Context Inconsistency Management in Pervasive Systems

Samuel Esposito Alexander Jurjens

Abstract—Thanks to the pervasive computing paradigm more and more computer systems in utility buildings and industry are context-aware. They use a representation of the world they operate in to reduce the human-computer interaction necessary for their operation. Unfortunately reasoning based on contexts is not without flaws and context inconsistencies are the main reason for context-aware applications' incongruous behavior. Context consistency management is not adequately studied in existing literature and approaches for detecting and resolving context conflicts are not suited for pervasive computing [2].
In this paper we present two complementary approaches for improving the mitigation of context inconsistencies. First we present partial constraint checking for timely identifying context inconsistencies at runtime. An extra constraint layer is added to the traditional ontology based context model and conflicts can be detected by locally checking partial constraints in the ontology. This dramatically improves performance compared to iterative evaluation of an entire ontology [2]. Secondly we discuss the extention of the traditional ontology model with context lifecycles to more accurately represent the environment of context-aware applications. This information can then be used to estimate the relative reliability of contexts in a conflict set and discard the contexts with lowest reliabilty [3]. Apart from resolving context conflicts it is also possible to represent inconsistencies into the context model itself. In this paper we explore the possibilities of incorporating inconsitencies into ontologies using fuzzy OWL and discuss the consequences of this approach on standard reasoning methods [4].

Index Terms—Pervasive Computing, Ontology Model, Context Lifecycle, Inconsistency Resolution.

1 Introduction

In pervasive computing, contexts are used to make applications perform a certain action. Contexts can be seen as pieces of environmental information, which the applications depend on. Sometimes contexts are not consistent due to crossreads and misreads. The correctness of the contexts can be checked by evaluating the consistency constraints associated to them. There are multiple ways to check for consistency constraints. Constaint checking techniques are split up in two classes: Non-incremental checking and incremental checking. Incremental checking consists also of two classes: Entire Constraint Checking and Partial Constraint Checking. Once inconsistent contexts have been detected, an algorithm can be used to select the correct contexts. The algoritms use specific datastructures, like ontology models, in order to resolve the inconsistencies. However, certain conflict resolving algorithms are too slow for use in practice. This paper describes a way to use an ontology model in combination with the Partial Constraint Checking algorithm in order to detect context inconsistencies fast enough, such that it can be used in practice.

2 Related Work

Pervasive or ubiquitous computing is a fast-developing discipline that has been receiving increasing attention from both researchers and software developers [2]. In the past decade, many context-aware systems have been developed, ranging from smart room environments to warehouse and supply chain management systems. Lots of effort has been put into building middleware infrastructures that handle vast amounts of sensory data and extract the context information relevant for pervasive applications. Examples of such systems are CoBrA and CORTEX [3]. Various modeling approaches have been proposed for capturing context information, of which the ontology based context model appears to be most promising for most pervasive applications [3].

Context management for consistency however has not been adequately studied in the existing literature. None of the studies on context-awareness discusses a way for detecting context inconsistencies for reliable pervasive computing [2, 3]. Even though there has been related research going on in other disciplines as artificial intelligence and software engineering, it doesn't provide adequate support for context inconsistency detection in ubuquitous computing. In addition the strategies proposed in literature for resolving context conflicts are not suited for pervasive computing. Some are based on assumptions that may not apply to general pervasive environments. Other require human participation for conflict resolution, which is usually expensive and slow for pervasive computing [2]. Finally no research has been done on the potential of fuzzy ontologies for representing context inconsistencies in the environment instead of trying to resolve them [4].

In this article we aim at putting a milestone for context management by presenting an efficient inconsistency detection algorithm based on a constraint language extending the traditional ontology based context model [2]. In addition we put forward a conflict resolution algorithm which is based on a context reliability heuristic [3]. Finally the use of fuzzy ontologies representing context inconsistencies as a promising alternative to conflict resolution is explored.

