Service Dependency Analysis in Ubiquitous Environment

Raheel Ali Baloch
RS2M
Institut Telecom SudParis
Evry, France

Noel Crespi
RS2M
Institut Telecom SudParis
Evry, France

***Abstract*— To provide a user with most personalized adaptive services only using the accessible computing resources in a ubiquitous environment, context-aware services need to assimilate both the accessed and derived context information in the environment. Context dependency may get introduced in a system due to numerous reasons, but as the number of context dependencies for a service increases, the more complex the process becomes to develop and maintain such an application. Due to highly dynamic nature of mobile agents in an ad-hoc network, far too many context dependencies can severely affect the performance of a context-aware service. Since almost-continuous operability is desirable from pervasive services, it is crucial to determine the existing context dependencies among the services to pinpoint weak points and highlight performance bottle-necks. In this paper, we present a time efficient approach to determine dependency relations among various services in ubiquitous environment to help better analyze the pervasive services.**

Keywords - ubiquitous services; dependency; context-aware; pervasive computing

I. INTRODUCTION

Context aware services are the foundation for today's world of ubiquitous computing. Services in a ubiquitous environment have to interact with a variety of mobile devices, communication networks, and wireless sensors along with the variability of their respective execution environment. Therefore, pervasive services should be conscious of their running context in order to react accordingly to multiple execution scenarios. Personalization of services depends upon context aware applications to achieve the vision of ubiquitous computing [1]. In a ubiquitous computing environment, various heterogeneous applications and devices are involved, generating and consuming context information rapidly. Data and context information from sensor nodes are available in ad-hoc networks to be utilized by context aware services and applications. Mobile devices are also leaving and joining ad-hoc networks at a dynamic rate, making context information highly unpredictable in nature. Such an ever changing nature of context information makes context aware services more complex to develop, maintain and comprehend.

Context is usually referred to the attributes in a continually dynamic ubiquitous computing environment, in which various aspects like the user's location, social situation and interaction with other resources are changing constantly. Because of its more general approach, the definition provided by Dey and Abowd is most widely quoted in the research community. The authors define context in a following way: "Context is any information that can be used to characterise the situation of an entity." An entity can be defined as "a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves." [2]

Service dependency may get inducted in a system due to various reasons, but that issue aside, the more context dependencies a context service has, the more complex the process becomes to develop and maintain such a service. Due to highly dynamic nature of mobile devices in an ad-hoc network, too many context dependencies can severely affect the performance of a context-aware service. Reducing dependencies or providing alternative context sources is crucial for performance gains in context-aware environment.

Dependency analysis can help to choose sensible sets of loosely-coupled dependent services, with only absolutely required dependencies, and can facilitate in determining exactly which of the context services rely on a context source along with the thorough nature of such dependencies. It must be determined which context services are affected directly and indirectly, if there is any change in source. If source is not currently available in the network, is there any other alternative similar source which can be utilized to provide the dependent services with relevant context information. The analysis can also help to determine the dependency of services on other ones, and how it affects the subsequent context service if its derivation is delayed or halted all together. The complete service dependency analysis of a complex ubiquitous system determines the dependency hierarchy among the participating context-aware services and applications. It facilitates the breaking down of large complex systems into smaller, loosely-coupled components whose complexity can be analyzed and managed more easily.

The rest of the paper is organized as follows: Section II briefly explain context dependency, Section III gives an overview of related work. Section IV presents an example

scenario. Section V describes in detail the constraint satisfaction problems, followed by our proposed approach determine dependency in polynomial time in Section VI and VII, and VIII concludes the paper with proposals for future work.

II. CONTEXT DEPENDENCY

Pervasive context aware services need to deal with heterogeneous context information and services. It is highly possible that such context information is related and dependent on other entities. According to [3], "A dependency is a special type of relationship, common amongst context information, which exists not between entities and attributes, as in the case of associations, but between associations themselves." The reliance between the entity and its attributes is represented with directional relationships, and such a relationship shows the dependence of one association on another. In [4], the importance of capturing dependencies in context aware applications is highlighted. Lack of information about the dependency relationships can lead to inappropriate decisions that can result in unwanted behavior or even unstable pervasive system. Moreover, information of context dependencies is significant from a context management perspective, as it can assist in the detection of context information that has become obsolete.

