Context-based Caching and Routing for P2P Web Service Discovery

Christos Doulkeridis[1], Vassilis Zafeiris[1], Kjetil Nørvåg[2], Michalis Vazirgiannis[1]
Emmanouel Giakoumakis[1]
[1]Dept. of Informatics, AUEB, Athens, Greece
{cdoulk,bzafiris,mvazirg,mgia}@aueb.gr
[2]Dept. of Computer Science, NTNU, Trondheim, Norway
Kjetil.Norvag@idi.ntnu.no

Abstract

In modern heterogeneous environments, such as mobile, pervasive and ad-hoc networks, architectures based on web services offer an attractive solution for effective communication and inter-operation. In such dynamic and rapidly evolving environments, efficient web service discovery is an important task. Usually this task is based on the input/output parameters or other functional attributes, however this does not guarantee the validity or successful utilization of retrieved web services. Instead, non-functional attributes, such as device power features, computational resources and connectivity status, that characterize the context of both service providers and consumers play an important role to the quality and usability of discovery results. In this paper we introduce context-awareness in web service discovery, enabling the provision of the most appropriate services at the right location and time. We focus on context-based caching and routing for improving web service discovery in a mobile peer-to-peer environment. We conducted a thorough experimental study, using our prototype implementation based on the JXTA framework, while simulations are employed for testing the scalability of the approach. We illustrate the advantages that this approach offers, both by evaluating the context-based cache performance and by comparing the efficiency of location-based routing to broadcast-based approaches.

1 Introduction

An increasing amount of data and services are becoming *contextual*, in the sense that context is inherent in the relevant metadata information. In addition, services also become *context-aware*, so that results are returned based on context. A simple specific example is location-dependent services, which deliver results based on location of the requestor. However, location is just one dimension of context. A further extension naturally includes time (spatio-temporal context), but *context can also be generalized to an arbitrary number of dimensions*.The contextual dimensions can for example be the result of sensor information like temperature or speed, constraints that depend on availability of resource like bandwidth or display characteristics, but also human factors like personal habits and social environment.

An example of a simple context-aware query is *"find images of local monuments"*, submitted by a user currently walking near Acropolis in Athens using a device that can display images of size at most 640x480 pixels. Apparently the user searches for low quality images of the monuments on/around Acropolis hill. In this case, images of monuments far away from the query location or of quality higher than the capabilities of the requestor device are irrelevant for the query. Thus, both the location of the requestor and the device capabilities have a significant impact on the query processing plan. Generalizing the above, such implicit - contextual - information related to the query or the device needs to be taken into account to increase effectiveness of query processing and quality of the results.

It is thus natural that future mobile web services will be characterized by an arbitrary number of contextual dimensions. In this paper, the term service refers to web services as defined by the W3C[1]. An

[1]W3C, Web Services Architecture. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-

important challenge is how to find appropriate web services based on context, i.e., *context-aware web service discovery*. Web service discovery from handheld devices differs from traditional web service discovery. The dynamic features of terminals - such as heterogeneity, availability, mobility - that act as service providers and share data, resources and functionality, make existing discovery mechanisms designed for stationary web services problematic. Mobile devices, despite their fast technological advances, still impose restrictions with respect to processing power, storage space, energy consumption and bandwidth availability (Jensen, 2002).

Modern mobile devices are able to produce a wide range of data, from multimedia files (images or video) to environmental measurements (for devices equipped with sensors). The heterogeneity of this information calls for a globally acceptable way for access and sharing, hence the use of web services is adopted. In our application scenarios, each mobile device may host a set of web services to enable access to different types of information. A wide variety of applications can benefit from such mobile web services, among others: peer-to-peer content sharing, provision of dynamic or real-time navigation information, smart homes, detection of emergencies.

