I.J. Information Engineering and Electronic Business, 2011, 1, 30-37
Published Online February 2011 in MECS (http://www.mecs-press.org/)

Research of the QoC based Middleware for the Service Selection in Pervasive Environment

Di Zheng
Department of Computer Science, Naval University of Engineering
Wuhan, China
dizheng@nudt.edu.cn

Jun Wang
Key Research Lab, Wuhan Radar Institute
Wuhan, China
junwang@nudt.edu.cn

***Abstract*—With the rapid development of information technology, it is inevitable for the distributed mobile computing will evolve to the pervasive computing gradually and whose final goal is fusing the information space composed of computers with the physical space in which the people are working and living in. Furthermore, with the development of SOA, more and more pervasive applications have been composed of different kinds of services. So how to choose a suitable service from all the useable services is the most important step for pervasive computing. Compare to the traditional service selection we must take more care of the context as well as the quality of them in pervasive environment. However, most of existing researches pay no attention to QoC(Quality of Service) which may lead to unreliable selections. Therefore we proposed a middleware based service selection scheme to support QoC-aware service selection efficiently.**

***Index Terms*—-QoC; middleware; Context-aware; Pervasvie**

I. Introduction

Nowadays, the vision of pervasive computing is becoming a reality. The paradigm for pervasive computing aims to enable people to contact anyone at anytime, anywhere in a convenient way. So, context-awareness has become one of the core technologies in pervasive computing environment gradually and been considered as the indispensable function for pervasive applications[1].A context-aware system generally consists of two parts: sensing a context scenario, and adapting the system to the changing context scenario by providing desired services for a user.

Furthermore, with the development of SOA, more pervasive applications begin to provide users with cost-effective services that have the potential to run anywhere, anytime and on any device without (or with little) user attention. Such services are usually called pervasive services and they are part of pervasive (or ubiquitous) computing [2] .

Existing Service Selection methods such as Mobile Agent, Bluetooth, Diane, Jini, Universal Plug and Play (UPnP) have proposed a number of solutions on how to deal with the problems including the semantic description, service discovery, system architecture, service binding in the service selection problem, and these programs focus on the different considerations. Based on these researches, a lot of work has been carried out for service selection [3-7] in pervasive environment. However, these systems rarely pay little attention to the Quality of Services for pervasive applications. In [17], Kun give a QoS based service selection method for pervasive applications by thinking about the contexts of Network quality, Node availability, Network delay, Network reliability, User-perceived quality and so on. By using these criteria they can complete the selection of services. However，how we get the values of the criteria? All the values are so-called service contexts coming from different sensors too. We should pay attention to the service contexts themselves. For example, two services have the same values for all the criteria. Then we should choose which one? In fact, the accuracy of service A’s contexts is 70% while the other one is 85%.It’s obviously the latter service is a better choise. So we should pay attention to the Quality of Context information (QoC) of the services when we complete the service selection.

Buchholz et al. [8] has been the first ones to define QoC “as any information describing the quality of information that is used as context”. Furthermore, context information can be characterized by certain well-defined QoC aspects, such as accuracy, precision, completeness, access security, and up-to-date [12]. Despite its importance few works [9~13] have proposed different QoC measuring methods. Moreover, these studies evaluate quality only on some aspects, i.e. they do not consider complex and comprehensive applications. Comparatively, pervasive environments have a wider range of applications such as performing collaborative work. Hence, complex data structures are used to gather data from sources ranging from the simple sensors to user interfaces and applications in mobile devices.

Footnotes: 8-point Times New Roman font;
Manuscript received January 1, 2011; revised June 1, 2011; accepted March 1, 2011. Copyright credit, project number, corresponding author, etc.

Therefore, based on our previous works [14-16], we propose a context-aware framework that support QoC management including threshold control, duplicate context discarding and inconsistent context discarding in various layers. Moreover we extend our autonomic management approach for the service selection so as to make these applications can be selected more accuracy and efficient than traditional methods.

