ProMWS: Proactive Mobile Web Service Provision Using Context-Awareness

Chii Chang, Sea Ling, Shonali Krishnaswamy
Faculty of Information Technology
Monash University
Melbourne, Australia
{chii.chang; chris.ling; shonali.krishnaswamy}@monash.edu

Abstract—**Applying Web service in mobile peer-to-peer service provisioning enhances the interoperability and resolves the heterogeneous challenges of ubiquitous environments. However, due to the nature of mobile peer-to-peer environments, a central repository for assisting service discovery is nonexistent, service discovery process relies on routing techniques, which increase the latency of the service interaction. This paper introduces ProMWS, a context-aware mobile Web service pre-fetching strategy to enable proactive service provisioning based on user preference and context. The proposed strategy predicts the mobile user's query, and cooperates with remote services to pre-fetch the service to the cache of mobile device in advance before the user requests for it.**

Keywords-mobile peer-to-peer service provisioning; Web service; cache pre-fetching; context-awareness; proactive service provisioning; user centric service; personalised service; prediction;

I. Introduction

Recently, mobile devices such as personal digital assistants (PDAs), smart phones, and handheld media players are powerful enough to be employed as service providers and participate within the ubiquitous computing environment. These devices are capable of establishing an infrastructure-less, peer-to-peer topology, and providing services to each other. Such a topology is known as mobile peer-to-peer network (MP2P), which plays an essential role in ubiquitous computing.

In a MP2P environment, devices participate in the decentralized wireless network as peers. Each peer is free to join and leave the network. Centralized management entities such as service repository, registry, and brokers are nonexistent. Peers are connected as cluster using peer-to-peer communication protocols such as Bluetooth [1] or WiFi Direct[2]. Service discovery in decentralized MP2P is usually based on routing techniques, or utilise Distributed Hash Table (DHT) [1].

Interoperability is one of the challenges in MP2P environment. In a highly flexible ubiquitous computing environment, there can be heterogeneous devices operating on different platforms and with different technologies. Enabling interaction between these devices can be a crucial task. Numerous researchers [2, 3] have proposed Web service standard-based solutions to realize loosely coupled interaction in such an environment. Web services describe their functionalities using XML-formatted Web Service Description Language (WSDL) documents, and communicate with XML-formatted Simple Object Access Protocol messages (SOAP) or JavaScript Object Notation messages (JSON)[3]. To invoke a Web service, retrieving and parsing the XML-formatted documents is a required task. Since centralized repository is not available in decentralized MP2P, and XML-formatted documents are quite heavy weight in an infrastructure-less environment, especially when the environment consists of a large number of service invocations, the latency can also be high during the service discovery process. Although binary-format-based compression can highly reduce the document size, such a solution requires all participants to support the mechanism of de-compression. In an open environment, it is difficult to guarantee that every peer can support such a mechanism. An alternative solution is caching the previous interacting services to improve service discovery performance [4]. However, such a solution is unable to benefit a newly joined peer, which has no prior knowledge about the environment.

The benefit of having a cache to improve the performance of MP2P service provisioning is explicit. Caching techniques have been used in various domains including traditional networked systems [5], Internet-based Web services [6], and also MP2P environments [7]. A popular strategy in the service-caching domain is cache pre-fetching or hoarding. Cache pre-fetching involves predicting a client's future request queries and pre-loads the results of a query before client sends the request. Such a strategy can highly reduce the latency and enhance the overall performance in networked service provisioning. Furthermore, a number of researchers [8, 9] have applied context-awareness to support the adaptive cache pre-fetching mechanisms in centralized network environments. Applying context-awareness in cache pre-fetching can improve the cache-hit of the pre-fetching, and avoid unnecessary data to be pre-fetched. However, in the decentralized MP2P environments, not much work has been done on context-aware cache pre-fetching

In this paper, we propose ProMWS — a context-aware cache pre-fetching strategy for **PRO**active **M**P2P **W**eb

[1] http://www.bluetooth.com
[2] http://www.wi-fi.org
[3] http://www.json.org/

Service provisioning. We show that our solution enhances the performance of MP2P Web service provisioning, and improves the user's experience during MP2P Web service interaction. It consists of the following three steps:

1. Predict what query may be sent by the newly joined user based on his/her current context information, the historical service interaction records, and the user preference profiles.
2. Cooperates with neighbouring peers to discover feasible service providers that can fulfil the predicted query.
3. Pre-fetches the results of the predicted query into the user's device in advance before the user actual sends the request to the mobile service hosted on his/her device.