The fundamental mechanism for web service discovery is querying service directories, i.e., registries of service descriptions that facilitate web service discovery based on specific parameters. A context-aware service directory (Doulkeridis et al., 2003; Doulkeridis and Vazirgiannis, 2004) increases the precision and efficiency of web service discovery by matching the context of the user submitting a request against the service's contextual information. Moreover, lack of scalability and the *single point of failure* problem require distribution of service directories. One suitable solution for solving this problem is to use a peer-to-peer (P2P) architecture. In our approach, the service directories are maintained by servers, each responsible for a particular geographic area. Each server maintains a context-aware service directory, storing information about all published web services in the corresponding area. Mobile devices perform web service discovery querying the server responsible for their area. The servers of different areas are interconnected in a P2P network, essentially functioning as super-peers.

The challenge we tackle is efficient discovery of web services. In this paper we present algorithms for context-based searching in the P2P network of service directories. The baseline search mechanism in unstructured P2P networks is broadcast-based, known as flooding. However this results in excessive communication cost and low granularity of search. It also imposes a query horizon to the service request, hence it is inappropriate for locating remote or rare contents. Furthermore contextual information is ignored, resulting in low quality web service discovery. In order to reduce the search cost and improve the search quality, we introduce context-based caching into the architecture. In this way, the results of context-aware service requests are cached, in order to save the processing costs of subsequent identical requests. These results (service descriptions) are of improved quality, due to context-awareness, and they can also be particularly useful when the query workload distribution is skewed. When the query horizon is not adequate for serving the service request, we propose context-based routing as a mechanism that aims to locate web services residing outside the query horizon.

The main contribution of this paper is the integration of contextual information in P2P web service discovery, providing an architecture and algorithms for efficient context-based caching and routing. Based on a prototype implementation we conduct experiments evaluating the performance of web service discovery based on parameters such as context (location), cache size, query load distributions, starting threshold for caching, etc. We also study a particular type of context-based routing, location-based routing and show how this ensures effective location-based service discovery. This paper is a revised and substantially extended version of an earlier conference paper (Doulkeridis et al., 2005), where we presented our approach regarding P2P service discovery based on caching and context-awareness. We now extend this work, by studying context-based routing of web service requests in a P2P architecture of service directories. This proves to be a promisingly scalable solution towards distributed and context-aware service discovery. In particular, the contents of Sections 4.1, 4.3, 5.2 are entirely new.

The remainder of the paper is structured as follows. In Section 2 we present related work. In Section 3 we present our context model for context-aware web service representation and we describe the MobiShare architecture for mobile web service discovery. In Section 4, we discuss query processing, particularly context-based caching and location-based routing, based on contextual queries for web services in a P2P

20040211/, 2004.

architecture of service directories. Section 5 presents the experimental results. In Section 6, we conclude the paper and describe future research directions.

2 Related Work

Distribution of service registries has emerged as a necessity in highly dynamic mobile environments. This also holds for static architectures. For example, the latest UDDI specification[2] addresses this challenge, however its perspective is orthogonal compared to our approach. UDDI focuses on avoiding key collision, when generating unique service keys in different registries.

Recently, several research initiatives have identified the need for enhanced service directories (Akkiraju et al., 2003; Jeckle and Zengler, 2002; ShaikhAli et al., 2003) or extended service descriptions. The WASP project (Pokraev et al., 2003) extends the functionality of UDDI by introducing UDDI+, an attempt to improve the existing service discovery mechanisms regarding semantic and contextual features. The CB-SeC framework (Mostefaoui and Hirsbrunner, 2003) attempts to enable more sophisticated discovery and composition of services, by combining agent-oriented and context-aware computing. Lee et Helal (Lee and Helal, 2003) also argue in favor of providing support for context-awareness by means of service registries and they propose the use of context attributes, as part of the service description. For a recent review on web service registries see (Dustdar and Treiber, 2005).

Service discovery in pervasive environments (Zhu et al., 2005) also calls for context-aware support, in order to make service discovery more efficient by intelligent selection of directories with relevant services. In (Chakraborty et al., 2006), an approach for distributed service discovery is presented based on P2P caching of service advertisements. Services are organized hierarchically and they are described using OWL to support semantic discovery.