II. MIDDLEWARE BASED QOC-AWARE SERVICE SELECTION

A. Architecture of the Context-aware Middleware

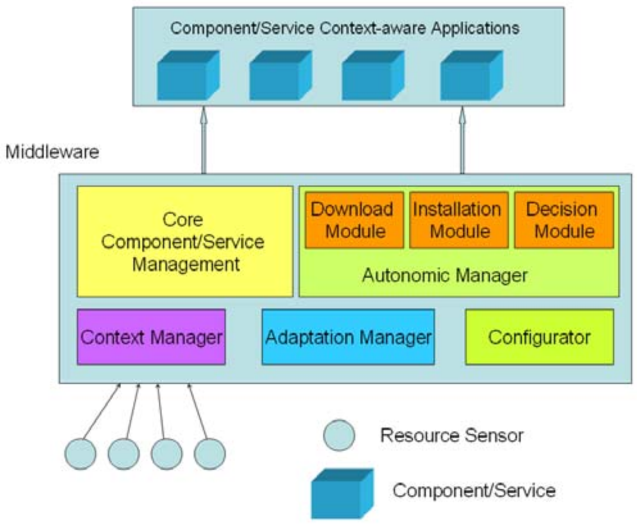

Figure 1. Adaptive Fault Tolerant Architecture

As our previous architecture depicted in figure 1[14,15,16], the core provides the fundamental platform-independent services for the management of the component/service based applications such as component deployment, service discovery, service combination and so on.

Context Manager is responsible for sensing and capturing context information and changes, providing access to context information (pull) and notifying context changes (push) to the Adaptation Manager. The Context Manager is also responsible for storing user needs and preferences on application services.

Adaptation Manager is responsible for reasoning on the impact of context changes on the application(s), and for planning and selecting the application variant or the device configuration that best fits the current context. As part of reasoning, the Adaptation Manager needs to assess the utility of these variants in the current context. The Adaptation Manager produces dynamically a model of the application variant that best fits the context.

Configurator is responsible for coordinating the initial instantiation of an application and the reconfiguration of an application or a device. When reconfiguring an application, the Configurator proceeds according to the configuration template for the variant selected by the Adaptation Manager. Thus, the Configurator carries out the adaptations decided by the Adaptation Manager by applying the configuration template.

Autonomic Manager provides the basis for realizing the dynamic, automatic binding of components/services into concrete functionality as well as the dynamic replacement of a component/service with another. The Download Module deals with the orchestration of the software transfer to the system, and other procedures, i.e. asserting the authenticity of the concerned component's source, and integrity checks. The Installation Module caters for post-download steps. The Decision Module defines certain actions and decisions for the configuration of the autonomic system, after evaluating its behavior.

B. Architecture of the Context Management

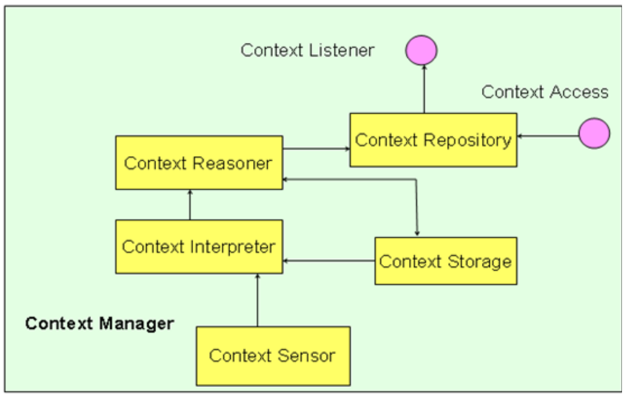

Figure 2. Architecture of the Context Manager

As depicted in figure 2, the context repository is the main entry point for clients to the context manager. The primary tasks of the context repository are to maintain a context model, register and notify listeners, give access to context elements, and keep registry of available components.

The context sensors are components which provide context information to the context repository (a type of context source). Sensors can be wrappers around specialized hardware drivers, or legacy code used for monitoring context, such as battery, memory, and network information.

The context interpreter abstracts raw or low level context information into richer or higher level information according to interpretation rules described by using the context meta-model provided by the middleware. Furthermore, this component can fuse kinds of basic information into more comprehensive elements.

The context reasoners can produce one or more context elements using other context elements as input. This component is used to filter context information to determine relevant ones, and notify the subscribed component of these context changes.

The reasoners are "plug and play" in order to make it possible to target reasoners according to different needs and domains.

The context storage keeps the track of historical context information which is often required in order to determine trends in context data (for example trends in user behavior, network stability, etc).

C. Ontology based Context-aware Service Model

As depicted in figure 3 is the ontology based context model, the contexts are composed of computation contexts, location contexts, user contexts and activity contexts. The service contexts belong to the computation contexts. Service ontology is not connect directly to upper ontology's context information but waits exit process of executing service. Domain ontology is inherited context information from upper ontology and creates core ontology in domain-specific area. If service deduction arises in domain ontology, service ontology is created after service matching between domain-extend ontology and new service ontology, and executes it.