3 Partial Constraint Checking

Constraint checking techniques have been extensively studied in software engineering. Existing constraint checking algorithms focus on checking software artifacts that do not change rapidly or frequently [2]. Context-aware applications require more efficient algorithms because they use a huge set of contexts, wich can change very rapidly and frequently (in the range of miliseconds). An inefficient software solution for this does not only require massive computing power, but interestingly also induces a higher inconsistency detection miss rate: because of the computing delay conflicting contexts slip through the context buffer before they are detected by the software [2]. One example of an inefficient approach is nonincremental checking: whenever there is a change in the set of software artifacts these artifacts are each checked against the entire set of consistency constraints to find out all detectable inconsistencies. An improvement to this would be incremental checking: only a subset of all constraints are checked, namely those that are affected by the specific change in the artifact set. But the real merit of Xu et *al.* was replacing the traditional entire constraint checking approach by partial contstraint checking based on a consistency computation tree [2]. Their idea is that constraints can be represented as trees with nodes for logical operations and leaves for specific properties of contexts or context sets. Whenever a context is added to or deleted from a context set in time, the branch corresponding to this context can respectively be added to or deleted from the tree (see Fig. 1). Because intermediate values are retained in the tree nodes after calculation they can be reused whenever the tree changes. More specifically, when a branch is added, only the values for the new branch itself and for the nodes from the branch top to the tree root need to be calculated. When a branch is deleted,

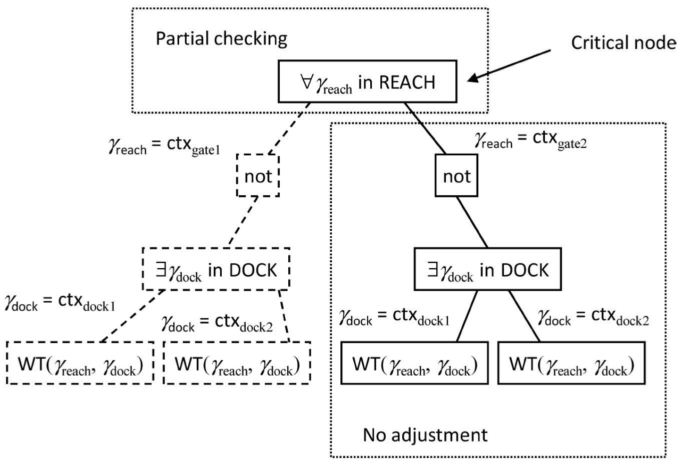

Fig. 1. Consistency Computation Tree: branches can be added or deleted to represent changes in a context set. Source: [2]

only the values for the nodes from the branch top to the tree root have to be recalculated.

With their partial constraint checking algorithm Xu et *al.* attained a time complexity between $O(n)$ for the worst case and $O(1)$ for the best case when a context was added to the set and $O(1)$ when a context was removed. Compared to traditional constraint checking with an overall complexity of $O(n)$, this is a dramatical improvement. In their experiments Xu et *al.* showed that their performance is 15 times better than the traditional approach and in a case study the inconsistency miss rate dropped from 52.2% in traditional checking to 0.1% with partial checking [2].

4 Context Inconsistency Resolution

Now that we have an efficient method for detecting inconsistencies the task of resolving the conflicts remains. As discussed above the strategies for conflict resolution in literature are not very suited for pervasive computing because their assumptions do not apply or they require human participation [2]. What really is needed is an algorithm that in the case of a conflict between two or more contexts decides which context has the highest reliability and retains that context. Bu et *al.* show us that this is possible through extending the ontology based context model with additional information about the context's status and temporal properties [3]. More specifically they propose an algorithm that for every conflict set retains the context with the highest *relative frequency*: frequency of context updates relative to update interval and context age (see Fig. 2). This is based on the assumption that con-

