Self-Adapting Context Definition

Neil O'Connor, Raymond Cunningham and Vinny Cahill
Distributed Systems Group
Trinity College Dublin, Ireland
{neil.oconnor, raymond.cunningham, vinny.cahill}@cs.tcd.ie

Abstract

Many approaches to context-aware computing have focused on providing means for developers to define application contexts. In these approaches, correct application behavior depends on the developer defining the right contexts. The application cannot adapt incorrectly defined contexts as they are statically defined by the developer. We propose a method by which an application can use feedback to evaluate its contexts and adapt them where necessary. This results in more accurate context definitions and should lead to improved application performance. We discuss how this is achieved using Q-learning, and present a scenario and experimental results to support our approach.

1 Introduction

The growth of context-aware computing has seen a move away from the development of "one-off" applications that use context, toward the provision of generic solutions such as middleware and toolkits that support defining and reasoning about context [6], and ontologies that can be used to describe contexts [4]. Using all these approaches relies, to varying degrees, on the developer to provide knowledge to the application.

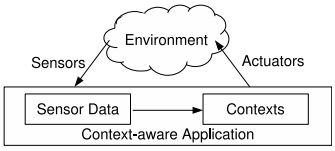

Figure 1. A context-aware application

Context information is often derived from sensor data (Fig. 1). Contexts can then be described as *abstractions* of environmental situations recognizable by sensors. By environmental situations we mean distinct states in which the environment can be. For example, office applications (e.g. ParcTab [1]) use location-sensor data to infer contexts. A context such as "in a room" is an abstraction of many distinct positions within the room. In particular, contexts should be abstracted from situations that share the same *meaning* to the application.

Typically developers are responsible for defining contexts. They use their knowledge of the operating environment, sensors and actuators to define relevant contexts. Sets of sensor readings are mapped to corresponding contexts.

For linear sensor data [12] the developer does not need to explicitly define each sensor reading relevant to a context. Instead the limits of a context can be defined, e.g. "room temperature" abstracts readings from 16-22^{o}C. This is possible for many sensors such as distance, location, weight, velocity etc.

Machine Learning approaches reduce the dependency on the developer to define correct behavior; by allowing an application improve itself. These approaches use abstractions in order to scale to real world problems, and function approximation has been applied to learning these abstractions [13]. Function Approximation is a supervised learning technique [13] so it relies on correct examples to learn from [10]. The developer selects these examples, so similar to context approaches there is a dependency on developer knowledge when defining abstractions.

We identify two issues with developer-dependent approaches to context/abstraction definition:

Inaccurate Contexts: If the developer has incorrect or incomplete knowledge of the domain then contexts may be poorly chosen, e.g., "room temperature" = 10-15^{o}C when it should be 16-22^{o}C. In addition, the dynamic nature of the real world may result in accurate contexts becoming inaccurate, e.g. in the summer room temperature might be regarded as 18-25^{o}C. Inaccurate contexts lead to incorrect behavior.

Inflexible Context Definitions: Developer-defined contexts imply fixed relationships with sensor data. The application is only aware of the implication of being in a context, rather than why the abstraction is appropriate. It cannot reason about the suitability of the abstraction, so it cannot

First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)
0-7695-2906-2/07 $25.00 © 2007 IEEE

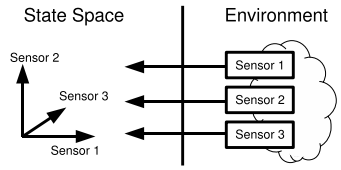

Figure 2. Application perception

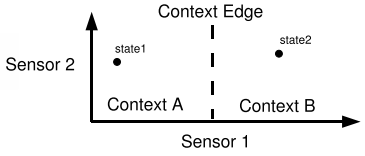

Figure 3. Contexts in the state space

adapt it to correct it or even identify that it is incorrect.

Using our approach the application adapts its context definitions to address these issues. We define a meaning for sensor data that is relevant to the application. The application uses feedback to learn this meaning and identifies contexts based on it. By defining more accurate contexts the application should perform better.

2 Application-Centric Context

There are a variety of definitions for context. They define types of information that make up context [3, 11], or how it is relevant to an application [5]. We consider context to be an abstraction of situations that have similar meaning (Section 1). In order for an application to abstract contexts it must have some notion of meaning for sensor data.

2.1 Meaning for Sensor Data

In [8] Nehaniv argues that the meaning of information is revealed in how it influences an agent. A discussion of the meaning of sensor data in [9] also suggests that the meaning of something is grounded in how it is used by the entity interpreting it.

