Agent-Based Context Management for Service-Oriented Environments

Adrija Bhattacharya, Avirup Das, Sankhayan Choudhury, Nabendu Chaki

Agent-Based Context Management for Service-Oriented Environments

Adrija Bhattacharya, Avirup Das, Sankhayan Choudhury, Nabendu Chaki

University of Calcutta, India
adrija.bhattacharya@gmail.com, avirup0310@gmail.com,
sankhayan@gmail.com, nabendu@ieee.org

Abstract. Context is an important aspect towards service discovery and selection. It is represented by a set of quality parameters. Any change in value of any one of the context parameter's (CP) changes the entire context. Relevance of the discovered services is often measured by similarity between service context and user's context. If these two does not match for a particular user's query; then corresponding services cannot be invoked or even if invoked, would perform poor. This paper proposes a novel context management framework. This holds the context information within a domain in a structured way such that the service discovery mechanism works faster as well as yields better result in terms of relevance of services specific to the queries from user. Autonomy, reactivity, and veracity properties of an agent help in achieving improved dynamics for the proposed framework. Implementation of the concepts and a comparative study is also reported. The proposed framework performs well with respect to search time, population size as well as varieties of queries.

Keywords: context, service discovery, agent programming, search time

1 Introduction

Context plays a very important role in service engineering. The relevance of discovered services is highly dependent on the context from which the service is being searched. Context is described as collection of some parameters {Context Parameter (CP)}, either qualitative or behavioral parameters, and their specific values. The context of a service actually answers some 'wh' questions (Who, Where, Which, etc) and also hold some additional information. In this paper, each context is described as a tuple that represents a specific instance of those parameters. Any change in any of the parameter values indicates the change of context. There are two types of contexts; service context and user's context. Service context is declared at the time of service creation. It is included within the description of services and generally static in nature. In other words, the services have fixed answers with respect to the 'wh' questions. A single service can have multiple service contexts; i.e., that service can be invoked in multiple specified contexts. Similarly, user's context is about the circumstance under

which the user queried or the services would be consumed. User context may be dynamic in nature. However, at a single point of time same service can satisfy multiple user queries with different context requirements.

A service can only be invoked if its service context matches the user's context of the query. Alternatively a service's performance varies over the different user contexts. Thus, to serve users more efficiently (by returning more relevant set of services), there is a need of new context management system. The importance of the system is both in terms of speed of searching and relevance of search results.

There has been some works [8, 9 and 10] that aim to grab the user's context and match with service context. In [8] EASY has been proposed which considers only QoS (Quality of Service) parameters as key factor to judge at the time of service discovery and selection. This is held incomplete in the sense of searching more appropriate service according to user's context. The context has more parameters compared to only quality parameters. A graph matching based context aware system is also developed in [9] that use non functional information to match service and user's context by developing concept graph and their matching. These two methods neither consider all service contexts that can be queried for, nor these methods can identify relations among contexts that could be used further to make searching better. In [10], services are structured by hyperspace analogue to context (HAC) information for identifying the changes in user context and adapt accordingly. This method also failed to consider all valid service context and according user context. This approach [10] is also very application-specific, where changes in contexts are more emphasized.

Agent is an entity which has the property of autonomy, social adeptness, reactivity, pro-activity, mobility and learning capability [1]. An agent perceives from the environment and act on the environment autonomously as well as independently. Sensing (i.e. perceiving some information from the environment), Reasoning (after perceiving, the reasoning ability of agent takes decisions) and Action (is about actions on the environment to interact autonomously) are basic features of agent[2]. An agent senses its environments by its sensor. Sensors basically collect information in any form from the environment. An agent determines the current state of the environment based on sensor provided information and takes decision about suitable actions by the reasoning ability. The environment from the perspective of an agent changes continuously. Even the environment may change within the running time of an agent. However, an intelligent agent must have the flexibility to interact with the environment at run time also [3, 4]. Agents can form a community and interoperability among the members of the community leads to achieve a collective goal.