The remainder of this paper is structured as follows: Section II describes the background and related works in the cache pre-fetching research area. Section III provides an overview of our pre-fetching model and components. Section IV explains the proposed context-aware user preference prediction scheme. Section V presents our experimental evaluation result. Finally, Section VI provides the concluding remarks of this work, and the future research direction.

II. Background and Related Works

Cache pre-fetching is one of the major strategies to reduce latency in networked service provisioning domain. The success of pre-fetching strategy is highly dependant on how the system predicts which service/resource should be pre-fetched.

A. Prediction

Prediction module aims to predict a user's near future request based on various factors. In Web browsing based systems, [5] have introduced a prediction module to determine what content needs to be pre-fetched based on the interaction histories. An extended approach proposed by [10] applied compound access graph to perform prediction based on the most recent browsing histories and the relationship between web pages. A mobile environment based pre-fetching proposed by [11] has considered the location factor. The prediction result was calculated based on the user's searching history in specific locations. These techniques were proposed for Web browsing systems and their prediction decision modules only considered static factors. They did not however consider the dynamic factors such as the user's preference in different situations and events.

A number of researchers [8, 11, 12] have proposed location-based and movement-based prediction scheme for cache pre-fetching. These works predict the probability of user's future query by analysing the user's present and future location (based on predicting his/her movement), the corresponding query history records, and the predefined user preference profiles. However, in reality, a user's preference can dynamically change at runtime due to different factors. Moreover, the pre-defined static user preference profiles and rules are difficult to fulfil unseen situations [15], unless the user is willing to adequately define many different preferences manually for all the possible situations. However, in most cases, a user is unable to define his or her probabilities for events accurately. [16]

B. User Preferences and Contexts

User preference profiling is an important aspect to improve the accuracy of the cache pre-fetching strategy. When the accuracy increases, the overall adaptivity is also improved due to the reduction of resource usage. However, existing works [11, 12] do not consider the dynamism of user's preference. As mentioned earlier, it is nearly impossible and inconvenient for most ordinary users to manually pre-define various preferences for all possible situations. The system needs to autonomously compute user's preference at runtime not only based on the historical query records, but also based on the user's current context. To overcome this challenge, our proposed context-aware cache pre-fetching strategy aims to dynamically predict user's preference at runtime using context-aware mechanisms.

Context is the environmental phenomenon that influences the system's behaviour, and supports the adaptive autonomous actions of the system. In this paper, we use Dey's definition [17] as the basis to describe context for our system, in which *"Context is any information that can be used to characterize the situation of an entity"*. [17].

III. ProMWS Model

In this section, we provide an overview on how the pre-fetching process is performed, and then the description of each main component of ProMWS.

A. Pre-fetching Process

In order to explain our proposed cache pre-fetching model, we firstly clarify each entity that is involved in the cache pre-fetching process.

- *Service* is an operation/method provided by a networked service provider. In order to invoke the service, one needs to retrieve the corresponding WSDL of the service.
- *Active Peer* (*AP*) is a newly joined mobile peer on the network.
- *Neighbouring Peer* (*NP*) is one of the peers already in the network. In order to join the network, an *AP* must discover and connect to at least one hub on the network. A hub can be a portable WiFi access device, or it can be a WiFi enabled portable computer. Hubs connect to each other to establish a cluster. Ideally, the *AP* should connect to a hub, which contains an *NP* who is already in the network for a while. Hence, the *NP* is capable of providing knowledge about the network such as what type services are available on the network, and how to reach these services. The *NP* can be a simple Web service provider that only provides its initial functionalities, or it can be a composite service provider which contains a number of *Service Caches*.
- *Service Cache* represents a local cache of a device that contains the WSDL and response data of a previously invoked service.