In (Schmidt and Parashar, 2004), an approach for decentralized web service discovery based on P2P technology is presented. A global service discovery architecture (GloServ) is presented in (Arabshian and Schulzrinne, 2004), where services are organized based on location hierarchically, similar to DNS domain names. Our work is also related to research in the area of location-based services, which covers one of the possible dimensions, i.e., areas, in our contexts. In (Jensen et al., 2004) modeling of this dimension is studied. Another relevant research work is presented in (Lee et al., 2002), which is about location-dependent information services in pervasive computing environments. In (Steen and Ballintijn, 2002), services are organized hierarchically in a distributed search tree and the authors focus on avoiding bottleneck problems on high-level nodes.

In (Baggio et al., 2001) Baggio *et al.* describes the location service in Globe. In order to designate objects, the Globe location service uses unique identifiers which they call *object handle*. This handle is location independent. The location of an object is described by means of a *contact address*. The Globe location service is based on a hierarchical organization of geographical/administrative/network-topological domains. Each domain is represented by a directory node, responsible for keeping track of all objects in its domain. For each object, the node either has a *contact record* with the object address, or a *forwarding pointer* indicating that the information is stored lower in the tree.

There is an increasing interest in the research community about caching in P2P networks. One of the first approaches to P2P caching was presented in (Iyer et al., 2002). A set of nodes cooperate to function as a traditional Web cache. In (Patro and Hu, 2003), the locality of queries in a Gnutella P2P network is presented, and a transparent query caching scheme is proposed. Cache updates of mobile agents organized in a P2P manner is discussed in (Leontiadis et al., 2004), whereas an approach regarding distributed caching in Gnutella-like networks, based on distributing the query results among neighboring peers, is presented in (Wang et al., 2004). In (Hu et al., 2004), the authors propose building a P2P service directory by grouping together service entities semantically. In (Zheng et al., 2002), cache invalidation for location-dependent data is discussed and two new cache replacement policies are proposed. Recently taxonomy caching has been proposed as a scalable approach for web service discovery, which extends the query horizon in unstructured P2P networks (Nørvåg et al., 2006). Compared to the above, our approach deals with caching in P2P service directories, where contextual query workloads initiate caching of service descriptions, in order to augment the performance of service discovery.

[2] The UDDI specification version 3.0, `http://www.uddi.org`.

