations.

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

2.2 Modularity

Our syntax of rules allow situation programs that refer to other situation programs. The above program might be rewritten as follows.

```
in_meeting_now(E) -e->
  with_someone_now*>E,
  has_entry_for_meeting_in_diary*>E.
```

where `with_someone_now` is a situation with its own situation program containing the rule `with_someone_now(E) -e-> location*(E,L),people_in_room*(L,N), N > 1`, and similarly, the situation `has_entry_for_meeting_in_diary`. The advantage of being able to split rules into separate situation programs is modularity which encourages reuse.

Because we represent situations as explanations for observations, the procedure for evaluating the in-situation goal is by forward-chaining over rules in situation programs as described in [5].

In other words, if we observe that E is with someone now and E has an appropriate entry for a meeting in E's diary, then E being in a meeting now is a (in this case, highly!) possible explanation for these observations.

3 Five Other Ways to Represent a Meeting

We have shown one way to represent the situation of a meeting occurring. In general, if we use a different set of sensors, we can define a different situation program for a meeting.

Five other possibilities are given as follows, with different assumptions about the situations, the sensors used, and what sensory information can be obtained (e.g., what objects or people are being tracked):

1. *co-location of filled coffee cups in a room*, as inspired by [4]: we assume a database of coffee cups whose location is tracked by a positioning technology, retrieved, when given a name, using the predicate `has_coffee_cup/2`, and a database of employees and their colleages, retrieved, when given a name, using the predicate `has_colleagues/2`. We have the following rule which says that if `E` is in a meeting with at least one other colleague then `E`'s coffee cup and at least one of `E`'s colleagues' coffee cups are co-located in the same room, and are warm (above 50 degrees Celsius):

   ```
   situation program meeting2:
   in_meeting_now(E) -e->
     has_colleagues(E, Fs),
     member(F, Fs),
     has_coffee_cup(F, CF),
     has_coffee_cup(E, CE),
     location*(CE, Room),
     location*(CF, Room),
     temperature*(CF, TCF),
     temperature*(CE, TCE),
     TCF > 50, TCE > 50 .
   ```

   The rule works on real-world assumptions about coffee usage at meetings. Another view is that the above rule defines a specific kind of meeting where people use coffee.

2. *weight sensors on the floor*: we assume a weighing machine on the floor of a room which gives the total weight of objects, including people, on it. A rule such as the following states that if a meeting is occurring in the room `E` is located in, then the weighing machine of the room would have a reading above some threshold.

   ```
   situation program meeting3:
   in_meeting_now(E) -e->
     location*(E,Room),
     floor_weight_machine*(Room,W),
     W > 200.
   ```

3. *devices in the room*: similar to [6], we assume that if a meeting is going on, lights will be on in the room, Powerpoint is running on the PC in the room, and the projector is working. The following rule captures this idea about the room that a person `E` is currently in:

   ```
   situation program meeting4:
   in_meeting_now(E) -e->
     location*(E,Room),
     projector*(Room,switched_on),
     room_light*(Room,switched_on),
     pc_software_applications*(Room,
               powerpoint,running).
   ```

   We have assumed predicates that will return the status of devices and PC applications.

4. *sounds and noises*: we assume the presence of microphones in the room which measure noise levels. Our assumption is that compared to other times where noise levels are generally low, the noise level when a meeting is going on will be significantly higher. Noise levels can be measured by a noise dosimeter[1] worn by a person or a sound level meter situated in a meeting room of interest. We can perhaps do better by matching sounds with voice patterns of speakers with the