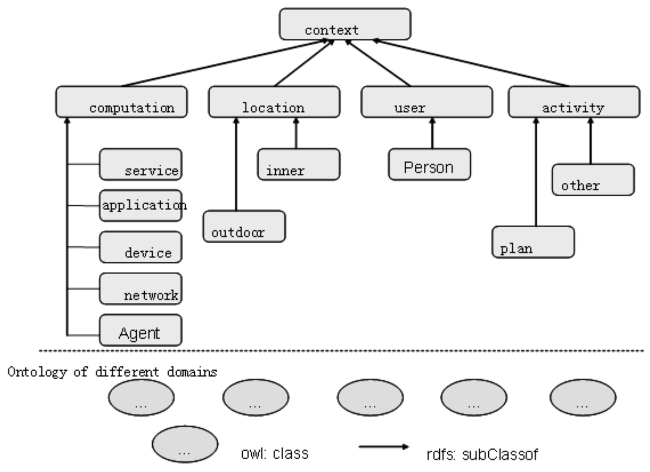

Figure 3. Ontology based Context Model

As depicted in figure4, we use the parameters as follows based on ontology:

(1). Security

We use security as the probability with which the context is delivered in security to the consumers. This parameter is useful to know the probability with which the context has been maintained in security, from its capture by sensors to its use.

(2). Precision

We use precision as the level of details in which the context characterizes the real world. For numeric context information, the value described with three significant figures (e.g. 32.2) is more precise than with two significant figures (i.e. 32).

(3). Resolution

We use resolution as the spatial granularity with which the context is being described/sensed from the environment. For instance, the lightness of a building can be described in the following spatial granularity levels: building, floor, and room.

(4). Freshness

We use freshness as the temple granularity with which the context is being described/sensed from the environment. This parameter reflects the time exhausted for passing the context to consumers.

(5). Certainty

We use certainty as the probability of the accuracy of the context. As we all know, the context comes from different kinds of sources and some of them more sensitive and useful than the others. So when we have similar contexts from different sensors, we should choose the right context with the help of certainty.

(6). Completeness

We use completeness as the ratio of the number of context information available to the total number of context gatherings. The completeness of a context object is computed as the ratio between the sum of the weights of available attributes of a context object, and the sum of the weights of all the attributes of that context object.

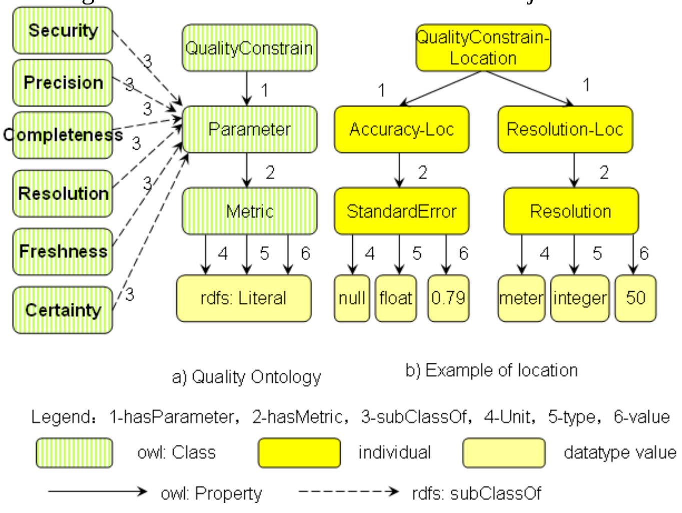

Figure 4. QoC-aware Context Model based on Ontology

D. Detection and Discardtion of Duplicate or Inconsistent Context

Besides the QoC factors we should also discard the duplicate and inconsistent context. In our system, every context has context ID, context name/value pairs. So we define duplicate contexts as the contexts have the same identifier, or the same name/value pairs. As described in Algorithm 1, after context gathering newly arrived context will be passed to check if it is the duplicate context.

Firstly we get the identifier of the newly arrived context and check if there is any context in the existing data representing the same entity. If we do not find any context having the same identifier, we will check the name/value pairs further.

If there are some contexts having the same identifier or the same name/value pairs, we will check the sources of context. If they have different sources then some errors may occur and we should check the gathering of the contexts. If these two context objects are from the same source then we check the time when these context objects are generated. If they have the same timestamp then it means that they are the exact duplicate of each other and anyone of them can be discarded as well as keeping the other one. If they have the different timestamps it means that these are the duplicate contexts and will be discarded.