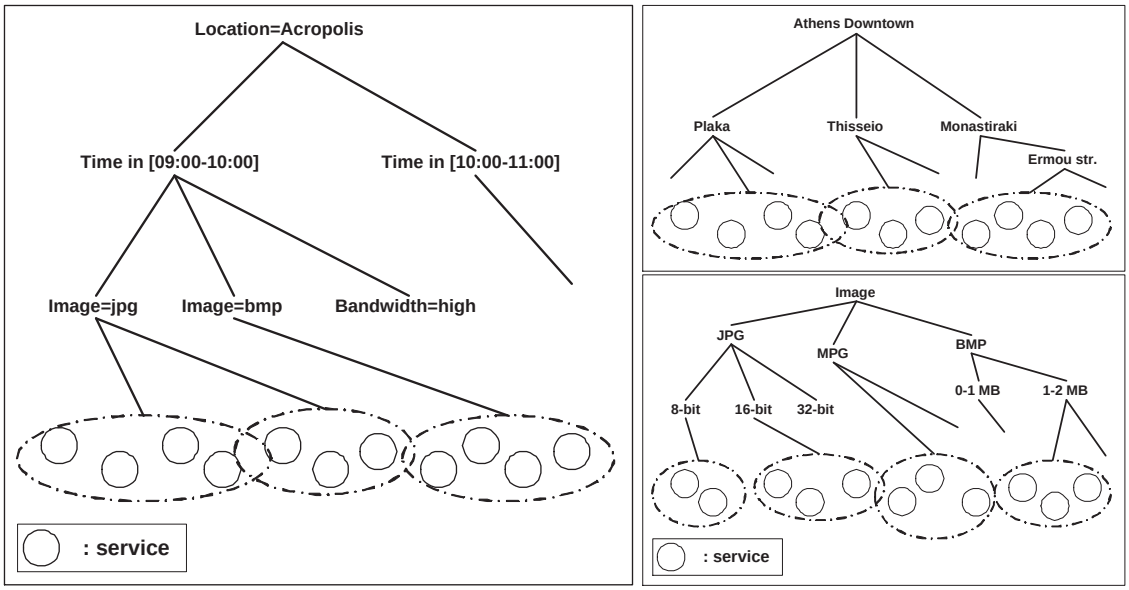


Figure 1: (Left) Conceptual model for context hierarchy. (Right) Different context hierarchies for each contextual dimension.

3 Data Model and Architecture

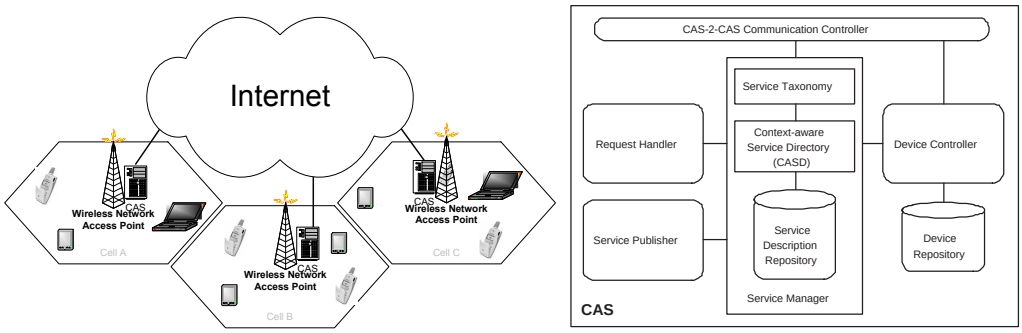
Our approach to web service discovery is based on the notion of a service directory and in particular we consider *context-aware service directories* (Doulkeridis et al., 2003) that support maintenance of contextual metadata for each web service. In our approach, this contextual information persists as part of the service directory and it is not maintained as a separate layer external to the web service architecture. This facilitates context-aware service requests aiming to discover more appropriate web services to the user needs. In this work, we stress the importance of taking into account context in forming and processing user queries. In (Doulkeridis et al., 2003), we have studied the benefits of context-awareness to the efficiency of processing and in the quality of the results. Any attempt to tackle mobile service discovery, without taking into account the involved contextual parameters, is bound to fail because it leads to: 1) small precision of the search, 2) extremely large volume of exchanged data, and 3) a time-consuming process that is annoying for the user (Doulkeridis and Vazirgiannis, 2004).

In the rest of this section we describe the data model for context representation of web services and we present the overall decentralized architecture of P2P service directories.

3.1 Context Model

The data model assumed in our work is a *context hierarchy*, as illustrated by an example in Figure 1 (left). The basic representation entity is the *contextual path*. In general, a contextual path is an ordered set of attributes (also called *dimensions*) that have been assigned values of a specific data type. The contextual path indexes one or more web services. The combination of all possible contextual paths within a service directory results in a context hierarchy. Note that in the case of a contextual dimension where this is not possible, we can always represent it as a degenerate hierarchy that consists of one level only. However, in general, the hierarchical representation model adopted here for contextual service descriptions follows the same principles as the most popular and widely used hierarchical model (XML), which is the cornerstone of all standardization efforts in the service-oriented world.

Mobile users issue keyword-based service requests that are semantically disambiguated by means of a service hierarchy (i.e. a set of service categories which is domain-specific and is organized in a hierarchical structure) (Valavanis et al., 2003), thus leading to those service categories relevant to the user request. At



(a) Mobishare infrastructure (b) CAS architecture

Figure 2: MobiShare and CAS architecture.

the same time, a contextual query is formulated, describing the user's current situation, in order to be issued to the context-aware service directory. In this paper, our focus is on contextual query processing, after having found the relevant service categories. The use of context-awareness in the service discovery process constitutes a filtering mechanism on the returned services, thus increasing the precision of retrieval. The issue of semantic service discovery will not be further addressed in this paper.