Due to highly dynamic nature of agents, too much context dependency can severely affect the performance of a ubiquitous system. With the increasing number of participants in an ad-hoc network, context dependency induced problems need to be handled efficiently. If the problem is not properly addressed, interdependence on low quality context will increase, leading to undesired or even poor decision making by context-aware systems. A solution based on our formal model can foster the development of context aware applications in better comprehensible mode.

As shown in the Fig 1., in the first dependency called Direct Dependency, a context of type A is being utilized by various applications to generate context of types B and C. Most simple scenario to depict such context dependency is the availability of the user on his mobile phone when he is in a meeting. In such a situation, user can only be contacted on his mobile phone through text message, avoiding voice calls altogether. The decision to drop voice calls is based on two contexts; user location and calendar entry of the meeting. The mobile application is dependent upon these two contexts to make correct decision. If any one of the context is not acquired entirely or even acquired late, the usefulness of such mobile application is not valid anymore as the decision is dependent upon these two contexts.

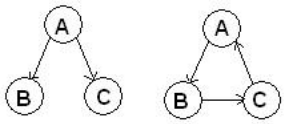

Fig. 1 Types of Context Dependency: Direct & Cyclic

The second type called Cyclic Dependency; there is mutual interdependence of context between the participants. A simple example of cyclic dependency can be bandwidth adjustment for a certain multimedia data transfer over a wireless channel. The mobile application monitors the available bandwidth and then adjusts the quality of the video transfer accordingly to achieve smooth streaming. The base station detects the video streaming requirements and adjusts the bandwidth, which in turn set in motion the whole process again. This cyclic context dependency can be serious threat to effective execution of a context-aware system if it is unintentional. Any service that contacts another for context is acting as a client or more specifically, a context consumer, and any service that accepts contact from the client is acting as a server or context producers in context aware application terminology. The developers must be careful to avoid cyclic context dependency among participating context consumers and producers. The chain of context requests introduced due to cyclic context dependency can continue indefinitely until all the involved applications exhaust their resources. The probability of circularity is particularly high when context services are designed independently, because no single developer can predict all possible interaction among services.

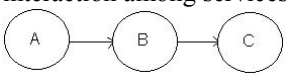

Fig. 2 Transitive Dependency

In addition to two direct dependencies, Fig. 2 shows a transitive dependency between A and C. A transitive dependency exists when two entities are connected through one or more than one intermediary nodes [5]. Most transitive dependencies are generally not as significant as direct dependencies. If a system is designed properly, it is likely that a majority of changes will not have significant effect through transitive dependencies. In fact, UML (Unified Modelling Language) does not acknowledge transitive dependencies as actual dependencies in software systems [6]; if the dependency between two items is not direct, the items are presumed to be mutually independent [7]. But we still have to analyze the effects of transitive dependencies in context-aware systems more thoroughly.

III. RELATED WORK

Most methodologies to service monitoring and analysis are concerned with the assessment of contracts between service provider and service consumer [8], [9]. Such approaches don't work if the services are composition of multiples services because composite services have vertical and horizontal dependencies that need to be taken under consideration.

The methodology presented in [10] tries to monitor vertical dependencies in the system. The methodology is limited in its approach for the complete monitoring of dependencies among composed services because dependencies between two single services are not fully captured. The same issue is with the COSMA approach [11] that also monitors vertical dependencies among the services but lack an approach to handle horizontal dependencies.

Most of the research work carried out in context-awareness involves either provision of frameworks for supporting abstract information in context aware systems, or modeling of context information for the relevant context queries. We review the context modeling efforts of these research directions in ubiquitous systems.

The approach called CONON [12] is based on ontology capabilities of knowledge sharing and its reuse. It concentrates on context classification, representation and its reasoning with the help of inference engines. The strength of such models is based on structure level [13].

A formal context modeling approach is proposed in [14]. It is based on object modeling paradigm. The Entity-Relationship model is used as a basis for the language to construct a conceptual model of context. The resultant context information is then stored in relation database.