Algorithm 1 Algorithm to detect duplicate context objects

```
INPUT: New arrived context
1.  get the identifier of contexts
2.  if  There exists contexts with same ID
3.  then
```

```
4.      if sourceID of both context objects match
5.      then
6.          if timestamp of both context objects match
7.          then
8.              Find duplicate contexts and discard anyone
9.          end if
10.     else
11.         Check the context gathering
12.     end if
13. else if There exists contexts with same name/value pairs
14.         Discard one according to the quality tuple
15.     end if
16. end if
```

After duplicate context dealing we need inconsistent context dealing including matching of name/value pairs and quality-aware tuple. Consistency constraints on contexts can be generic (e.g., "nobody could be in two different rooms at the same time") or application specific (e.g., "any goods in the warehouse should have a check-in record before its check-out record").

Firstly, we define the relations as follows to indicate the relation of the fields of different contexts:

$(1) identical(v1 = v2)$

$(2) equivalent(\text{equal in value})$

$(3) plug-in(equivalent \;\; or \;\; E(v1) \subset E(v2))$

$(4) covering(equivalent \;\; or \;\; E(v1) \supset E(v2))$

$(5) overlapping(E(v1) \cap E(v2) \neq \varnothing)$

$(6) unrelated(E(v1) \cap E(v2) = \varnothing)$

Secondly, in our consistency checking, each constraint is expressed by an *FOL* formula as given in Figure 5, where *bfunc* refers to any function that returns true (T) or false (F). Each expressed constraint is called a *context consistency rule* (or *rule* for short). Note that we are only interested in well-formed rules that contain no free variables.

$$\begin{aligned} formula ::= & \forall var \in pat(formula) \mid \exists var \in pat(formula) \mid \\ & (formula) and (formula) \mid (formula) or (formula) \mid \\ & (formula) implies (formula) \mid not(formula) \\ & bfunc(var_1, ..., var_n) \end{aligned}$$

Figure 5. Rule Syntax

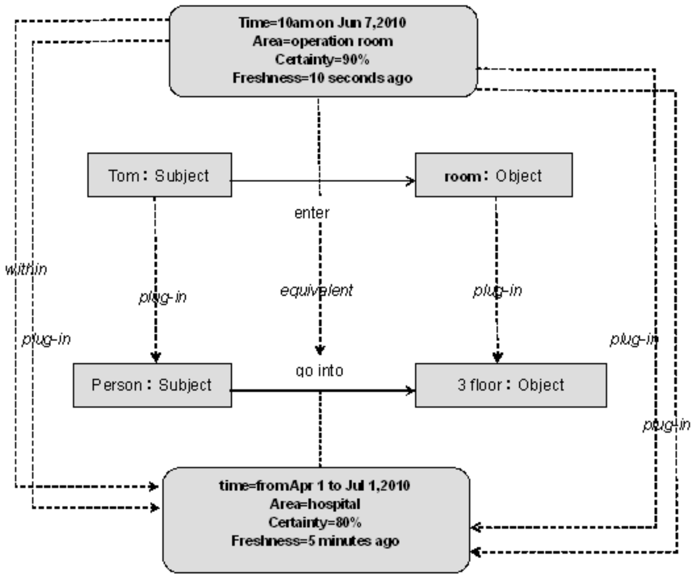

Figure 6. An example of context matching

The rule syntax follows traditional interpretations. For example, $\exists var \in pat(formula)$ the constraint that any context instance matched by pattern pat must satisfy formula. The formula definition is recursive until $bfunc$ terminals. Thanks to the expressive power of FOL, expressing complex constraints becomes easier than using ECA counterparts.

As depicted in figure 6 is the example of the inconsistent context matching. We get two different contexts about the event that Tom went into the operating room and we can find the difference between the certainty and the freshness fields.

Furthermore, we use triggers to find the inconsistence. As depicted in figure 7 is the example of the inconsistent trigger in the complex condition.