In this paper, a novel framework is proposed for efficient context management and service provisioning. This framework contains a model in the form of a hierarchical structure with some special properties for holding context information. Autonomous feature and the reasoning ability of an agent is exploited and used in the framework for service discovery. Agents generally communicate among them by message passing. This feature helps in finding appropriate matches according to user query by method of backtracking. Sharing of each combination's information within the hierarchical structure at run time is actually needed. Thus, the adaptability feature of agent is also needed in the proposed hierarchical model implementation.

Theory behind the hierarchical structure is discussed in brief in section 2 of this work for the sake of completeness. Section 3 presents a comparative study on agent based architectures. An emphasis is given to justify the selection of BDI (Belief-Desire-Intention architecture) for agent based implementation of the framework discussed in section 2. Section 4 describes the detailed implementation. Section 5 illustrates the performance and finally section 6 concludes.

2 Proposed Framework for Context Management

In this section we will discuss the proposed mechanism for context management in brief. The motivation behind the research work is discussed in previous section. The framework consists of a model that is nothing but a hierarchical structure. The framework consists of multiple levels. It holds the contexts information in a structured way. Each level contains multiple 'context-node'. Each node may have one or more parents adjacent to the next level. It helps in searching the all options across the levels. Here comes some sense of hierarchy with respect to contexts. Services within a closed domain can be arranged according to their described context within the unique structure.

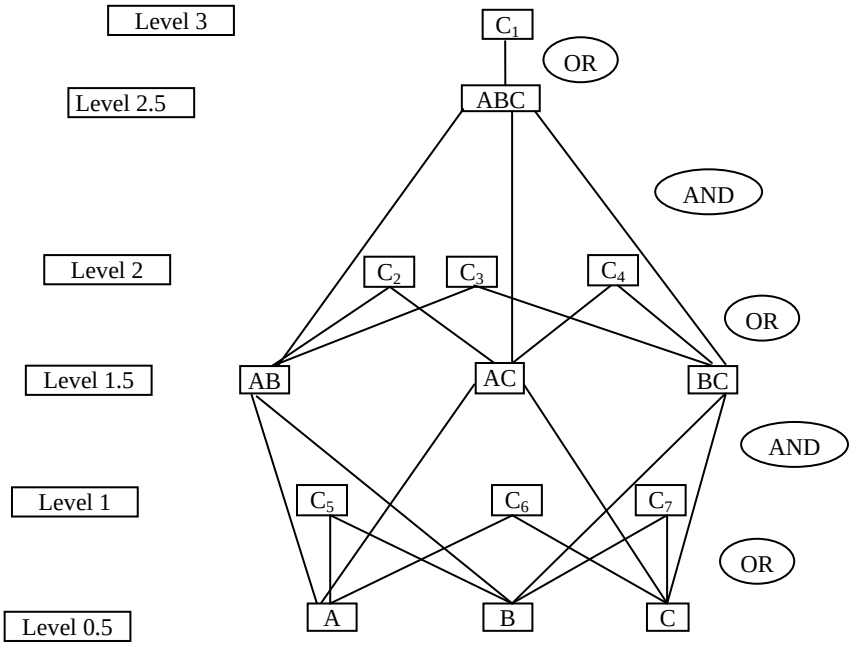

Fig. 1. Hierarchical Context Structure

At the time of matching user's context with that of the service; this structure is explored. This is to note that the structure is constructed previously at design time. However, it is explored at runtime, i.e., according to user's query. Figure 1 describes

the structure for three goal levels (1, 2, and 3) and three intermediate levels (0.5, 1.5, and 2.5). It can be populated further.
At level-0.5, the contexts are specified by only one context parameter. These nodes are ORed at level-1. There are 3 context-nodes in level-1 and each node has some service and their context information integrated at a point. After that level-1.5 contains three nodes that are basically produced by ANDing level 0.5 nodes. Again level-2 has three context nodes and that are related with the level-1.5 nodes by OR. In each node the contexts are derived by generating combinations of the level 1.5 nodes. Level 2.5 again contains one intermediate node that is produced by ANDing level-2 nodes. Similarly, level-3 has by default only one node i.e. produced by ORing with only one previous level node; this node contains all the common contexts of all previous nodes. This way each level of the hierarchical structure is developed.

