Context Inconsistency Management Using Partial Constraint Checking

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 this operation. Unfortunately reasoning based on contexts is not without flaws and context inconsistencies are the main reason for context-aware applications' incongruous behavior. Detecting context inconsistencies using traditional context modeling is very computer intensive, making timely conflict resolution unfeasible in most settings.
In this paper we present two complementary approaches for improving the mitigation of context inconsistencies. First we extend the traditional ontology based context modeling approach with context lifecycles to more accurately represent the world surrounding the application. Many of the conflicts in context reasoning can simply be resolved using information about the lifecycle state of a set of specific contexts [2]. Secondly we propose partial constraint checking for more efficiently identifying and timely resolving context inconsistencies at runtime. By adding an extra constraint layer to the traditional ontology model, conflicts can be resolved by checking constraints or partial constraints only locally in the ontology. This dramatically improves performance compared to iterative evaluation of an entire ontology [1]. Apart from resolving conflicting contexts it is also possible to represent them into the ontology model. In our paper we explore the possibilities of incorporating inconsitencies into ontologies using fuzzy OWL and discuss the consequences of this approach on standard reasoning methods [3].

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

1 Introduction

- WHAT
Conflict detection
Conflict resolution
Inconsistent ontologies

- WHY
Performance
Automate resolution
Better model context

- HOW
PCC
CIR
HMM - Fuzzy set theory

2 Related Work

Pervasive or ubiquitous computing is a fast-developing discipline that has been receiving increasing attention from both researchers and software developers [1]. 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 [2]. 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 [2].

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 [1, 2]. 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 [1]. 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 [3].

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 [1]. In addition we put forward a conflict resolution algorithm which is based on a context reliability heuristic [2]. 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 [1]. 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 [1]. 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 [1]. 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, only the values for the nodes from the branch top to the tree root have to be recalculated.

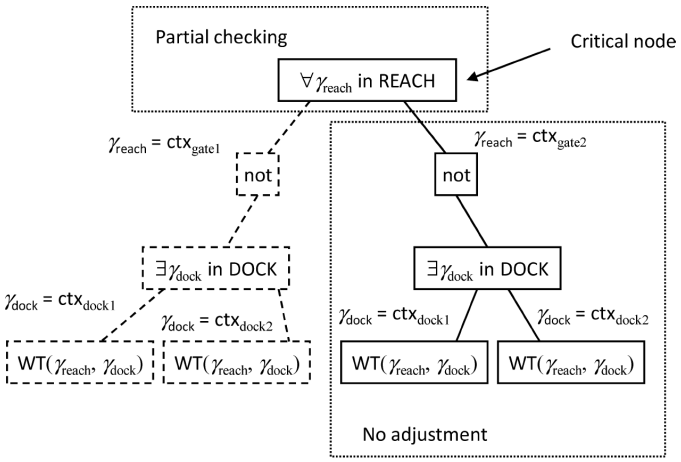

Fig. 1. Consistency Computation Tree: branches can be added or deleted to represent changes in a context set. Source: [1]

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 [1].

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 [1]. 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 [2]. 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-

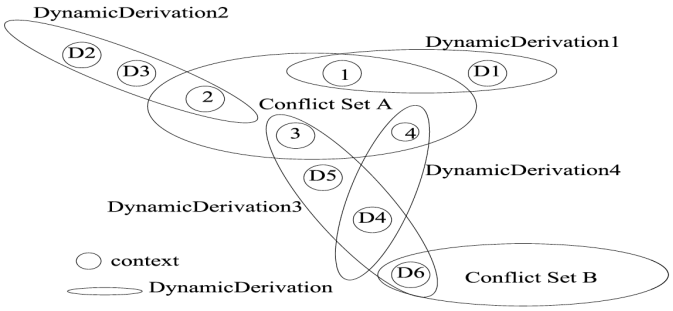

Fig. 2. Context inconsistency resolution in action. Source: [2]

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 [1].

5 Fuzzy Ontologies

better modeling training required maybe slower extensive research necessary

HMM Forward-backward algorithm

6 Discussion

7 Conclusion

References

[1] 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.

[2] 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.

[3] 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.