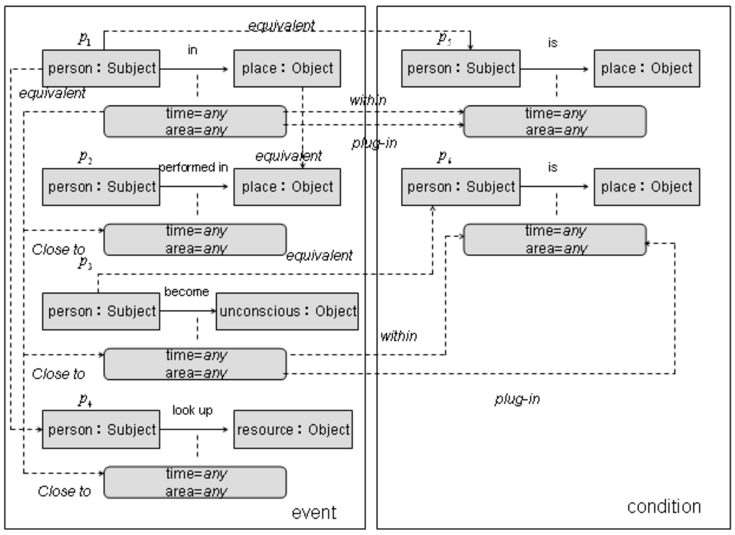

Figure 7. Complex context matching

After detecting inconsistent contexts we should use the algorithms as follows to discard the conflicted contexts.

Algorithm 2 Discarding all inconsistent context instances
INPUT: New arrived context

```
1.  get the new instance of context in the queue of matching patterns $pat\_que$
2.  To all the patterns $pat_1, pat_2, ...pat_n$
3.  In the pat's trigger $tgr$
4.    if exists $ins_1$ in $pat\_que_1$ , $ins_2$ in $pat\_que_2$ , ..., and $ins_n$ in $pat\_que_n$ and $tgr$ satisfy the constraint of $ins_1$ , $ins_2$ , $\cdots$, $ins_n$
5.      then
6.          if the constraint of tgr is satisfied
7.          then
8.            We get inconsistency
9.            **delete all the inconsistent context instances**
10.           add the remaining instances to the repository
11.         end if
12. end if
```

Algorithm 2 is deleting all the inconsistent context instances. However, the occurrence of many conflicts is due to the entrance of the new incoming context instance. So we get algorithm 3, discarding the newest context instance.

Algorithm 3 Discarding the newest incoming context
INPUT: New arrived context

```
1.  get the new instance of context in the queue of matching patterns $pat\_que$
2.  To all the patterns $pat_1, pat_2, ...pat_n$
3.  In the pat's trigger $tgr$
4.    if exists $ins_1$ in $pat\_que_1$ , $ins_2$ in $pat\_que_2$ , ..., and $ins_n$ in $pat\_que_n$ and $tgr$ satisfy the constraint of $ins_1$ , $ins_2$ , $\cdots$, $ins_n$
5.      then
6.          if the constraint of tgr is satisfied
7.          then
8.            We get inconsistency
9.            **delete the newest incoming context instances**
10.           add the remaining instances to the repository
11.         end if
12. end if
```

However, in some examples the newest context may be the right one, so we get algorithm 4 to discard the inconsistent context by the help of the field of certainty.

Algorithm 4 Discarding the context with lower certainty
INPUT: New arrived context

```
1.  get the new instance of context in the queue of matching patterns $pat\_que$
2.  To all the patterns $pat_1, pat_2, ...pat_n$
3.  In the pat's trigger $tgr$
4.    if exists $ins_1$ in $pat\_que_1$ , $ins_2$ in $pat\_que_2$ , ..., and $ins_n$ in $pat\_que_n$ and $tgr$ satisfy the constraint of $ins_1$ , $ins_2$ , $\cdots$, $ins_n$
5.      then
6.          if the constraint of tgr is satisfied
7.          then
8.            We get inconsistency
9.            **choose the context having the highest context and delete all the others**
10.           add the remaining instances to the repository
11.         end if
12. end if
```

In algorithm 4, we need compare the certainty of all the context instances and this may add the overall exhaustion of the algorithm. Therefore, to different kinds of contexts, we pay attention to the frequency of the contexts for the context having higher frequency may be right. Moreover, it is difficult to compare the frequency of different kinds of context. So we import the concept of relativity as follows:

$$context_i.rf = \begin{cases} \dfrac{context_i.period \cdot context_i.freshness}{currenttime - context_i.starttime} & (\text{dynamic context}) \\ \infty \ (\text{permanent context}) & \end{cases}$$

As in algorithm 5, we compare the relativity of different contexts and discard the ones having the lower relativity.

Algorithm 5 Discarding the context with lower relativity
INPUT: New arrived context

```
1.  get the new instance of context in the queue of matching patterns $pat\_que$
2.  To all the patterns $pat_1, pat_2, ...pat_n$
3.  In the pat's trigger $tgr$
4.    if exists $ins_1$ in $pat\_que_1$ , $ins_2$ in $pat\_que_2$ , ..., and $ins_n$ in $pat\_que_n$ and $tgr$ satisfy the constraint of $ins_1$ , $ins_2$ , $\cdots$, $ins_n$
5.      then
6.          if the constraint of tgr is satisfied
7.          then
8.            We get inconsistency
9.            **compute the relativity of the contexts and keep the highest context as well as deleting all the others**
10.           add the remaining instances to the repository
11.         end if
12. end if
```

E. QoC-aware based Service Selection

Firstly, based on the QoC-aware context model in figure5 and the algorithms to detect and discard the duplicate/inconsistent context, we propose a QoC-aware context manager as depicted in figure 8.