The model presented in [15] is concerned with capturing meta information from the context, and then describes features like quality, correctness, source, format, along with the transformation process it underwent for its current form from raw sensed data.

The context service presented in [16] called Owl maintains and provides on request context information to its clients. Various context attributes like context history, scalability, quality, access rights, etc. are handled in Owl. These models lack formal basis to capture relevant context information.

Set theory [17] is used to define a context tuple of a certain size n, where n is the number of different context sources present in the device. The variable present in the tuple represent a value of certain context corresponding to a context source. The tuple contains an extra variable which represents the time when the tuple was created. Context information is schematically described by the set theory approach, but lacks any information about dependency relations.

Directed graph approach is proposed [3] in which an entity is described by context information modeled in the form of a directed graph. The nodes are used to represent entity and attribute types, while the associations among them are arcs connecting the nodes. The model is comparatively comprehensive as it can represent quality of context, and context dependency relations but lack accuracy in dependency representation.

A distributed algorithm called Genetic Relation of Contexts, (GRC), analyze interdependence of context data in a decentralized environment [18]. GRC tries to solve the problem of cyclic context dependencies with an approach of splitting and multiplication of context. Initial results are interesting, but the formal model for context dependencies is still lacking.

IV. EXAMPLE SCENARIO:METTING ROOM

We present here a rather simple scenario that is used later in the paper to highlight our approach. The scenario is concerned with meeting room situation. During the duration of the meeting, it is considered highly inappropriate to have someone's mobile phone ringing. In our example scenario, the user has a mobile phone with him at a meeting in his office. The context-aware application determines that the user is in the meeting after considering location (i.e. the meeting room), time, and calendar entry about the meeting. If all these context readings indicate that the user is busy in a meeting at the moment, the context-aware application turns on the mute mode so that no one gets disturbed by any incoming call. In addition to mute mode, the application also turns on the mobile vibrator to alert the user of any incoming calls. The messages received through SMS are handled in the same way as the incoming calls. The Fig. 3 explains the scenario with the help of a decision tree. When the user leaves the meeting, the change in the context is sensed by the context-aware application and the mute mode is turned off along with the mobile vibrator.

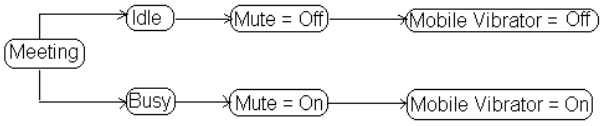

Fig. 3 Meeting Scenario

V. CONSTRAINT SATISFACTION PROBLEMS

In this section, we will present the theoretical and mathematical introduction of our work. An introduction to constraint satisfaction problem (CSP) is presented, and then their formalization is specifically explained with the help of the above mentioned example scenario to demonstrate our algorithm. Later, the complexity of our algorithm designed to determine dependencies is derived.

A. Formal Introduction of CSPApproach

To capture design decisions and the assumptions related to those decisions, we have used CSP approach to serve as a model. The focal components that can be considered part of CSPs are: a constraint network with a finite domain, a dominance relation.

1) Constraint Network (CN)

A CN is a set of variables and constraints that are inter related and define the valid values for the variables that satisfy the relevant constraints. A variable is used to represent each distinct piece of information in a system. The value of each of the variable is assigned from a given, finite domain. At any given instance of time, a subset of the domain of a variable comprises its set of possible, valid values. The system's view of the available choices for that variable is represented by such possible values. These possible values are always consistent with all the current constraints. Apart from possible values, each variable may have an assigned value at any given time which can be assigned by the user in the first place, or selected by the system from the set of possible values. The assigned value remains until it becomes inconsistent with the relevant constraints. Implicitly, a constraint defines the valid combinations of values for a given set of variables. A simple constraint defines valid permutations of values for a set of n variables. A 4-tuple is used to represent CN, (V, U, M, C). $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n variables present in the system. U

represents the universal set which contains all the possible valid values for the variables in the set *V*. The mapping from variables from the set *V* to valid values from the set *U* is represented by *M*. Constraints upon the variables are represented by a finite set *C*.