The offline formed hierarchical structure is searched based on the context information present in the users query online. Thus the structure of the services formed according to service context information, must be able to match service and user contexts in lesser time. The goal contexts are only at levels (1, 2 and 3). In figure 1context goals are defined by C1 to C7.The efficiency of the service hierarchical structure is that the valid service contexts are readily available in the structure and the hierarchy between consecutive level nodes provisions for switching among levels in case of context mismatch. The structure actually helps to decrease search time and fine tune the search results with various over varying subsets of contexts.

3 Different Agent Architectures

Reactive, Deliberative, BDI are three broad types of classifications of agent based architectures. Reactive agents only can sense the environment and respond accordingly. It can be easily modeled by if-then-else [5]. It can be called as a lower level abstraction and gives fast response. Deliberative agent architecture consists of deliberative agent sensing the environment and then to decide the next action using logical reasoning and pattern matching [11]. Actually this act of adaptively is called deliberation and it helps to design complex systems. A very powerful high level abstraction tool for designing and implementing a complex multi-agent system is BDI. It uses practical reasoning to achieve its desired goal. There are three different modules with respect to agents; that are Beliefs, Desires and Intentions. This architecture supports deliberation with means-end reasoning [7]. Among all of three plans a comparative study has been made to judge the most suitable architecture for implementing hierarchical structure with agent based programming. Table 1 contains the comparisons among three basic agent architectures.

Table 1 clearly states the superiority of BDI architecture with respect to reactive and deliberative agent architecture. The most important factor of BDI is that it enables the multi agent architecture and thus to build a complex structured system like hierarchical structure it held indispensible. At each level of the hierarchical structure there exists an agent. It will not be possible to design with Reactive or Deliberative agent architecture. The plan selection and execution process of an agent is completely sepa-

rated and independent which makes it time efficient. The need of hierarchical structure also lies in optimizing the search time that can be possible through BDI. BDI is an event-driven architecture and maintains an even queue. Thus, the user query and each combination search are treated as a new event. All valid contexts are generated.

Table 1. Comparative study among popular agent architectures

Agent Architectures / Features	Reactive	Deliberative	BDI
Building Complex System	N	Y	Y
Plan library exists	N	N	Y
Separation between plan selection and execution	N	N	Y
Event-driven	N	N	Y
Simple Communication	N	N	Y
Supports Multi-agent system	N	N	Y
Supports Deliberation	N	Y	Y
Supports Means-end reasoning	N	N	Y
Supports Deliberation & Means-end reasoning	N	N	Y
Dynamically changes intentions	N	N	Y
Multiple desires active simultaneously	N	N	Y
Strong Negation	N	N	Y
Rules	N	N	Y
Failure Handling	N	N	Y
Internal Actions	N	N	Y

Message passing is another essential feature of agents. Through messages an agent can shares its beliefs, goals and can ask about the situation of another agents which helps to implement hierarchical structure; as in the hierarchical structure if match at one level agent fails then it communicate to its previous level agent. BDI architecture has flexibility to dynamically update its intentions. That helps in implementing the backtracking in match algorithm in hierarchical structure. Due to the important rule that enables BDI agents to derive knowledge from existing knowledge, the Search mechanism is claimed to be efficient enough. There exists a set of BDI languages that supports different internal actions. All of the external features for construction of hierarchical structure as well as exploiting the hierarchical structure based on user query more or less can be mapped by the internal actions of the BDI language actions.