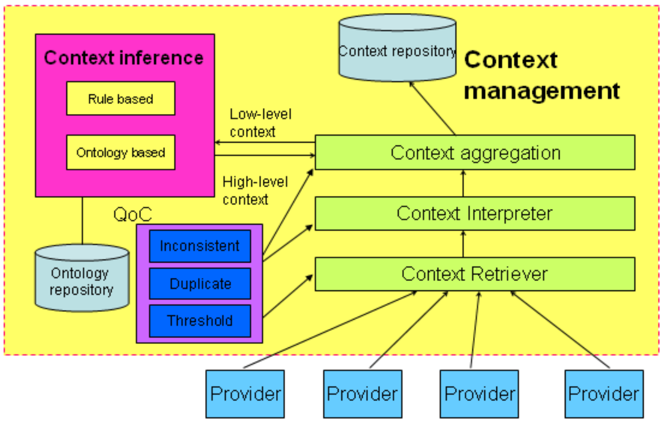

Figure 8. QoC-aware Context Manager

As depicted in figure8, we provide different levels of QoC. When the context retriever gets the context from different providers, it can use different thresholds to discard some of the context for its lower certainty, precision, freshness. Then we can get less and more useful context. Furthermore, the main question when completing context interpretation and context aggregation is the detection and discarding of the duplicate or inconsistent context. The detailed quality-aware context processing procedure is shown in Figure6.

The first step is the raw context gathering, in which raw contexts from various sensor sources are collected during a fixed short period. In this step we will use one or multiple thresholds to refinery the raw context.

The second step is the duplicate and inconsistency resolution during context interpretation and aggregation. We resolve inconsistency among different raw contexts in this step because duplicate and inconsistent raw contexts may lead to high-level contexts more difficult to handle. We process raw contexts in a batch by batch manner instead of a piece by piece manner. Inconsistency in a batch of raw contexts should be cleaned prior to context reasoning so that the inconsistency of high-level contexts can be mitigated in certain degree. We will update the context repository with raw contexts and check the dependency. Outdated or incorrect high-level contexts will be deleted in this step. If they are not removed, they will result in serious inconsistency among contexts after reasoning.

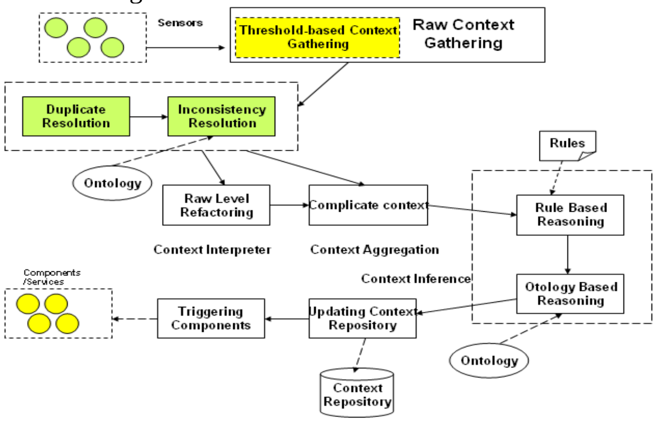

Figure 9. Quality-aware Context Processing Procedure

Based on the Context-aware technology, the Service Selection architecture obtains and using the context information. We can define context-aware criteria that link the two sides of context and service nonfunctional constraints. Context-aware criteria consist of a number of criteria that are initialized from the meta data of the correct service category. For example, we can use one of the QoC criteria such as precision, freshness, certainty, completeness and so on. We can also use two or more criteria to complete the selection. All the selection is managed by the autonomic manager according to the configuration of users.

All the procession is controlled by the Autonomic Management Module. When deployed, every service will have several context configuration constraints which imply the services pay more attention to which contexts. We can also set the thresholds for the constraints. If a service's contexts are under thresholds, then the service cannot be selected.

Initially, the Autonomic manager of an our context-aware middleware discovers the service providers that match the user's requirements by given service discover methods. Then the Autonomic manager compares QoC of different services by using the methods choose by users and selects one of the suitable providers and creates a binding to it. During service execution, when the Autonomic manager detects broken service bindings (e.g. the bound service provider becomes unavailable), it will repair them by discovering and binding to an alternative provider.