The choice of a hierarchical representation of context is straightforward for several types of context. In the case of location, geographical information supports the hierarchical notion in itself; countries consist of cities, each city has districts, and each district is further split into neighborhoods and streets. This is an example of disjoint contexts, i.e., the same district cannot belong to two different cities. In other cases, a context may belong to more than one contextual paths. For example, a service-specific contextual parameter, such as its average processing time may be preceded in the context hierarchy, by the bandwidth availability for accessing the service or by the hosting device characteristics. This results in contextual subpath replication in some parts of the context hierarchy, so that in order to allow easier management of contextual paths and support higher context granularity, the context hierarchy will be implemented using a separate hierarchy for each dimension, as illustrated in the right part of Figure 1. We will elaborate more on this later in Section 4.2.1.

Definition: Given a set of D contextual dimensions, we define a context hierarchy H such as: $H = \cup H_i$ with ($i = 1..|D|$), where H_i is an arbitrary hierarchical ordering of the values of each context domain.

Query and update algorithms for a hierarchical context model are usually similar to algorithms for tree-based representations. In previous work, we have studied both query (get a service based on its context) (Doulkeridis et al., 2003) and update (update the context of a service, insert new service) (Doulkeridis and Vazirgiannis, 2004) algorithms for a contextual model that resembles one hierarchy. In general, these algorithms are variations of breadth-first or depth-first search, according to the specific problem requirements.

The context hierarchy is used to support service requests, henceforth also referred to as query for services. The queries are enriched with a contextual query that captures the context of the request. The set of services S that satisfy a request is the intersection of the services S_t that match the topic of request, and the services S_c that match the context of the request: $S = S_t \cap S_c$.

While the context model is general enough to be applied in other domains, we focus on mobile environments in this paper. Even though we believe that the value of context is elevated by mobility, the scope of our work can be easily extended in any context-aware domain, as long as the necessary context hierarchies are defined for the particular domain.

3.2 Architecture

The work presented in this paper is done in the context of the MobiShare architecture (Valavanis et al., 2003). MobiShare provides an infrastructure for ubiquitous mobile access and mechanisms for publishing, discovering and accessing heterogeneous mobile resources in a large area, taking into account the context of both sources and requestors. As illustrated in Figure 2(a), MobiShare consists of numerous wireless

network access points that partition the geographical two-dimensional space into areas of coverage called *cells*. Mobile users carrying portable devices stroll around and share their data through web services. Besides mobile web services, there may also exist a variety of stationary web services in the surrounding environment. Within each cell, a *cell administration server* (CAS) is responsible for device monitoring and management, as well as for indexing the contents and services provided by the mobile devices. Data sharing web services are just one type of context-aware services. There are also web services that provide aggregated environmental information, like temperature, lighting conditions, amount of traffic, gathered from static sensors or mobile sensors attached to devices.

The CAS architecture is illustrated in Figure 2(b). Each CAS maintains a *context-aware service directory* (CASD) that stores information about all published services in the corresponding cell. Apart from CASD, a CAS consists of the following modules: 1) *CAS-2-CAS communication controller* containing addresses of other CASs and responsible for managing interactions with other CASs, 2) *Service Manager* containing the CASD, a *Service Taxonomy* and a *Service Description Repository*, 3) *Device Controller* that manages the device-specific information, 4) *Device Repository* storing a list of all devices in the cell along with their profiles, 5) *Request Handler* for receiving incoming service requests, and 6) *Service Publisher* for publishing service descriptions.

