# 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 [3].
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 [3]. Secondly we discuss the extension of the traditional ontology model with context life cycles 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 reliability [2]. Apart from resolving context conflicts it is also possible to represent inconsistencies into the context model itself [5]. In this paper we present a case study in which we explore the possibilities of incorporating inconsistencies into context models using fuzzy ontologies.

**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 cross reads 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. Constraint checking techniques are split up in two classes: Non-incremental checking and incremental checking. Incremental checking also consists 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 algorithms use specific data structures, 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. Another approach to handling context inconsistencies is representing them into the context model, as opposed to resolving them. This can be done efficiently by using fuzzy ontologies in a tree like structure, as we show in a use case.

## 2 Related Work

Pervasive or ubiquitous computing is a fast-developing discipline that has been receiving increasing attention from both researchers and software developers [3]. In the past decade, many context-aware systems have been developed, ranging from smart room environments to warehouse and supply chain management systems. A lot 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 [4] and CORTEX [1]. 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 [3, 2]. Even though other disciplines as artificial intelligence and software engineering conducted relate research, it does not provide adequate support for context inconsistency detection in ubiquitous 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 [3]. Finally no research has been done on the potential of fuzzy ontologies to represent context inconsistencies in the environment instead of trying to resolve them [5].

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 [3]. 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 [3]. Context-aware applications require more efficient algorithms because they use a huge set of contexts, which can change very rapidly and frequently (in the range of milliseconds). 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 [3]. One example of an inefficient approach is non-incremental 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 constraint checking based on a consistency computation tree [3]. 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 of the new branch itself and of the nodes from the branch top to the tree root need to be

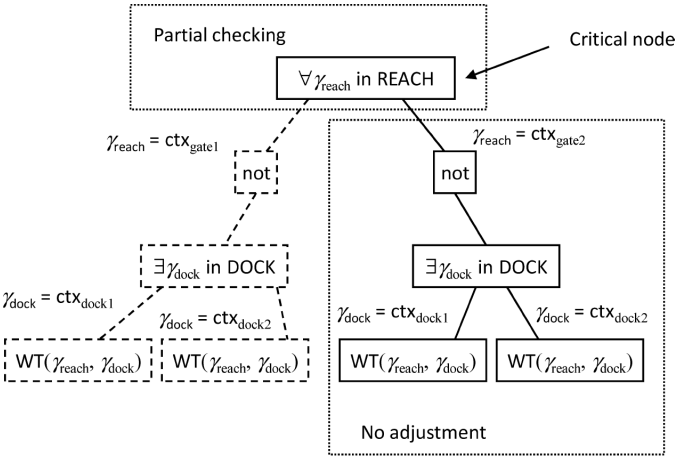


Fig. 1. Consistency Computation Tree: branches can be added or deleted to represent changes in a context set. Source: [3]

calculated. When a branch is deleted, 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 $O(n)$ in the worst case and $O(1)$ in the best case when a context is added to the set and $O(1)$ when a context is 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 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 [3].

## 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 well suited for pervasive computing because their assumptions do not apply or they require human participation [3]. What is really 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-

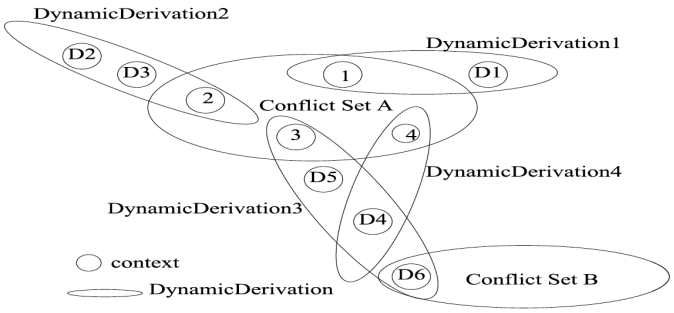


Fig. 2. Context inconsistency resolution in action. Source: [2]

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 [3].

## 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. Every observation that does not exactly fit in this model results into a conflict. This conflict can lead to misadjusted behavior in applications, unless it is timely detected and resolved [3]. Because it is impossible to perfectly model the unpredictable environment in which context-aware applications run, it would be nice if the context model was more resilient to unexpected observations. This resilience can be obtained by using a fuzzy ontology: a stochastic model representing lower level contexts (for instance sensory data) and their probabilities in the presence of higher level contexts (for instance user activities). The most simple approach to this would be using a hidden Markov Model[1]: a statistical model of the environment with unobserved states (the higher level contexts) in which future states only depend on the present state. The use of such a model is illustrated in the case study below.