Besides the initial binding configuration and repair facilities, the Autonomic manager can be configured to continually optimize service selection during runtime. Furthermore, a service provider that was optimal in a certain context may be automatically replaced by a different service provider, which becomes the optimal choice in a new execution context. The context filter can be specified to trigger the dependency optimization each time a new service provider with the required specification becomes available. And we will take more complex replacement methods into account in the future researches.

F. Performance Measurements

We are deploying the proposed framework in a university building in order to provide context-aware services to the users, such as context-based access control and context-aware control of heating and lighting, among others. Afterwards, the gathered information was transmitted to a server running a CIS (Intel Core Duo 2.8GHz, 4 GB, Windows vista 32bits, SQL Server 2008). The sensing was carried out during 24 hours, with intervals of 5 seconds. The evaluation consisted of (i) a study of performance verifying the time overhead added by the quality support in the framework and (ii) the compare of the different discarding algorithms.

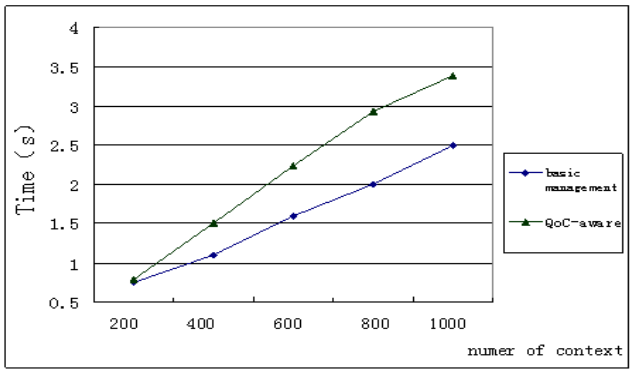

Figure 10. The Overhead of the QoC-aware Context Dealing

As depicted in figure 10, we compare the overhead of the common context processing procedure as well as the procedure with QoC-aware dealing. We can see the QoC-aware dealing may exhaust more time.

However, by using the QoC-aware dealing schemes, we can select better service. As depicted in figure 11, we compare the true probability of the contexts by using different contexts dealing algorithms such as selecting the newest service, selecting the service with the highest certainty and the service with the highest relativity as follows.

$$context_i.rf = \begin{cases} \dfrac{context_i.period \cdot context_i.freshness}{currenttime - context_i.starttime} & \text{(dynamic context)} \\ \infty \ \text{(permanent context)} \end{cases}$$

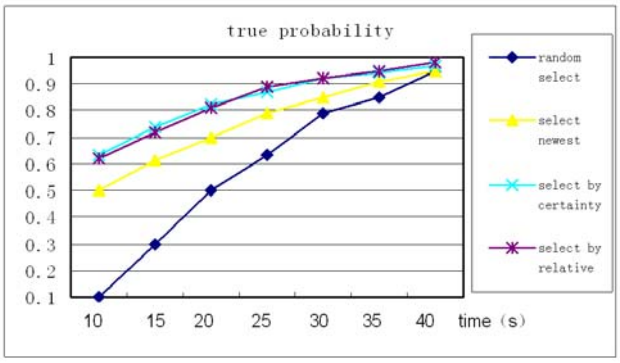

Figure 11. Analysis of the true probability

We can see all the dealing algorithms can get better true probability. If using the composition of more than one QoC criteria, we can make better selections. However, more time will be exhausted too. So, we should balance the accuracy and efficiency as well as making assure the difference of kinds of applications. In future works, we will complete more experiments continuously to find better service selection algorithms based on our framework.

CONCLUSION

With the rapid development of the information technology, it is inevitable that the distributed mobile computing will evolve to the pervasive computing gradually whose final goal is fusing the information space composed of computers with the physical space in which the people are working and living in. To achieve this goal, one of the problems is how to continuously monitor/capture and interpret the environment related information efficiently. Sensing context information and making it available to the people, involved in coordinating a collaborative task, is a preliminary phase in making a system adaptable to the prevailing situation in pervasive environments.

Many attentions have been paid to the research of the context-aware pervasive applications. However, the diversity of the sources of context information, the characteristics of pervasive environments, and the nature of collaborative tasks pose a stern challenge to the efficient management of context information by sensing a lot of redundant and conflicting information. Most of existing research just use the raw context directly or take just some aspects of the Quality of Context (QoC) into account. In this paper, we have proposed a middleware based context-aware framework that support QoC management in various layers. By this framework we can evaluate raw context, discard duplicate and inconsistent context so as to protect and provide QoS-enriched context information of users to context-aware applications and services. In future work, we will complete more experiments to discuss more aspects of the framework.