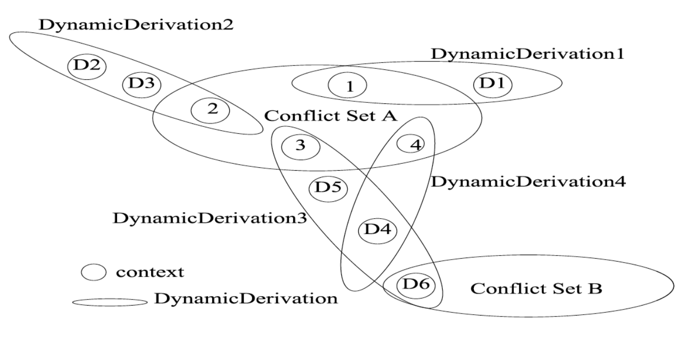

Fig. 2. Context inconsistency resolution in action. Source: [3]

texts that are most stably perceived by a system are most likely to be correct. This assumption is applicable to most of the environments in which pervasive systems run and is applied in many domains as a domain specific approach [2].

5 Fuzzy Ontologies

In pervasive computing traditionally first order logic is used to model the environment of an application. This results in a so called ontology based context model and every observation that does not exactly fit in this model results into a conflict that leads to misadjusted behavior in applications unless they are timely detected and resolved [2]. Because it is impossible to perfectly model the unpredictive environment in which context-aware applications run, it would be nice if the context model was more resilient to unexpected observations. As we illustrate in the case study below, this can be obtained by using fuzzy ontologies.

5.1 Case Study

Suppose we want to build a context-aware application that has certain notion of teacher's activities to assist them in the best possible way (for instance automatically opening slides when a lecture starts). In the traditional approach we could build an ontology specifying where lectures take place and that a lecture starts when the teacher appears at the lectern. If now an unpredicted observation is made (sensor cross read or teacher momentarily leaves the room), an inconsistency occurs that has to be resolved for the application to proceed with normal operation. In the fuzzy ontology approach these unpredictive events can simply be modeled so that the ontology is more robust and always yields the most apt interpretation of the environment without conflict resolution. A simplified example of such an ontology is depicted in Fig. 3. Using this fuzzy ontology the relative probability

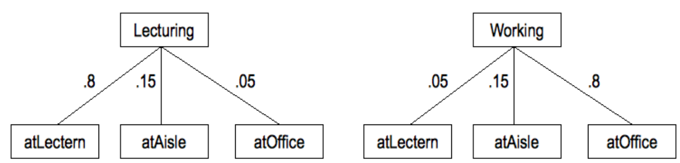

Fig. 3. Fuzzy ontology specifying teachers activities and their most probable location.

of each specified higher-level context can be calculated from the contexts in the observation window. For instance the observation sequence [L_1, L_2, L_3, A_1, A_2] gives a relative probability of 0.9998 for 'Lecturing' and 0.0002 for 'Working', event though the observations are clearly conflicting, since the teacher clearly cannot be at the lectern and in the aisle at the same time. The sequence of [L_1, A_1, A_2, O_1, O_2] gives a 0.941 probability for 'Working' and 0.059 for 'Lecturing', even though the teacher stricly spoken cannot be at the lectern and in the office at the same time (see appendix for exact calculations).

5.2 Results

Regardless of any inconsistencies in the observed contexts a fuzzy ontology always yields the most probable higher-level context without the need for conflict detection and resolution.

Interestingly a fuzzy ontology has some tree-like structure that allows for very efficient incremental partial probability calculation (PPC). The probability of a higher-level context is calculated by simply multiplying the probabilities of the observed contexts. When a new context is added to the observation window, the intermediate calculation result stored at a node (the root in our case study) can simply be multiplied by de probability of the new observation in order to obtain the updated probability for a higher-level context. When a context dissapears from the observation window, the intermediate result can simply be divided by the context's probability to obtain an updated probability for a higher-level context.

Note that it is possible to extend the simple fuzzy ontology from the use case by specifying start and transition probabilities for the higher-level contexts, just as hidden markov models[1] do, and other information such as the expected lecture time window (fuzzified of course [1]).