Variables	Domain Values
Meeting	Busy, Idle
Mute	On, Off
Mobile Vibrator	On, Off

Constraints
Meeting = Busy => Mute = On
Meeting = Idle => Mute = Off
Mute = On => Mobile Vibrator = On
Mute = Off => Mobile Vibrator = Off

TABLE I. VARIABLES AND CONSTRAINTS

A CN that is presented in Table I shows the shows the variables, valid values and constraints derived from the scenario. To specify the value of a variable when represented in a constraint, the term binding is used.

2) Dominance Relation(DR)

A DR relates dependency among variables to achieve design rules' annotation. From the presented scenario, we can extract two pairs of DR, e.g. (mute, meeting) and (vibrator, meeting) in which the two variables, mute and vibrator, are dependent on meeting, and as such, have no reverse effect on the meeting variable. The dependency among the variables can be formally defined as a relation $(u,v) \subseteq V \times V$, if *v* is dependent on *u*. Any change to *u* which results in CN's invalid state by negating any of the constraints, must force *v* to adopt a new value which is minimally different from its current one to restore the CN to a consistent, valid state by conforming to all the constraints. If *S* is the set of all solutions in a constraint network then a solution to a constraint network is a mapping of all variables to valid domain values $\forall v \in V \wedge s(v) \in \mathrm{M}$ such that all the constraints are met accordingly.

B. System States

A finite number of states are derived from the CN and the DR. The states provide solutions to the CN, and the set aggregating such solutions is called *S*. If there is a solution s belonging to *S* in the design decision, with *v* the variable who can take different values, and *u* is one of the values in the solution that *v* can take, the transition function is represented as $\delta(s, v, u)$. Assigning value *u* to *v* must be handled in such a way that there is no violation in the initial valid state. If there is a violation when assigning a value *u* to *v*, then the values of the other variables in the initial state must also be altered to keep the system in valid state. Updating the value of the secondary variables must follow the DR. If a variable *v'* must be changed to achieve valid state after *v* has got a new value, it must be ensured that *v* dominates *v'*. If (*v', v*), then there is no need to update *v'* as it's the variable *v* that is the dependent on *v'*, and any change to *v* will be considered void.

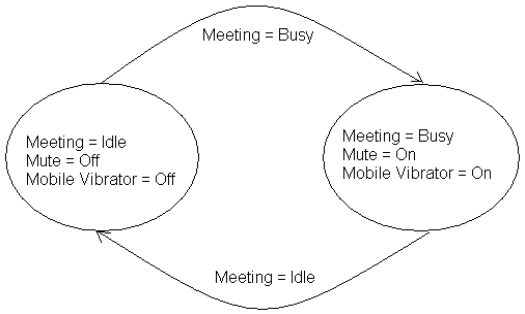

Fig. 4 System States and Transitions

A system's states of meeting room scenario are shown in Fig. 4. As the CN of this simple scenario has only two possible valid solutions that satisfy the constraints, there are only two valid states of the system. The system can be in either of the two states that we refer as busy and idle. In busy state, the variables 'meeting', 'mute', and 'mobile vibrator' have values of 'busy', 'on', and 'on', respectively. Alternatively, the state can be in idle state in which the variables 'meeting', 'mute', and 'mobile vibrator' have values of 'idle', 'off', and 'off', respectively. The valid transition between the two states is the resultant of minimal change. The variable 'meeting' is the dominant variable in the both of the DRs that we derived earlier. So, the transition from one state to another requires change in the variable 'meeting' only. Change in the value of variable 'meeting' requires the remaining variables to also adopt new values to satisfy the constraints, and hence, both variables, individually, are dependent on 'meeting'.

C. Determining Dependencies and NP-Completeness

According to [19], deriving dependency from a CN is NP-complete as CSP is NP-Complete. The constraint in CSP is true if the values assigned to all the system's variables satisfy the constraint. Therefore, finding a solution to CSP is equivalent to searching for an assignment of values to all variables such that all system's constraints are satisfied. In our approach, we convert the CSP into a dependency decision problem by using a given CN, and two given variables, *a* and *b*, to determine whether there exists a dependence (*a*,*b*) between the variables.