4 Implementation and Mapping to BDI agents

In general, in BDI architecture an agent consist with four data structures i.e. Be-liefs, Goals, Plans and Intentions. Beliefs are represented the informative part of an agent that defines what an agent knows about the environment and itself. Goals or desires are represented the motivational part of an agent. It defines what an agent wants to achieve. Plans are represented as set of procedural knowledge and decide how an agent can achieve a desired goal. Each agent has a plan library that stores all the plans related to different situations. The plan library is consulted for a specific goal and a

set of plan related to the goal is selected. Then the context part of a plan is checked using some expression evaluation and depending on beliefs. A specific plan among the selected set is chosen through the necessary matching of the context part of the plans and it becomes an Intention. If goal is a new arrived goal then the selected intention creates a new intention stack. But for a sub-goal, the selected intention is pushed at the top of the existing intention stack. Only then the Intention stack is selected and executed. It is an event-driven architecture since it maintains an event queue. This queue stores events which are perceived from the environment and subgoals which are generated by executing another Goal as an event. A plan is represented as an event with two parts, a context part and the action part. In this particular agent paradigm the beliefs are basically the constraints of context parameters. Collectively some beliefs determine a typical context instance. The rest part of an event is singleton; that is an action or subgoal. It looks like,

Event: $belief_1$ & $belief_2$ & …. & $belief_n$ ← actions / sub-goals

Using means-end reasoning an agent decides how to achieve a desired goal and the output of this reasoning are intentions. Here in the implementation of hierarchical structure by BDI agent based architecture, each goal level of the hierarchical structure is considered as agents. The information on context of services is stored in service database. It is mapped into an agent's beliefs and the user query into a goal of an agent. There exists a plan library consisting of different service records with context specifications, based on which the hierarchical structure is constructed offline (i.e., at design time). Now by means-end reasoning (incorporated within BDI) a most suitable step to response the user query is selected. The hierarchical relations among the different values of the same context parameter (CP) are expressed by derived knowledge of the BDI agents. Such as location parameter can take both the values 'Kolkata' and 'Westbengal'. But there exists a relationship among these two: Kolkata is a city in west Bengal. Thus, if a service has location context parameter as 'Westbengal'; it can be invoked from Kolkata also, but the reverse is not true. Again, it helps in backtracking at the time of searching services matching the user context in a query. BDI languages (such as AgentSpeak) give some internal actions like .print, .send, .broadcast which helps us to manipulate the output and messaging. In this implementation .print internal function is used to show the output and .send internal function to send a message from one agent to another agent and .broadcast internal function sends one message to all of the agents at a time.

Construction of the hierarchical structure is done mainly based on the two basic operations AND and OR. Agents at each level have information about the context combinations and their details service containment. Each agent is working as a level of abstraction for whole service data registry. In the implementation of the structure, all services in the service repository are represented as a belief of the agents. A service is represented in the belief base with N+2 fields (i.e. Id of the service, Context parameter1, Context parameter2, Context parameter3,…, Context parameter N, Functionality of the service) where N is the no. of context parameters as follows:

ID	CP_1	CP_2	CP_3	…	CP_n	Functionality

The following sub section illustrates the design details for service hierarchical structure and search procedures with the help of agent based programming using BDI.
Each goal node of the structure contains the services which have the same context parameters as the node have. Services in each node are arranged context wise; this helps in fast context matching and relevant service retrieval.