Brown [3] states that context-aware applications change their behavior according to the context. If contexts adapt behavior and contexts are abstracted from sensor data, then it logically follows that sensor data adapts behavior. Based on this usage we define application-centric meaning as a mapping *M* from sensor data to application actions

$$M : Sd \rightarrow A$$

where *Sd* is the set of possible sensor values and *A* is the available actions. Feedback is used to identify this meaning, and the process is dependent on an underlying learning technique.

2.2 Context Definitions

The application perceives environmental situations through its sensors. We have chosen to consider sensor data as a state space in order for the application to reason about and compare situations, and abstract contexts.

The combined readings of an application's *n* sensors form a unique point in n-dimensional space. Figure 2 shows an application with three sensors. When combined the sensor readings form a triple (sensor1_value, sensor2_value, sensor3_value) that uniquely identifies a situation in the environment. Situations are distinguished at the highest resolution offered by the sensors. We assume that sensor data is linear [12], which allows the proximity of states within the state space to be calculated using a metric.

From the perspective of the application, a context is a group of states within the state space that share the same meaning. Due to our linear state space it is our intuition that similar states will be close to each other. In this case contexts emerge as continuous spaces within the state space.

Contexts are defined in terms of *context edges*. A context edge is the border between two contexts. Atkin [2] explores the issue of state definition in continuous search spaces, and proposes that only states at which a decision is made are needed. Our context edges are equivalent to these points, as a context change causes a change in behavior. If neighboring states within the state space have dissimilar meaning then we say that a context edge exists between them.

Figure 3 shows a two-dimensional state space. *Contexts A* and *B* are regions within the state space, separated by a context edge. All of the states in *Context A* have the same meaning as *state1*, and all of the states in *Context B* have the same meaning as *state2*. *Context A* is defined as any state to the left of the context edge, and *Context B* is defined as any state to the right. The meaning of a context is the same as that of the states it abstracts; it influences the behavior of the application in the same way as each of its states.

By defining contexts in terms of context edges we make our context definitions flexible. The application identifies context edges, and uses their locations in the state space to evaluate and update existing context definitions.

3 Implementation

There are two main phases to the implementation. Meaning is associated with sensor states, and contexts are subsequently abstracted based on the meaning. For this implementation we used Q-learning to identify meaning.

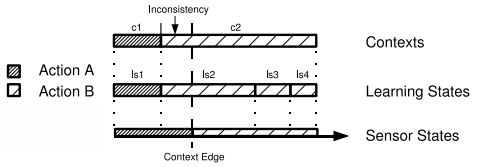

Figure 4. Inconsistent learning state

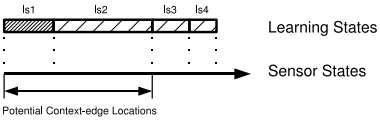

Figure 5. Locating context edges

3.1 Adding Meaning to Sensor States

Q-learning [13] is a Reinforcement Learning algorithm that uses feedback to learn appropriate behavior. The aim of the learning algorithm is to find the optimal action for each of a set of states. To achieve this the algorithm receives a reward from the environment when an action is taken in a state. By exploring all combinations of states and actions the application learns the optimal action for each state- a policy for optimal behaviour.

In our implementation the policy maps states in the sensor state space to actions. This is the same as our notion of application-centric meaning (Section 2.1). The policy defines the meaning of states to the application.

Our metric for comparing states is based on this policy, e.g. if states *s1* and *s2* have different optimal actions we say they have dissimilar meaning.

3.2 Scaleable Learning

Learning becomes infeasible in large state spaces [10] so we cannot apply Q-learning to all states in the state space. We introduce another layer of abstraction between the sensor state space and contexts called *learning states,* to reduce the number of states to be learned about.

Figure 4 shows the two levels of abstraction. Contexts are expressed on learning states rather than directly on the sensor states, so the accuracy of context definitions depends on the accuracy of the learning states. A learning state accurately represents the sensor states below it if the appropriate action is consistent across the learning state.

Figure 4 illustrates a situation where the learning states are not accurate. *ls2* is an *inconsistent* learning state as it contains two distinct regions where different actions are appropriate. The size of the inconsistency is the distance between the context edge and the nearest learning-state boundary. A context edge based on these learning states is inaccurate by this amount.

Some Reinforcement Learning approaches have been extended to include state-space adaptation [7]. They adjust their states to maximise performance. We use a similar approach to evolve an accurate set of learning states.

Our focus is on identifying context edges. Therefore our algorithm focuses on increasing the accuracy of learning states in areas of the state space where there is a context edge. To identify these areas neighboring learning states are compared using our similarity metric (Section 3.1). A context edge exists somewhere in the state space represented by neighboring, dissimilar learning states.

Figure 5 shows an area that contains a context edge. The states *ls1* are *ls2* are dissimilar, therefore there is a context edge somewhere in their combined state space. To refine our knowledge of the context-edge location a new learning state is introduced between the dissimilar states, increasing the resolution. Q-learning is applied again to find the meaning for the adjusted set of learning states. Increasing the resolution in the region of the context edge reduces the size of the area shared by dissimilar states, and therefore decreases the uncertainty regarding the context edge location.