REFERENCES

[1] A. K. Dey, “Understanding and using context,” Personal and Ubiquitous Computing, vol. 5, no. 1, pp. 4–7, 2001.

[2] Kun Yang ,Alex Galis ,Hsiao-Hwa Chen, “QoS-Aware Service Selection Algorithms for Pervasive Service Composition in Mobile Wireless Environments, ” Mobile Netw Appl (2010) 15:488–501.

[3] Ardagna D, Pernici B (2007) Adaptive service composition in flexible processes. IEEE Trans Softw Eng 33(6):369–384

[4] Zhao Tang, Wenan Zhou, Jie Chang, Research on the Context-aware Service Selection Architecture,2010.

[5] Liu shulei, Liu yunxiang, Zhang fan. A Dynamic Web Services Selection Algorithm with QoS Global Optimal in Web Services Composition[J].Journal of Software, 2007, 18 (3): 646-656.

[6] Reiff-Marganiec, S., Truong, H.-L., Casella, G., Dorn, C., Dustdar, S., Moretzki, S.: The incontext pervasive collaboration services rchitecture. In: Mahonen, P., Pohl, K., Priol, T. (eds.) Proceedings of Service Wave 2008. LNCS, vol. 5377, pp. 134–146. Springer, Heidelberg (2008)

[7] Riva, O., Toivonen, S.: A hybrid model of context-aware service provisioning implemented on smart phones. In: Proceedings of ACS/IEEE International Conference on Pervasive Services (2006)

[8] T. Buchholz, A. K¨upper, and M. Schiffers, “Quality of context: What it is and why we need it,” in (HPOVUA 2003), Geneva, 2003.

[9] Y. Kim and K. Lee, “A quality measurement method of context information in ubiquitous environments,” in ICHIT’06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 576–581.

[10] P. N. MA Razzaque, S Dobson, “Categorization and modelling of quality in context information,” in Proceedings of the IJCAI 2005, 2005.

[11] D. Preuveneers and Y. Berbers, “Quality Extensions and Uncertainty Handling for Context Ontologies,” in Proceedings of (C&O 2006), P. Shvaiko, J. Euzenat, A. L´eger, D. L. McGuinness, and H. Wache, Eds., Riva del Garda, Italy, August 2006, pp. 62–64. [Online]. Available: http://www.cs.kuleuven.be/ davy/publications/cando06.pdf.

[12] K. Sheikh, M. Wegdam, and M. van Sinderen, “Quality-of-context and its use for protecting privacy in context aware systems,” JSW, vol. 3, no. 3, pp. 83–93, 2008.

[13] A. Manzoor, H. L. Truong, and S. Dustdar, “On the evaluation of quality of context,” in Smart Sensing and Context, Third European Conference, EuroSSC 2008, Zurich, Switzerland, October 29-31, 2008. Proceedings, 2008, pp. 140–153.

[14] Di Zheng，Jun Wang，Yan Jia，Wei-hong Han，Peng Zou. Deployment of Context-aware Component-based Applications based on Middleware. In UIC 2007，July，2007，Hongkong.

[15] Di Zheng，Jun Wang，Yan Jia，Wei-hong Han，Peng Zou. Middleware based Autonomic Management of the Context-aware Component-based Applications. In ATC 2007，July， 2007，Hongkong，

[16] Di Zheng, Yan Jia, Peng Zhou, Wei-Hong Han. Context-Aware Middleware Support for Component based Applications in Pervasive Computing. In： Proceedings of The 7th International Conference on Advanced Parallel Processing Technologies，November， 2007.

[17] Kun Yang , Alex Galis , Hsiao-Hwa Chen. QoS-Aware Service Selection Algorithms for Pervasive Service Composition in Mobile Wireless Environments,2009.

Di Zheng is a Lecturer of Department of Computer Science at Naval University of Engineering, China. She is a pervasive computing researcher whose interests include context modeling and management, development of the context-aware middleware and evaluating the quality of the context-aware applications. She received her Ph.D. in the area of context-aware pervasive computing from NUDT in 2008.

Jun Wang is a Lecturer of Department of Key Research Lab at Wuhan Radar Institute, China. He is a Middleware researcher whose interests include development of the middleware based applications. He received his Ph.D. in the area of context-aware pervasive computing from NUDT in 2007.