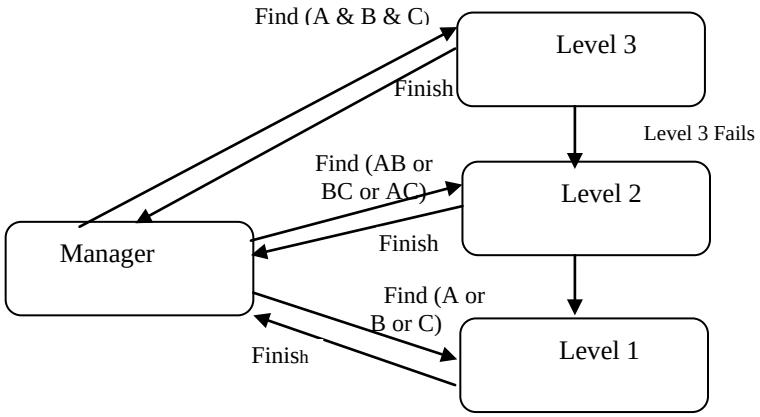

Fig. 2. Agent architecture and components

4.1 Basic layout of agent based architecture

The proposed framework is implemented with Multi-agent programming. Here define N+ 1 agent, when the number of context parameters is N. For example, if there are 3 CPs, then 4 agents are defined to handle the whole structure. For the hierarchical structure consisting of 3 CPs (A,B and C) the agents with their functions are depicted in figure 2. Different context parameter combinations generated are A, B, C, AB, AC, BC, ABC. At level-3 all context parameters are present (ABC) altogether. i.e. in this level context is defined by three CPs. In level-2, the responsible agent has information about the nodes in that levee specifies on CP combination. Here context is declared by two parameters at time. In this level there are three nodes (AB, AC and BC). Similarly at level 1 only (A, B and C) three nodes are there representing corresponding context (specified by one parameter at a time). The following subsection detailed out the agent's working algorithms.

4.2 Discussion on algorithm for different agents

The algorithm for manager agent and other level agents are described in this subsection. In figure 3 the flow chart for a manager agent is described. When a user query arrives then, at first, the Agent-manager is initialized with the input parameters of the query. Then number of context parameters was declared as variable N. In case of searching at level N, the corresponding agent algorithm differs and becomes complex. An agent at Level N executes for user requested services at that level only. As shown

in figure 2, if it fails then the search procedure is routed towards the next lower leve and so on. Similarly in figure 3 the flow of information and how it is all managed by manager agent is described.

In the manager algorithm, at first each of the CPs supplied by user's query is checked. The algorithm goes forward for only the valid CPs. After that the manager agent routes the search to appropriate level agent depending on the number of parameters specified in the query.