To proceed with our approach, a CN with two more variables $V' = V \cup \{a, b\}$, and two additional domain values for the added variables, $U' = U \cup \{true, false\}$ is considered. A corresponding constraint is also added to the CN, $a=true \Rightarrow b=true$. The variables in CN can have any values with the only restriction that the values should belong to the set of valid domain values. But here we have added an extra constrain on *a* and *b* to have only the Boolean values of either *true* or *false*, and the constraints also ensures that *a* and *b* have the same value in all solutions, thus any change to *a* has a direct impact on the value of *b* to restore solution's validity. Since our approach is derived from CSP and the constraints of CN are subsets of the CSP constraint set, if there is no solution to the CSP case that can be obtained in polynomial time, then there is also no solution to the CN case, and therefore, it means determining dependency is non-polynomial, too. It is explicit that if there is a solution to the CSP case in the form of *s*, then there can be only two solutions to the constraint network case.

One solution is when the value of both the variables, *a* and *b*, is *false*, the particular solution called s_0, and the other solution when the variables have the value of *true* with the solution labeled s_1. As there is only minimum difference between s_0 and s_1, any change to *a* causes *b* to update its domain value, it shows dependency of *b* on *a*, and if there is a solution to CSP case then *a* and *b* have dependence relation. But since CSP is NP-Complete [19], computing dependence relation is NP-Complete also.

VI. DEPENDENCY ANALYSIS AND CSP

The CN can be successfully utilized to analyze dependencies on the conjecture that one entity is dependent on another one, and a change in an entity can have an effect on the dependent entity, which causes to alter the current state of the dependent entity. In such scenarios, usually, the prime interest is to determine the influence. This observation helps to abstract that a domain value of a variable in a constraint network can have only two possibilities: current state and changed state. The reduction to two states significantly facilitate in studying the modularity of pervasive systems, and the analysis of context dependencies thoroughly.

Essentially, in our approach, we consider a CN where each domain can provide two values to satisfy the variables, and every constraint involves two variables. Later, we will show that determining dependencies is not NP-Complete by presenting a polynomial time algorithm which is based on CN with the CN having properties just mentioned above. The two domain values approach is the reason to the polynomial time solution to the computation of dependency. Any CSP can be altered to another form with only having binary constraints [20]. Since the domain values in the CN can now have two states in our approach, it can be appropriately stated that one state is true and the other one is false, which makes the CN an instance of 2-SAT problems. Indeed, our approach is to treat this CN as a 2-SAT instance as 2-SAT can be solved in polynomial time [21]. The easiest way to compute dependency relation is to search for all satisfying solutions and then get a construct and then obtain the remaining solutions in polynomial time [22]. This seems rather tempting approach but soon, the state explosion problem surface up, and since in the worst case scenario, there are exponentially large number of solutions that makes this approach non workable for large domain space [23]. The subsequent section shows that our algorithm is independent of the number of solutions, which help us to avoid the state explosion problem.

VII. DEPENDENCY ANALYSIS ALGORITHM

We continue with our earlier example scenario to explain our algorithm to determine context dependencies. The approach we adopt is based on a graph structure generally employ to solve conventional 2-CNF problems. This graph structure is called Implication Graph, and we utilize it to construct another graph structure, called Dependency Graph. The Dependency Graph is built to provide dependent variables' pairs.

A. Implication Graph

In the first part of our approach, we develop an implication graph [21] which has two vertices for each of the variable in the constraint network as the domain values in our case can only have Boolean values. We build this graph so that we can model the constraints present in the constraint network. An implication graph is a non-symmetric directed graph G (V, E) composed of vertex set V and directed edge set E. Each vertex in V represents the truth states (i.e. either true or false) of a Boolean literal, and each directed edge from vertex u to vertex v represents the implication "If the literal u is true then the literal v is also true"[1]. In Fig. 5, the implication graph of the example scenario is shown where each constraint has two vertices for corresponding Boolean values.