- *Preference Profile* (*PP*) is a runtime-generated document that describes what service types are needed by the mobile user. It is retrievable by requesting the preference operation from the peer.
- *Service Distributed Hash Table* (*ServiceDHT*) contains a list of available services on the network, and what operation types each service provides. Operation types are defined in a common ontology (we assume there is only one common service ontology document disseminated on the network). Each peer on the network should maintain its own ServiceDHT to ensure the content is up-to-date, so that later on, the peer can send the ServiceDHT to a newly joined *AP*.
- *Response Package* (*RP*) contains a set of services (S). Each s∈S encompasses service description (WSDL) and a set of data corresponding to the result of particular service.

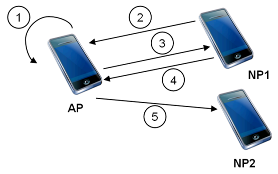

Figure 1. Cache pre-fetching process

Figure 1 illustrates a simplified cache pre-fetching process model. Each step is described as follows:

1. *AP* joins the network.
2. *NP1* posts *ServiceDHT* to *AP*.
3. *AP* generates *PP*, and then sends *PP* to *NP1* if *AP* reckoned that *NP1* is capable of responding some of *AP*'s preferred operations. (This information is available from *ServiceDHT*).
4. *NP1* checks its cache, if an operation of a service cache matches one of the preferred operation defined in *PP*, *NP1* adds the service cache and the corresponding data (if the data is still valid) to *RP*. After all the cached items have been checked, *NP1* sends the *RP* back to *AP*.
5. If the result data of the preferred operations have been fully pre-fetched, the process is completed. Otherwise, *AP* will searches for networked services and invoke feasible services to continue pre-fetching data for the rest of the user preferred operations.

Note that the pre-fetching process should always be performed in the background and must not interrupt or affect the device user's activities if he/she is using the device. (e.g., the pre-fetching process must not consume too many resources and slow down the entire system).

B. *ProMWS Architecture*

Figure 2 illustrates the architecture of MP2P Web service. It consists of a number of primary components to support dynamic service interaction and the cache pre-fetching mechanism. We describe each of the components below.

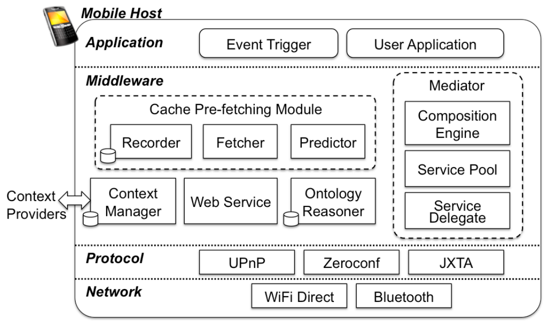

Figure 2. Mobile hosted ProMWS

User Application — it is a regular application with user interface support that enables the device user to interact with the middleware or perform settings.

Event Trigger — it cooperates with Context Manager to identify current situation. If current contexts match to one of the defined rule, Event Trigger will submit a request to Mediator to perform proactive action (e.g., retrieve all special lunch menus from each cafe in the current area).

Web Service — each peer on the network should apply Web service as the common communication interface.

Context Manager — It continuously operates individually to retrieve up-to-date raw context data from context providers, and interprets the collected raw context data to compound contexts based on the pre-defined matching rules. For example, a rule may define a compound context - noise level is loud when the value of environmental context raw data - noise is between 30 and 50.

Context Provider — A context provider can be an external sensor device, or it can be an embedded application within the same mobile device. A context provider can be a compass application, a map application (e.g., Google map), a sound detection application, etc.

Ontology Reasoner — Networked services are described using WSDL and a common ontology file. The ontology reasoner searches for the keyword described in the ontology for a particular service type. The search result will be used to define which networked service is capable of providing the need of a user's request. The details of service reasoning can be found in [18].

Mediator — It provides dynamic service invocation by applying Web service late-binding technique. Mediator encompasses three sub-components:

- *Composition Engine* — applies mechanism to trigger dynamic service invocation for multiple operations, and generates a composite result for a user' s request. It cooperates with Ontology Reasoner to identify a feasible URI (Uniform Resource Identifier) for each operation.
- *Service Pool* — supports networked service browsing (*NSB*) and service registry (*SR*). NSB mechanism lets Service Pool hold a list of the names of currently available networked services. *SR* mechanism deploys a Web service that allows external networked service providers to actively advertise their WSDL to the embedding mobile host.