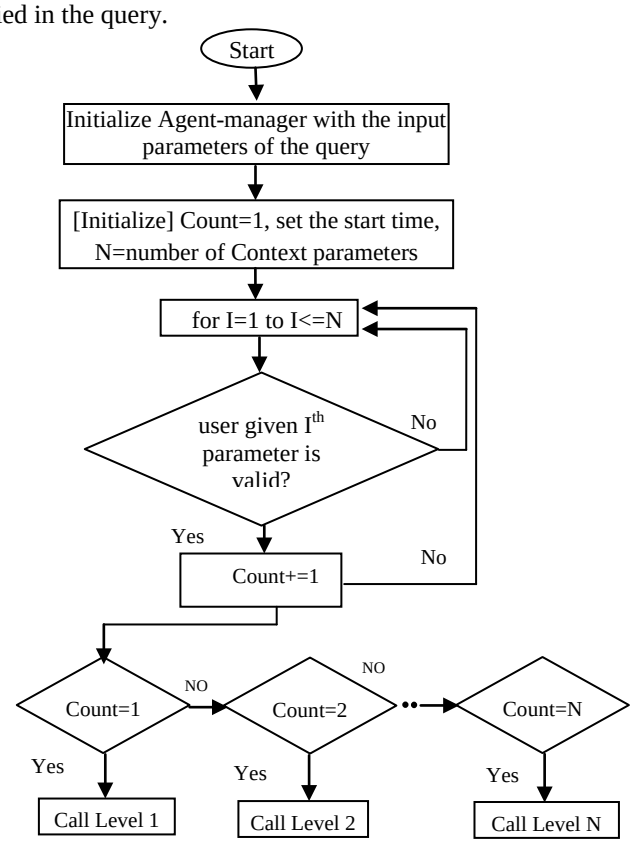

Fig. 3. Algorithm for Manager Agent

When the manager agent routes to a particular level agent; the search procedure starts. At first, the dedicated level agent searches according to the values of CP specified in the query. If it fails to find out any services matching user's requirement; it routs the search information to the next lower level agent. Then that agent also searches in the same way and so on. Algorithm for a any level agent is described in figure 4. It works in a generalized way for all level agents. In this algorithm the query CPs are collected and calculated that how many CPs are involved within the query. Then it collects all the services from the required node and checks individual CP specification of services. If the specification is same as in the user's query then the service is chosen as a relevant one. After successful completion of all level agents; the set of retrieved rele-

vant services goes to the manager agent. Manager agent then decides and sends the relevant set of service to the user. A few methods and objects are declared in the figure for the sake of easy understanding of the complex algorithm.

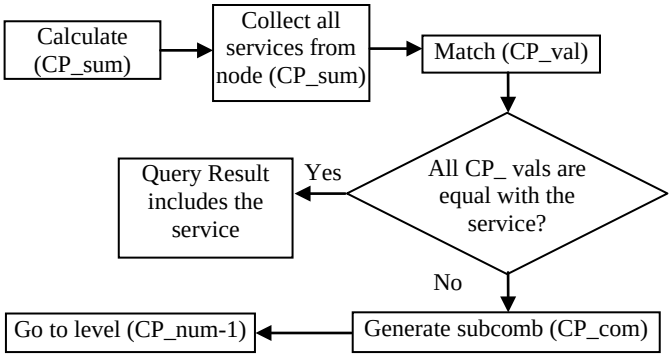

Fig. 4. Flow diagram for searching by any level agent

The definitions of those methods and variables are defined as follows:

CP_num: This is an integer value that denotes total number of CPs

CP_ sum: This is an integer value. It generates unique sum for each set of CP combination

CP_val: This may be any string. It denotes value of each CP. i.e. CP_1="c_1" CP_2="c_2" CP_3="c_3",…, CP_k="c_k" here all c_i's are considered as CP_val

CP_com: This is a particular cp combination, i.e., CP_1 CP_2 CP_{k-1}

Node(X): Node whose CP_sum is X

Level (X): It denotes the Level -X

Match(X): Here X is a set of CP_val. This method matches each CP_val of query with that of services stored previously.

Subcomb(X): X is a CP_com. Subcomb generates all possible sub components having CP_num-1 elements at a time from X. As for example, if X is ABCD then Subcomb will generate ABC, ABD, ACD, and BCD.

5 Performance analysis

The construction of the hierarchical structure is done offline. Thus the overhead of construction is little. The structure is used for context matching and corresponding set of relevant services for dynamic user query. Complexity of the search mechanism is reduced by generating unique sum of CPs, i.e., in case of searching a particular CP the search procedure instantly finds out the CP combination that is matching with the combination provided in user query. This involves O(n) time complexity where n is the number of NFPs specified in user query.

A multi agent implementation of hierarchical structure is done with the notion of Belief-Desire-Intension architecture. The corresponding language used is AgentSpeak and complied in JASON 1.3.8 [6]. Each level of the hierarchical structure is managed

by individual agents. An additional agent is responsible for coordinating among different level agents.

Table 2. Specifications of the agent based system

Specifications	**Used components**
Agent architecture	Belief-Desire-intention
Language used for agent programming	AgentSpeak language
Complier	JASON ver 1.3.8
Supported codes written	JAVA with JDK 7
Form Design	VB 6.0
Operating System	Windows XP
RAM size (minimum)	512 MB
Disk space required (minimum)	40 GB

A multi agent implementation of hierarchical structure is done with the notion of Belief-Desire-Intension architecture. The corresponding language used is AgentSpeak and complied in JASON 1.3.8. Each level of the hierarchical structure is managed by individual agents. An additional agent is responsible for coordinating among different level agents. Table 2 describes the system specifications. This experiment has been worked on almost 3000 Health care services. A portion of the list typically looks like figure 5, which is a snapshot of the partial belief base used in the agent based system.