[1] See http://www.ccohs.ca/oshanswers/phys_agents/noise_measurement.html

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

person E of interest to see if this person is present in the meeting, and perhaps even analyzing the person's speech by noting keywords. Assuming a dosimeter worn by a user (say, by a person E), the following rule states that the person is in a meeting if the noise level as detected by a dosimeter worn on the person is above a certain level averaged over a period of time (say measured over 5 minutes since the time of the start of the query to determine if E is in a meeting), and the average sound readings from a sound meter in the room is above a threshold for a similar period of time (such a threshold serves to distinguish the meeting situation in the room from other times when it is typically quiet in the room) - some calibration of the meters are required to determine the thresholds:

```
situation program meeting5:
in_meeting_now(E,
  PersonalMeetingNoiseThreshold)
-e->
  dosimeter*(E,5,AvgNoiseLevel),
      % average noise level
      % returned after 5 minutes
      % of measurement
  AvgNoiseLevel >
    PersonalMeetingNoiseThreshold,
  location*(E,Room), % check noise
  %level in the room the person is in
  meeting_room(Room,
    RoomMeetingSoundThreshold),
     % ensure that the room is
     % a meeting room and
     % retrieve the threshold
  sound_meter*(Room,5,AvgSoundLevel),
     % average sound level
     % measured over 5 minutes
  AvgSoundLevel >
    RoomMeetingSoundThreshold.
```

5. *use cameras*: cameras can be used to detect the presence of people in a meeting room using a technique called *background substraction*.[2] This technique can be used to "watch" meeting rooms for activity.

```
situation program meeting6:
in_meeting_now(E) -e->
  location*(E,Room),
  people_present*(Room).
  % this is done by a camera
  % watching the room
```

Observations. We make the following observations based on the above examples:

[2] See http://www.flong.com/writings/texts/essay_cvad.html

- *multiple representations*: we note that the same situation can be represented in multiple ways and that we can combine multiple situation programs in modelling a given situation. For example, we can employ not only one of the above situation programs (and the related sensors) for determining a situation, but several of such programs (and their related sensors) to yield a higher degree of certainty.

- *inadequacy of representations*: each situation program captures some aspect of the situation of a meeting occurring, but each might be viewed as not totally conclusive, i.e. if all the relationships specified in a particular situation program holds, one could guess that a meeting is occurring to a high degree of certainty but not with absolute certainty. Hence, we view the above situation programs abductively: a meeting occurring is a possible explanation that the relationships specified in a particular situation program are being observed to hold.

- *modular representations*: because each situation program contains a set of relations which should hold given that a situation occurs, one can devise different situation programs (containing different sets of relationships) for modelling situations. As we have seen with the in-situation operator, one situation program can refer to others. Also, one can build sophisticated representations of (more complex) situations in terms of an existing repertoire of situations.

- *design patterns*: from the software engineering perspective, there is indication from the above examples that design patterns can be developed for recognizing situations. Taking a situation program, one can elaborate on it to spell out particular sensors to be used and what reasoning techniques might be employed to (in-)validate (with respect to the real world) each of the mentioned relationships. Based on the same template of a situation program, different reasoning techniques might be employed to recognize different relationships in different applications. For the same situation program, different sensors and underlying technology might also be employed according to what is available, i.e. situation programs can represent situations at a reasonable level of abstraction, decoupled from underlying technologies.

- *programming with situations*: the situation programs can be embedded into logic programs using metaprogramming, and reasoned about within a declarative framework.

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

4 Conclusion and Future Work

The paper advocates a high level explicit representation of situations for the purposes of context-aware pervasive computing, approached from the traditional knowledge engineering perspective. Our proposed LogicCAP formalism is merely illustrative, not prescriptive - other approaches and formalisms can be employed but which, we contend, should retain the spirit of this approach. Benefits of the approach includes enabling abstraction from underlying sensor technologies, modularity and reuse of representations, and meta-programming style manipulation of situation representations.

Future work involves developing the design pattern language for situations, and applying the techniques to model more applications.

References

[1] J. Barwise and J. Perry. *Situations and Attitudes*. Cambridge: MIT-Bradford, 1983.

[2] K.J. Devlin. Situations as Mathematical Abstractions. In J. Barwise, J.M. Gawron, G. Plotkin, and S. Tutiya, editors, *Situation Theory and its Applications*. CSLI, 1991.

[3] A.K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing Journal*, 5(1):5–7, 2001.

[4] H-W. Gellersen, A. Schmidt, and M. Beigl. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. *Mobile Networks and Applications (MONET)*, 7(5):341–351, 2002.

[5] S.W. Loke. Representing and Reasoning with Situations for Context-Aware Pervasive Computing: a Logic Programming Perspective. *The Knowledge Engineering Review*, 19(3):213–233, 2005.

[6] A. Ranganathan and R.H. Campbell. An Infrastructure for Context-Awareness Based on First Order Logic. *Personal and Ubiquitous Computing Journal*, 7:353–364, 2003.

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