- *Service Delegate* — is generated at runtime based on a corresponding WSDL and ontology. Its basic function is the same as a Web service client. However, it cooperates with the service caching mechanism to act like a linked Web service end-point. When it receives a request query from other local components, it first checks whether the local cache has the corresponding result data. If such data exists and is valid, the query will be responded locally without any networked transaction.

Cache Pre-fetching Module — encompasses three major components for enabling the cache pre-fetching mechanism. They are described as follows:

- *Recorder* — Each time the device user sends a request query to the Mediator from the User Application, the recorder will record the details of the request, and a set of current context.
- *Fetcher* — It manages pre-fetched data item. Each data item is stored in a particular local directory in the device storage, and corresponding information can be retrieved from the Fetcher
- *Predictor* — It uses the prediction technique (to be described in the next section) to predict the mobile user's future query.

IV. Context-Aware Prediction Scheme

The main technique that ensures the success of cache pre-fetching in our system is the context-aware prediction scheme. The context-aware prediction scheme takes user's current contexts as the basis, and then compares the current contexts to historical records to compute which query requested by the user has the highest. In this section, we describe our proposed context-aware prediction scheme. Firstly, we explain each element involved in the scheme.

Raw Context Data (*rc*) — is the data retrieved from context providers such as Global Positioning System, Compass application, image sensor, video sensor, voice senor, and so on. An *rc* will be used as the basic input parameter to describe an interpreted context.

Context (*c*) — is an output from a rule-based context interpreting process. Based on [19], an interpreting rule consists of context type (type), the scope of raw context data value, which includes minimum value and maximum value, and the output represents the interpreted value from this definition. For example, an interpreting rule describes `inputMin="x12y14", inputMax="x37y22", type = "location", output="meeting_room"`. When a retrieved location *rc* contains a value: x15y17, which is within the scope of inputMin, and inputMax, the system will consider a *c*: `location="meeting_room"` as one of the current contexts.

Candidate Query (*query*$_x$) — represents the query, which is sent by the user based on current contexts. When the Predictor component receives a set of *c*, it can predict the user's near future query based on the comparison result between the current contexts and the contexts of each query record. User may also define a preferred query manually by setting a set of *c* and the corresponding query in a file, which will be loaded in the beginning of the process. If user's definition exists, it will be used as the priority option. Otherwise, the system will perform the prediction automatically.

Query Records (*R*) — Each device should maintain a set of query records (*R*) representing the device user's previous queries associated with contexts. Each *r* in *R* consists of a *query* and a *c* set (*C*). Each *c* encompasses a context attribute and a context value.

Context Importance (*e*) — is a user-defined value in the *context importance rules* (*CIR*) for clarifying the importance of a context type to a query. By default setting, each context type has equal importance (set to 0) to all the queries. For example, a user may consider the location context to be more important to a query for searching the train arrival time. Hence, the user can increase the importance of the location context (e.g., set it to a number greater than zero) to the query to improve the prediction accuracy. Such a setting can also be applied globally. For example, user may prefer the location context should always be the primary consideration. Hence, whenever the prediction is performed, the location context will always be allocated a higher importance value than the other contexts.

A. Predicting The Weight of Query

The prediction scheme applies *Bayes' theorem* [20] and the *context weight* model [19] to compute the weight of *query*$_x$ found in *R*. The three main elements of the prediction scheme include the probability of a query, the weight (influence) of a context to a query, and the user's ranking for a context to a query. The following illustrates the formula to compute the weight of *query*$_x$ based on current contexts (*C*) and query records (*R*).

$$w(query_x|C,R) = \sum_{i=1}^{|C|}\left(\frac{P(c_i|query_x)\cdot P(query_x)}{P(c_i)}\cdot\frac{1+e(c_i,query_x)}{|C|+\sum e}\right) \quad (1)$$

where $w(query_x|C,R)$ denotes the weight of *query*$_x$ computed from summing the influence value of each current context. The influence value of a current context (c_i) to *query*$_x$ is computed based on matching the current context c_i to the contexts in each $r_i \in R$, plus the *context importance* (*e*). We describe each element used in (1).