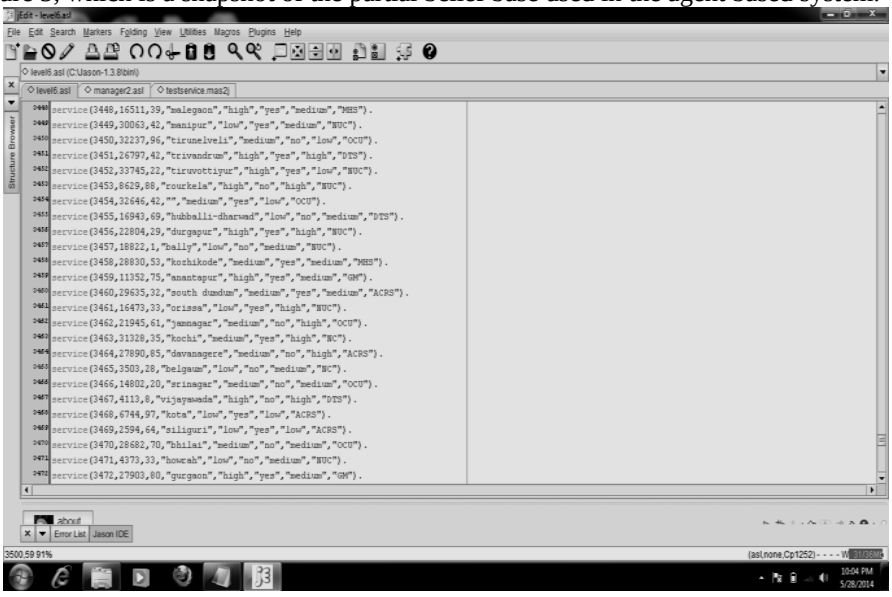

Fig. 5. Snapshot of the Belief-Base

A set of 70 queries were prepared (typically looks like Table 3). Here each column contains context parameter and corresponding value (from second to eighth column). Each tuple in the table is a query. The functional need of each query is specified by

the value at column 1 and the rest is specified as the context. Randomly selected queries were run and a comparative study has been made between usual process and proposed hierarchical match model performances.

Table 3. Sample query set

Functionality	**Relia-bility**	**Insurance rights**	**Where care needed**	**Accuracy**	**Cost**	**Waiting time**	**Location**
Accidental care	"yes"	"yes"	"body"	"high"	20000	25mins	Burdwan
Diagnostic	"yes"	"no"	"body"	"medium"	12000	55mins	Kolkata
Nursing Care	"yes"	"no"	"arm"	"high"	1200	25mins	
Emergency	"yes"	"yes"	"arm"	"high"	1200	1hrs	Chennai
Diagnostic		"yes"	"body"		1500	2 hrs	Kolkata
Mental care	"yes"	"no"			1500	2hrs	Mumbai
- - - - - - - - - - - - - - - -							
Neonatal care	"yes"	"no"	"leg"	"high"	1700	1hrs	W. Bengal
General Medicine	"yes"	"yes"	"leg"	"high"	1700	25mins	Kolkata
Oldcare	"yes"			"high"	12000		Chennai
General Medicine	"yes"			"high"	9000	1 hrs	W. Bengal

A comparison between the two has been done in two manners. In first case, the queries are selected randomly from a wide varying query set. In any case our proposed mechanism works better with respect relevance of result. Another side of the comparison is time based. With the increasing size of service population; our proposed mechanism takes less amount of time. Thus the proposed mechanism works in an optimized way with respect to size of service population, wide varying context scenarios and off course for search time. The comparison graph is shown in figure 6.