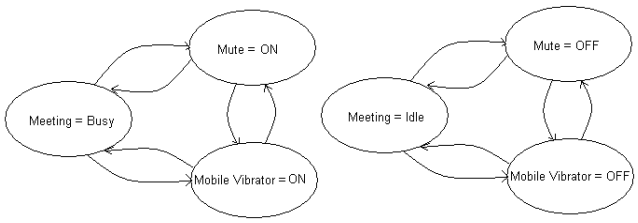

Fig. 5 Implication Graph

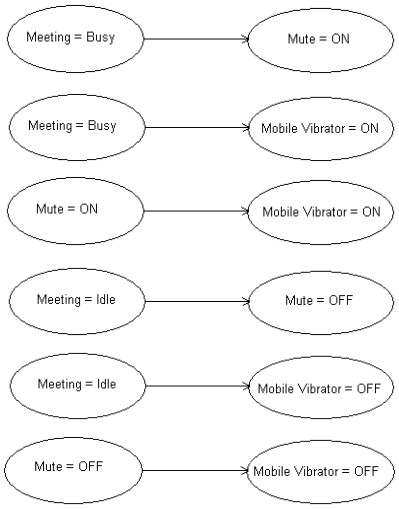

Fig. 6 Dependency Graph

B. Dependency Graph

In the second part of our approach, we develop an dependency graph using implication graph. In the implication graph, we have two edges, in the first instance, between the two vertices, and two more edges for the other two corresponding vertices. We initiate the construction of the dependency graph by following the rule in which we keep the edges between the vertices if there is any dependency exist, otherwise if a vertices can not influence the other vertices then the edges are removed between such two vertices in the implication graph. Formally, for two variables *a* and *b*, if (*a*, *b*) and (*b*, *a*) are not valid, we remove the edges (*a*, *b*), (*a'*, *b*), (*a*, *b'*) and (*a'*, *b'*) from the implication graph .We repeat the rule

[1] http://en.wikipedia.org/wiki/Implication_graph

for each of the vertices, and at the end we have our dependency graph, Fig. 6.

C. Complexity Analysis

For determining the complexity of our approach, we need to consider the running time complexity of the two main components. Let *n* be the number of variables involved in the CSP, and m be the number of constraints that need to be satisfied in the CSP.

1) Construction of Implication Graph – $O(n+m)$
2) Construction of Dependency Graph
   a) Selection of Dominant Edges – $O(m)$
   b) Removal of Invalid Edges – $O(m)$

So, the overall time complexity to determine dependencies is $O(n+m)$. The graphs constructed in our approach also use space corresponding to given variables and constraints, i.e. 2n nodes and 2m relation edges, resulting in both polynomial time and space complexity.

VIII. CONCLUSION AND FUTURE WORK

The paper provides a dependency analysis approach based on constraint satisfaction problem for the context-aware ubiquitous services. Context-aware services are usually developed using context models and concepts that are informal, lack any clarity, mostly aimed at a particular application domain. The dependency analysis may reveal a lot about dependencies among ubiquitous services, and for each dependency some possible dependency reduction strategy may exist that need to be studied in the future. But the question is how to determine and manage all these dependencies, as new dependencies may get introduced due to reduction of a previously existing dependency. As a consequence, some dependency reduction approaches will introduce new dependencies, and some dependency reduction strategies will introduce new possibilities to resolve certain context dependencies. The issue of constraint solving to determine service dependency has got its complexity reduced from NP-complete to polynomial time complexity through our approach.

The future work that we envision in this dimension involves enhancing the algorithm further by introducing dependency confirmation logic. We further intend to undertake simulation studies to validate and support our work. The essential task will be to evaluate the pervasive systems with high coupling and direct dependencies in comparison to ones with low coupling and indirect dependencies.

REFERENCES

[1] M. Weiser., “The computer for the 21st century”, Scientific American, 265 (3):66–75, 1991.

[2] Dey, A. & Abowd, G., “Towards a Better Understanding of Context and Context-Awareness”. In CHI ’00: Workshop on the What, Who, Where, When, and How of Context-Awareness, Georgia Institute of Technology, Atlanta, GA, USA, 1999.