[1] http://en.wikipedia.org/wikiHidden_Markov_model

6 Discussion

In this article we started with presenting partial constraint checking: an efficient algorithm for timely detecting inconsistencies in huge sets of contexts which change very frequently. Partial constraint checking was 15 times as efficient as the traditional gready approach and reduced the conflict detection miss rate from 52.2% to 0.1%. This algorithm is clearly a dramatic improvement of the traditional approach and vital to any context aware application that has to handle high-frequently generated contexts.

After that we shed light on ways to resolve the detected conflicts and discussed a context inconsistency resolution algorithm that estimates the reliability of conflicting contexts and only retains the most reliable context from every conflict set. Using this algorithm allows for timely conflict resolution at runtime without the need for human participation. The algorithm is a valuable addition to the research on conflict resolution in pervasive computing, but when applying it one has to verify the aptness of the heuristic used for context reliability estimations in the domain at hand. In particular the assumption that contexts with the highest relative frequency are most reliable has to apply in the domain.

Finally the use of fuzzy ontologies for modeling the environment of context-aware applications seems promising. By incorporating possible inconsistencies in the model the need for context inconsistency management vanishes, vastly reducing the complexity of the used middleware. Fuzzy ontologies always yield the most probable higher-level contexts, regardless of the inconsistencies in the observed context set. In addition the calculation of the probabilities in a fuzzy ontology has great potential for optimization by using incremental partial probability calculation. When using fuzzy ontologies one however has to take into account the required training of the system for obtaining well defined probabilities in the context model.

7 Conclusion

In conclusion we can say that the partial constraint checking algorithm can dramatically improve the performance of context-aware applications where context consistency management is needed. With this performance improvement we get an impressive reduction of missed context inconsistencies for free. The context inconsistency resolution algorithm is a good solution in most domains for timely resolving context conflicts without human intervention. The fuzzy ontology approach seems a promising alternative to traditional context modeling but needs extensive further research in order to determine its real value in context-aware applications.

References

[1] J. J. Buckley. Fuzzify. *Fuzzy Sets and Systems*, 73(2):245 – 248, 1995.

[2] W. C. Chang Xu, S.C. Cheung and C. Ye. Partial constraint checking for context consistency in pervasive computing. *ACM Transactions on Software Engineering and Methodology*, 19, 2010.

[3] T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou, J. Wijsen, Y. Bu, S. Chen, J. Li, X. Tao, and J. Lu. *Context Consistency Management Using Ontology Based Model*, volume 4254, pages 741–755. Springer Berlin / Heidelberg, 2006.

[4] H. Kong, G. Xue, X. He, and S. Yao. A proposal to handle inconsistent ontology with fuzzy owl. *Computer Science and Information Engineering, World Congress on*, 1:599–603, 2009.

A Fuzzy Ontology Calculations

Observed Context Set 1: $[L_1, L_2, L_3, A_1, A_2]$

$$
\begin{aligned}
Lecturing &= .8 * .8 * .8 * .15 * .15 * c_1 && (1)\\
&= .01152 * c_1 && (2)\\
&= 0.9998 && (3)\\
Working &= .05 * .05 * .05 * .15 * .15 * c_1 && (4)\\
&= 0.000002813 * c_1 && (5)\\
&= 0.0002 && (6)\\
& && (7)
\end{aligned}
$$

Observed Context Set 2: $[L_1, A_1, A_2, O_1, O_2]$

$$
\begin{aligned}
Lecturing &= .8 * .15 * .15 * .05 * .05 * c_2 && (8)\\
&= 0.000045 * c_2 && (9)\\
&= 0.059 && (10)\\
Working &= .05 * .15 * .15 * .8 * .8 * c_2 && (11)\\
&= 0.00072 * c_2 && (12)\\
&= 0.941 && (13)\\
& && (14)
\end{aligned}
$$