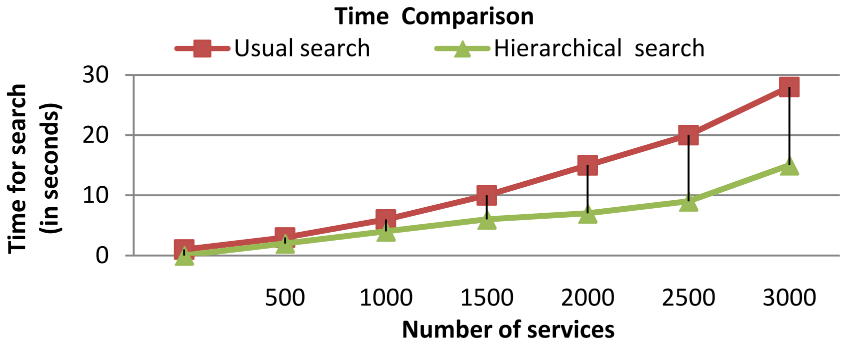

Fig. 6. Comparative Performance Analysis

6 Conclusions

Contexts defined in terms of service parameters for the users as well as services are to be matched before service provisioning. In this work, an effective Framework to

represent all possible service contexts within its hierarchical structure is proposed. The offered hierarchical structure is built offline and is utilized for finding the most relevant services in terms of user context. The performance of the proposed framework is verified by an exhaustive simulation through agent based approach. As for experiment, the hierarchical structure is populated with service size 3500 and returns results within a feasible amount of time with respect to the existing alternative solutions. The most important contribution of this paper is that it delivers results considering all possible sub sets of CPs supplied in query and that will be beneficiary for the user as it offers the nearest match (may not be the exact match) based on his(her) choice. Thus the proposed framework should be considered to work in more flexible way, compared to their counterparts.

References

1. Nicholas, R., Jennings and Michael Wooldridge; “Applications of intelligent agents”, pages 3-28. New York, Inc., Secaucus, NJ, USA, Springer-Verlag (1998)
2. Getchell, Adam. "Agent-based modeling." *Physics* 22, no. 6 (2008): 757-767.
3. Wiebe van der Hoek, B., van Linder, John-Jules Ch., Meyer: An integrated modal approach to rational agents. In Proceedings of 2nd AISB Workshop on Practical Reasoning and Rationality, pages 123-159 (1997)
4. Wiebe van der Hoek, Michael Wooldrige: Towards a logic of rational agency. Logic Journal of IGPL,11(2):135-159 (2003)
5. Michael Wooldridge, Nicholas R. Jennings.: Intelligent agents: Theory and practice. Knowledge Engineering Review, vol. 10 (2):115-152. (1995)
6. Rafael, H., Bordini, Jomi Fred Hubner, Michael Wooldridge.: Programming Multi-Agent System in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons (2007)
7. Anand S. Rao, Michae,l P., Georgeff.: BDI agents: From theory to practice. In Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95), pages 312-319, San Fransisco, USA. (1995)
8. Mokhtar, S.B., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y. :Easy: efficient se-mantic service discovery in pervasive computing environments with qos and context sup-port. J. Syst.Softw. 81(5), 785–808 (2008)
9. Manuele Kirsch-Pinheiro, Yves Vanrompay, Yolande Berbers.: Context-aware service selection using graph matching 2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC'08), ECOWS. CEUR Workshop proceedings, volume 411. (2008)
10. Katharina Rasch, Fei Li, Sanjin Sehic, Rassul Ayani , Schahram Dustdar.: Context-driven personalized service discovery in pervasive environments. Springer Journal on World Wide Web Volume 14, Issue 4 , pp 295-319. Print ISSN 1386-145X DOI 10.1007/s11280-011-0112-x. (2011)
11. Castelfranchi, Cristiano, et al.; “Deliberative normative agents: Principles and architecture" Intelligent Agents VI. Agent Theories, Architectures, and Languages. Springer Berlin Heidelberg, 364-378. (2000)