[3] K. Henricksen, J. Indulska,A. Rakotonirainy. “Modeling Context Information in Pervasive Computing Systems”. Proceedings Pervasive 2002 -Zurich August 2002.

[4] C. Efstratiou., K. Cheverst, N. Davies, A. Friday, “An architecture for the effective support of adaptive context aware applications”,.Mobile Data Management (MDM) Hong Kong, China, Springer (2001)15-26.

[5] Jungmayr, S., “Testability measurement and software dependencies”. Proceedings of the 12th International Workshop on Software Measurement, Magdeburg, Germany, 2002.

[6] Jackson, D., “Module dependences in software design”. Radical Innovations of Software and Systems Engineering in the Future: 9th International Workshop, RISSEF 2002, Venice, Italy. 198-203, 2004.

[7] Fowler, M., “Reducing coupling”. IEEE Software, 18(4), 102-104, 2001.

[8] Ameller, D. and Franch, X. (2008). Service level agreement monitor (salmon). In ICCBSS ’08: Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008), pages 224–227, Washington, DC, USA. IEEE Computer Society.

[9] Flehmig, M., Troeger, P., and Saar, A. (2006). Design and integration of sla monitoring and negotiation capabilities. Adaptive Services Grid Deliverable D5.II-7

[10] Bodenstaff, L., Wombacher, A., Reichert, M., and Jaeger, M. C. (2008). Monitoring dependencies for slas: The mode4sla approach. In IEEE SCC (1). IEEE Computer Society.

[11] Ludwig, A. and Franczyk, B. (2008). Cosma an approach for managing slas in composite services. In Bouguettaya, A., Krueger, I., and Margaria, T., editors, ICSOC 2008, number 5364 in LNCS, page 626632. Springer-Verlag Berlin Heidelberg.

[12] X. H. Wang, D. Q. Zhang, T. Gu, H. K. Pung, “Ontology based Context Modeling and Reasoning using OWL”. In Proceedings of CNDS, 2004.

[13] T. Strang, C. Linnhoff-Popien, “A Context Modeling Survey”.In Proceedings of UbiComp, Workshop on Advanced Context Modeling, Reasoning and Management, 2004.

[14] A. Harter, P. Steggles, A. Ward, and P. Webster, “The Anatomy of a Context-Aware Application”. Proceedings of Conference on Mobile Computing and Networking (MOBICOM 1999).

[15] P. Gray, D. Salber, "Modelling and using sensed context in the design of interactive applications". 8th IFIP Conference on Engineering for Human-Computer Interaction, Toronto (2001).

[16] Ebling, M., Hunt. G., and Lei, H., 2001, "Issues for Context Services for Pervasive Computing", Workshop Middleware for Mobile Computing, Heidelberg, Germany, 2001.

[17] Stephen S. Yau, F. Karim,“Context-Sensitive Middleware for Real-time Software in Ubiquitous Computing Environments.”Proceedings. 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC2001). May 2001. p163 –170.

[18] Tobias Zimmer, “Quality of Context: Handling Context Dependencies”. Proceedings of 2nd International Workshop on Personalized Context Modeling and Management for UbiComp Applications, California, USA, Sept. 2006.

[19] Yokoo, M., E. H. Durfee, T. Ishida, and K. Kuwabara: 1998, 'The Distributed constraint satisfaction problem: formalization and algorithms'. IEEE Transactions on Knowledge and Data Engineering 10(5), 673-685.

[20] F. Bacchus and P. van Beek, “On the conversion between nonbinary and binary constraint satisfaction problems,” in Proc. 15th AAAI Conference on Artificial Intelligence, Jul. 1998.

[21] Aspvall, Bengt; Plass, Michael F.; Tarjan, Robert E. (1979), "A linear-time algorithm for testing the truth of certain quantified boolean formulas", Information Processing Letters 8 (3): 121–123.

[22] T. Feder, “Network flow and 2-satisfiability,” Algorithmica, vol. 11, no. 3, pp. 291–319, 1994.

[23] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Progress on the state explosion problem in model checking,” in Informatics, 2001, pp. 176–194.