$P(c_i|query_x)$ denotes the probability of *query*$_x$ requested by the user when c_i occurs. This is computed by (2).

$$P(c_i|query_x) = \frac{\left|\{r \in R : query_r = query_x \wedge \exists c \in C_r, c = c_i\}\right|}{\left|\{r \in R : query_r = query_x\}\right|} \quad (2)$$

$P(query_x)$ denotes the probability of *query*$_x$ based on its occurrence in *R*, which is computed by (3).

$$P(query_x) = \frac{\left|\{r \in R : query_r = query_x\}\right|}{|R|} \quad (3)$$

$P(c_i)$ is the probability of a randomly selected query that contains c_i as one of its attributes, which is computed by (4).

$$P(c_i) = P(c_i | query_x) \cdot P(query_x) + P(c_i | query_x{}') \cdot P(query_x{}') \quad (4)$$

$P(query_x{}')$ denotes the probability of other queries in R excluding the consideration of records that have $query_r = query_x$. It is computed by (5).

$$P(query_x{}') = \frac{|\{r \in R : query_r \neq query_x\}|}{|R|} \quad (5)$$

$P(c_i|query_x{}')$ denotes the probability of other queries requested by the user excluding $query_x$ when c_i occurrs, This is computed by (6).

$$P(c_i | query_x{}') = \frac{|\{r \in R : query_r \neq query_x \wedge \exists c \in C_r, c = c_i\}|}{|\{r \in R : query_r \neq query_x\}|} \quad (6)$$

In order to retrieve the rate of a single c_i to $query_x$, we compute the value using (7):

$$rate(c_i | query_x) = \frac{P(c_i | query_x) \cdot P(query_x)}{P(c_i)} \cdot \frac{1 + e(c_i, query_x)}{|C| + \sum e} \quad (7)$$

where $\sum e$ is the sum of a set of e in the context importance rules, in which the defined value of e is not 0. $e(c_i,query_x)$ denotes one of the defined rule, where $e(c_i, query_x) \leftarrow e \in E, c_e = c_i \wedge query_e = query_x$.

V. Experimental Evaluation

The experimental evaluation consists of two parts. Firstly, we determined the accuracy of our proposed context-aware prediction scheme. Secondly, we evaluated the improvement in performance for scenarios when the context-aware prediction scheme was applied.

We have implemented the prototype on an iPhone 3GS and a Macbook. The Active Peer described in Section III.A was implemented on the iPhone 3GS. 30 MP2P Web services have been implemented and deployed on the Macbook. We used Zeroconf[4] as MP2P service discovery protocol under an IEEE 802.11g WiFi network.

A. Prediction Scheme Evaluation

We have evaluated the prediction scheme by using 100 pre-generated query records. Each query record consists of a query and a set of contexts occurred when the user sent the query (see section IV). In the setting, we made the *time context* to have higher influence to the user-selected query, so we can show the differences between the default prediction and the prediction that applied *CIRs* (see section IV). A sample query record is shown as below:

```
recordID001 = { query = {
      type = "trainTime",
      input = { from = "F. station",
                to = "C. station"}}
   contexts = { location = "F. station",
time = "12:30", weather = "raining"}}
```

We used a subset of records as training set to predict the rest of the records. Figure 3 illustrates the accuracy of prediction when 60, 70, 80, or 90 out of 100 query records (x-axis) were used as training set to predict the rest of records. Furthermore, it shows the improvement of the accuracy when the user-defined *CIRs* (*Rule-applied*) were applied comparing to the predictions without *CIRs* (*Default*).

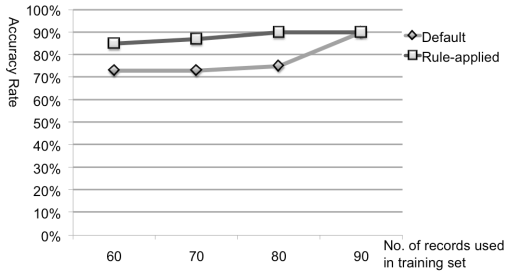

Figure 3. Precision influenced by training set

The result shows that the accuracy of prediction in *Default* was between 73% to 90% depending on the number of records used as training set, and the accuracy of prediction in *Rule-applied* was between 85% to 90%.