This process is repeated to increase the accuracy. Ideally the learning-state boundary matches the context edge exactly, however it may be infeasible to reach this level of accuracy due to the demands of learning.

3.3 Adjusting the Context Definitions

Context edges are identified once an accurate set of learning states is discovered. Learning states are compared to their neighbors, again using our similarity metric (Section 3.1). Dissimilar neighbors have a context edge between them, while similar neighbors are in the same context. Context edge locations are compared to existing context definitions. If the bounds of existing definitions do not match the identified edges they are updated.

4 Experimental Results

To validate our initial implementation we performed experiments on a simple scenario where the environment is represented by a single-dimension state space (Fig. 6). The environmental situation at any point in time is some real value in the range 0-100. There are two actions: action A changes the environment towards 100, and action B changes it towards 0. This might equate to a real-world scenario such as a heating controller based on temperature readings.

The goal of the application is to keep the environment near a particular value. This value is the context edge between situations where Action A and Action B are appropriate. The task of the algorithm is to discover this edge so contexts can be defined.

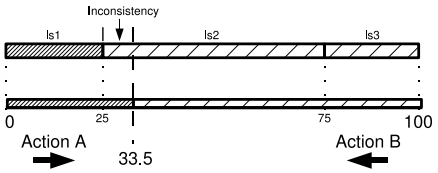

Figure 6. Initial scenario set up

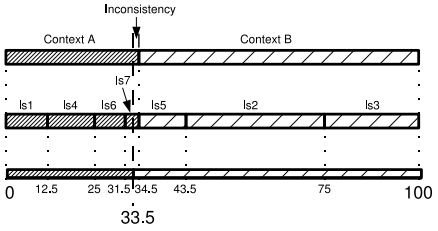

Figure 7. Context edge after four passes

To illustrate the algorithm we outline an experiment where the context edge was 33.5 units. We summarize the results of our experiments in Section 4.2.

4.1 Context-edge Identification

We applied the approach discussed in Section 3.2. Three default learning states were initially defined (Fig. 6), and the inconsistency was 8.5 units. Figure 7 shows the learning states after four passes of the algorithm. The region of state space near the context edge has been refined by new learning states, and the inconsistency is 1 unit.

A context edge was identified between *ls7* and *ls5*. Based on this edge *Context A* was defined as the region below 34.5 and *Context B* as the region above. This definition was within 1% of the actual context edge (33.5).

4.2 Inconsistency-Related Results

We performed 100 experiments on this scenario with randomly chosen context edges. Table 1 summarizes the inconsistency observed in learning states after 0, 2 and 4 iterations of the algorithm.

The average and standard deviation values demonstrate the algorithm's effectiveness. Both values fell as the number of iterations increased. By the fourth iteration a context definition based on the evolved learning states was on average within 1.03% $\pm$ 1.02 of the actual context definition.

# Iterations	Average	Standard Deviation
0	11.23	6.56
2	3.38	2.56
4	1.03	1.02

Table 1. Inconsistency-related statistics

5 Conclusion

We have presented an approach for making context definitions adaptable. It focuses on the application identifying context edges by interpreting sensor information for itself, and addresses some limitations of developer-defined contexts.

Having validated our approach using this simple scenario, we intend to apply it to a set of more complex context-aware scenarios. In the longer term, we hope to use this approach to allow the application adapt the set of sensors it uses to distinguish contexts.

References

[1] Parctab, http://sandbox.parc.com/parctab/.

[2] M. S. Atkin and P. R. Cohen. Using simulation and critical points to define states in continuous search spaces. In *Winter Simulation Conference (WSC)*, Orlando, 2000.

[3] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: from the laboratory to the marketplace. In *IEEE Personal Communications*, 1997.

[4] H. Chen, F. Perich, T. Finin, and A. Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems*, 2004.

[5] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. Technical report, Georgia Institute of Technology, 1999.

[6] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Special issue on context-aware computing in the HCI Journal*, 16, 2001.

[7] R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *IJCAI*, 1999.

[8] C. L. Nehaniv. Meaning for observers and agents. In *Internation symposium on Intelligent Control/Intelligent Systems and Semiotics*, Cambridge, 1999.

[9] D. Polani, T. Martinetz, and J. Kim. An information-theoretic approach for the quantification of relevance. In *European Conference on Artificial Life (ECAL)*, Prague, 2001.

[10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003.

[11] B. Schilit, N. Adams, and R. Want. Context-aware computer applications. In *1st IEEE Workshop on Mobile Computing Systems and Applications*, California, 1994.

[12] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[13] R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, 1998.