### 5.1 Case Study

Suppose we want to build a context-aware application that has a 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 its normal operation. In the fuzzy ontology approach these unpredictable events can be modeled so that the ontology is more robust and always yields the most apt interpretation of the environment without the need for conflict resolution. A simplified hidden Markov model for this ontology is depicted in Fig. 3. As the numbers in this model indicate, the

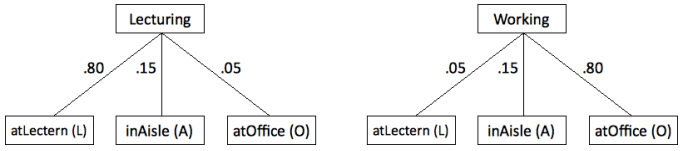


Fig. 3. Fuzzy ontology specifying a teacher's activities and the probabilities of different locations in the presence of these higher-level context.

most probable location of a teacher while lecturing is at the lectern (L). In the aisle (A) is less probable and at the office (O) is very unlikely. For the activity of performing other work (denoted as Working in the model), the probabilities are the other way around. This fuzzy ontology model can now be used to calculate the relative probability of each higher-level context (c.q. Lecturing and Working) based a set of low level contexts observed within a certain time window. A possible set of observations could be $[L_1, L_2, L_3, A_1, A_2]$: the teacher was observed three times at the lectern and twice in the aisle in this time window. The probability of the Lecturing activity can now be calculated by multiplying the probabilities of the observations in the presence of this higher level context[2]: $pLecturing = .80*.80*.80*.15*.15*c_1 = .01152*c_1 = 0.9998$. The probability for Working can be calculated in a similar way: $pWorking = .05*.05*.05*.15*.15*c_1 = 0.000002813*c_1 = 0.0002$. The probabilities indicate that the teacher is still lecturing, even though the teacher being observed in the aisle conflicts with being observed at the lectern. Another example of observations could be $[L_1, A_1, A_2, O_1, O_2]$: the teacher is observed at the lectern once, twice in the aisle and twice at the office. The probabilities for Lecturing and Working are calculated as follows: $pLecturing = .80*.15*.15*.05*.05*c_2 = 0.000045*c_2 = 0.059$ and $pWorking = .05*.15*.15*.80*.80*c_2 = 0.00072*c_2 = 0.941$.

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

[2] We multiply the results with a constant $c$ to obtain a valid probability value in the end

The probabilities now indicate that the teacher is doing work other than lecturing, even though the observation of the teacher at the lectern is definitely conflicting with the observations at the office.

### 5.2 Results

The case study shows that a fuzzy ontology allows us to always define the most probable higher-level context given the observations, regardless of any inconsistencies in the observed context set and without the need for conflict detection and resolution.

Another observation we can do is that the hidden Markov model for a fuzzy ontology has a tree-like structure. This structure allows for very efficient incremental Partial Probability Calculation (PPC). The PCC principle works as follows: the probability of a high-level context is calculated by multiplying the probabilities of its children contexts. When a new context is added to or removed from the observation set and thus from the tree, only the probabilities of the observation's ancestors in the tree up to the root have to be recalculated. In case of adding a context, the probability of an ancestor in the tree can be recalculated by simply multiplying its previous probability with the probability of the new or updated child. In case of removing a context, the probability is divided by the probability of the new or updated child. Because the recalculation only takes place in part of the tree and allows us to preserver previous calculations for later use, it is partial. And because this recalculation only happens in trees that contain the added or removed context as a leaf, the calculation is incremental. We illustrate the PPC process using an extension of the hidden Markov model from the case study (See Fig. 4). In this figure the

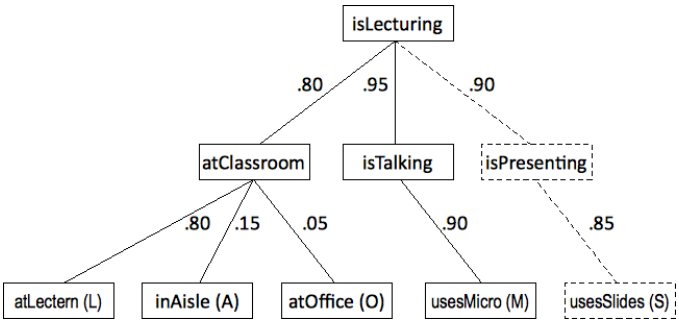


Fig. 4. Fuzzy ontology specifying the Lecturing activity and the probabilities of different contexts in the presence of this higher-level context.

usesSlides context is added to the observation set at some point. This change in the context set requires us to recalculate the probabilities of the higher-level contexts. For this we use PPC: we recalculate only the values of the ancestors if the usesSlides context (c.q. isPresenting and isLecturing). The probability of isPresenting is $.85 * .90$, because the usesSlides context is its only child. The probability of isLecturing has to be recalculated. This can be done by simply multiplying its previous probability with the probability of the new child isPresenting: $isLecturing = 0.0004104 * .90 * .85 = 0.000313956$[3].

The time complexity of the PPC principle is related to the height of a hidden Markov tree, because recalculations only take place on the path from a leaf to the root. Since this height remains constant at runtime, the time complexity of adding or removing a context is $O(1)$.

It is possible to extend the hidden Markov model used in the case study by specifying start and transition probabilities for the higher-level contexts. With this extension the probability of a higher-level context does not solely depend on the contexts in the observation set, but also on the previous most probable higher-level context.

## 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 frequently. Partial constraint checking was 15 times more efficient than the traditional greedy 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 frequently generated contexts.

After that we shed light on ways to resolve the detected conflicts and discussed a context inconsistency resolution algorithm which estimates the reliability of conflicting contexts and only retains the most reliable context from every conflict set. 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.

[3] Note that this value has to be multiplied with a constant in order to get the real probability value

## References


[1] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pages 361 – 365, 2004.

[2] 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] 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.

[4] H. Chen, T. Finin, and A. Joshi. Semantic web in the context broker architecture. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pages 277 – 286, 2004.

[5] 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.