B. Performance Evaluation

For performance evaluation, we implemented three different scenarios. The first two scenarios applied the *context-aware prediction scheme* to dynamic generate and provide user preference from newly joined peer (*AP*) to neighbouring peers (*NP*). Based on this knowledge, *NP*s can assist the pre-fetching process with *AP* to reduce the overall latency. The third scenario illustrates a situation when an active *NP* for assisting pre-fetching is not available in the environment. Hence, the *AP* relied on routing-based technique to perform the generic late-binding-based Web service invocations for pre-fetching. For each scenario, we recorded the time taken for the pre-fetching process in which a different number of services were matched to the required service type, and have been invoked to retrieve the data for pre-fetching. Following is the description of each scenario.

Scenario 1: An active neighbouring peer (*NP*) notices the newly joined peer A (*AP*). Hence, *NP* invokes *AP* to retrieve its preference. After *NP* matches the preferred service of *AP* to *NP*'s service caches, *NP* finds its service caches can fully satisfy *AP*'s preference. Hence, *NP* sends its service caches to *AP* for pre-fetching.

Scenario 2: *NP* notices *AP*'s join. Hence, *NP* invokes *AP* to retrieve its preference. After *NP* matches the preferred service of *AP* to *NP*'s service caches, *NP* found the data of its service caches cannot fulfil *AP*'s preference. However, the WSDL files of *NP*'s service caches match the preferred service type. Hence, *NP* sends the WSDL files to *AP*. Based on the WSDL files, *AP* can invoke those services to retrieve data for pre-fetching.

Scenario 3: *AP* joins the network. *NP* only sends *ServiceDHT* to *AP* without further assistance. Based on *ServiceDHT*, *AP* retrieve the WSDL file of each networked

[4] http://www.zeroconf.org

service that matches the preferred service type, then invoke each service to retrieve data for pre-fetching.

Figure 4 illustrates the evaluation result that shows the time taken influenced by the number of matched service that were required to be invoked for each scenario.

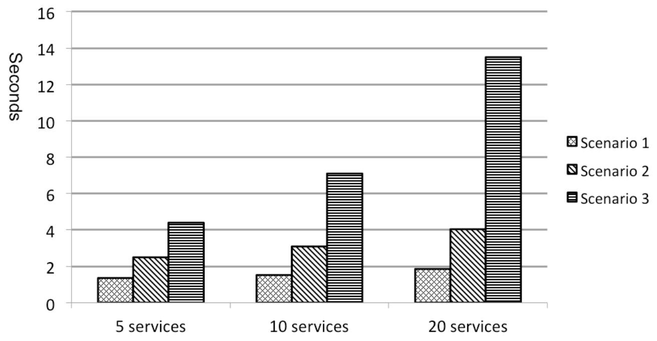

Figure 4. Time spent in seconds influenced by number of matched services for each scenario

Comparing the results from each scenario, we have seen that the process for retrieving and parsing WSDL from each service provider used in scenario 3 has increased the overall latency. The performance of scenario 1 and 2 were benefited by the cooperating caching mechanism along with our prediction scheme, in which the WSDL files of the required services were provided by an active neighbouring peer. Hence, the latency in scenario 1 and 2 was much lower. However, in an environment where an active neighbouring peer is not available, scenario 3 is the only choice.

VI. Conclusion and Future Work

In this paper, we have described our proposed framework — ProMWS to enable proactive mobile Web service provision using context-awareness. The main strategy — *cache pre-fetching* predicts user's near future query based on current contexts. Based on the predicted result, ProMWS dynamically generate user preference and deploy it as a service for neighbouring peers to assist the ProMWS peer in performing cache pre-fetching before the ProMWS user sends the request. This strategy saves the time spent by a MWS user to search and interact with networked services when he/she enters the network coverage area. Distinct from existing works that only used static factors [5, 10, 11, 12] for prediction, our prediction scheme is dynamic based on context-awareness at runtime.

For future work, we intend to address resource-aware service interaction, in which when multiple neighbouring peers intend to assist the newly joined peer, the newly joined peer should distributes its tasks to one or multiple remote peers based on service and resource availabilities.

References

[1] G. S. Manku, "Routing networks for distributed hash tables," in Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing (PODC '03), pp. 133-142, Jul. 2003.

[2] H. Schmidt, A. Köhrer and F. J. Hauck, “SoapME: a lightweight Java ME web service container,” in Proceedings of the 3rd Workshop on Middleware For Service Oriented Computing, pp. 13-18, Dec. 2008.

[3] S. N. Srirama, M. Jarke, H. Zhu and W. Prinz, “Scalable Mobile Web Service Discovery in Peer to Peer Networks,” in Proceedings of the Third International Conference on Internet and Web Applications and Services, pp. 668-674, June 2008.

[4] C. Doulkeridis, V. Zafeiris, K. Nørvåg, M. Vazirgiannis and E. A. Giakoumakis, “Context-based caching and routing for P2P web service discovery,” Distributed and Parallel Databases, vol. 21, pp. 59-84, Feb. 2007.

[5] Z. Jiang, L. Kleinrock, "An adaptive network pre-fetch scheme," in IEEE Journal on Selected Areas in Communications, vol. 16, no. 3, pp. 358-368, Apr. 1998.

[6] D.B. Terry and V. Ramasubramanian, "Caching XML Web Services for Mobility," Queue vol. 1, no. 3, pp. 70-78, May 2003.

[7] H. Artail, H. Safa, K. Mershad, Z. Abou-Atme and N. Sulieman, "COACS: A Cooperative and Adaptive Caching System for MANETs," in IEEE Transactions on Mobile Computing, vol.7, no.8, pp.961-977, Aug. 2008.

[8] S. Drakatos, N. Pissinou, K. Makki and C. Douligeris, "A future location-aware replacement policy for the cache management at the mobile terminal," in Wireless Communications & Mobile Computing, John Wiley and Sons Ltd., Chichester, UK, vol. 9, no. 5, pp. 607-629, 2009.

[9] V. Charvillat and R. Grigoraş, "Reinforcement learning for dynamic multimedia adaptation," in Journal of Network and Computer Applications, pp. 1034-1058, Aug. 2007.

[10] N. J. Tuah, M. Kumar and S. Venkatesh, "Resource-aware speculative pre-fetching in wireless networks," in Wireless Networks, Kluwer Academic Publishers Hingham, MA, USA, vol. 9, no. 1, pp. 61-72. Jan. 2003.

[11] S. Bürklen, P. J. Marrón, K. Rothermel and T. Pfahl, "Hoarding location-based data using clustering," in Proceedings of the 4th ACM international Workshop on Mobility Management and Wireless Access, ACM, New York, NY, pp. 164-171, Oct. 2006.

[12] I. Choi, H. Lee and G. Cho, "Enhancing of the pre-fetching Prediction for Context-Aware Mobile Information Services," in Mobile Ad-hoc and Sensor Networks. pp. 1081-1087, 2005.

[13] B. Jin, S. Tian, C. Lin, X. Ren and Y. Huang, "An Integrated pre-fetching and Caching Scheme for Mobile Web Caching System," in the 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, vol.2, pp.522-527, Jul.-Aug. 2007.

[14] C. Boldrini, M. Conti, F. Delmastro, A. Passarella, Context- and social-aware middleware for opportunistic networks, Journal of Network and Computer Applications, Middleware Trends for Network Applications, vol. 33, no. 5, pp. 525-541, Sep. 2010.

[15] A. Chen, "Context-aware collaborative filtering system: predicting the user's preferences in ubiquitous computing," in CHI '05 Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, pp. 1110-1111, Apr. 2005.

[16] D. Heckerman, "Bayesian networks for knowledge discovery," in Advances in Knowledge Discovery and Data Mining, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. American Association for Artificial Intelligence, Menlo Park, CA, pp. 273-305, 1996.

[17] A. Dey, "Understanding and Using Context," in Personal Ubiquitous Computing, vol. 5, no. 1, pp. 4-7, Jan. 2001.

[18] L. Steller, S. Krishnaswamy, M. M. Gaber, "Enabling Scalable Semantic Reasoning for Mobile Services," in International Journal on Semantic Web and Information Systems, IGI Global, 2009.

[19] P. Delir Haghighi, S. Krishnaswamy, A. Zaslavsky and M. M. Gaber, “Reasoning about Context in Uncertain Pervasive Computing Environments,” in Proceedings of the 3rd European Conference on Smart Sensing and Context, pp. 112-125, 2008

[20] J. K. Clema and M. Fynewever, M. "The dynamic re-evaluation of alternatives and the emulation of human decision making," in Proceedings of the ACM Annual Conference, vol. 2, pp. 852-859, 1972.