CASs are only aware of neighboring CASs, so global knowledge of the network topology is unavailable. In this sense, the CASDs form a super-peer network, with the actual peers being the mobile devices that offer mobile web services. This approach is robust and scalable, because it ensures system operation even during the failure of some nodes, and also allows dynamic addition or removal of CASDs. In this paper we are mainly interested in studying context-aware service discovery in this P2P network of CASDs, so henceforth they will also be called *peers* for simplicity.

3.3 Realizing Context Processing with SOAP and WSDL

The context processing features of the proposed architecture can be realized with base web services standards such as SOAP and WSDL. Compatibility with these standards is important as context has a key role in service-oriented architectures situated in a mobile and ubiquitous computing environment.

The capability of a node, either a CASD or mobile terminal, to process context information (and probably adapt its behavior upon it) is realized as a WSDL 2.0 feature component[3] identified by a URI, e.g. `http://web-service-standards.org/base-extensions/context-processing`. Features are a basic extension mechanism of WSDL 2.0 and represent abstract functionality that characterizes the message exchange specified by an operation or interface. Examples of such functionality are: reliable and secure transfer of messages, message routing, correlation etc.

The *context-processing* feature complements the message exchange during service discovery and publishing (and generally of any context-aware operation) with information related to the context of the interacting parties. It specifies the capability of the node implementing the "context-aware" operation to extract and take into account context information, but not the actual way that this information is used by the operation. The way that context enhances the behavior of the operation is part of the operation specification.

For purposes of illustrating the above concepts, a WSDL 2.0 code excerpt of the interface provided by CASDs' is provided below. CASD operations are grouped in two WSDL interfaces, a search and a publishing interface. The search interface comprises of operations for submitting service discovery requests to a service directory either from user terminals or peer directories. As regards the publishing interface it is used by user terminals for registering to service directories and managing their published services. The enhancement of one or more of the following operations with the *context processing* feature is expressed with a respective WSDL feature element.

```
<description
    xmlns="http://www.w3.org/2006/01/wsdl"
    targetNamespace="http://cs.aueb.gr/wsdl/casd"
    xmlns:tns="http://cs.aueb.gr/wsdl/casd"
    xmlns:wsoap="http://www.w3.org/2006/01/wsdl/soap">
```

[3]Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. `http://www.w3.org/TR/wsdl20`. World Wide Web Consortium Candidate Recommendation, 2006.

```
<types> ... </types>

<interface name="serviceDiscovery">
    <operation name="search"
              pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <!-- Context-aware search for services -->
        <feature ref="http://services-standards.org/base-
            extensions/context-processing" required="false"/>
        <input messageLabel="In" element="tns:searchRequest"/>
        <output messageLabel="Out" element="tns:searchResponse"/>
    </operation>
</interface>

<interface name="servicePublish">
    <operation name="register"
            pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <!-- Register a mobile device to a service directory -->
        <feature ref="http://services-standards.org/base-
            extensions/context-processing" required="false"/>
        <input messageLabel="In" element="tns:registerRequest"/>
        <output messageLabel="Out" element="tns:registerRequest"/>
    </operation>

    <operation name="publish"
            pattern="http://www.w3.org/2006/01/wsdl/in-out">
        <!-- Publish a set of services to the directory under a
           specific context -->
        <feature ref="http://services-standards.org/base-
            extensions/context-processing" required="false"/>
        <input messageLabel="In" element="tns:publishRequest"/>
        <output messageLabel="Out" element="tns:publishResponse"/>
    </operation>

    <operation name="unpublish"
            pattern="http://www.w3.org/2006/01/wsdl/in-out">
     <!-- Unpublishing does not require any context processing -->
        <input messageLabel="In" element="tns:unpublishRequest"/>
        <output messageLabel="Out" element="tns:unpublishResponse"/>
    </operation>
</interface>

<binding name="discoverySOAPBinding"
        interface="tns:serviceDiscovery"
        type="http://www.w3.org/2006/01/wsdl/soap"
        wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
   <operation ref="tns:search"
        wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response">
      <wsoap:module ref="http://soap-modules.org/context-processing/"
              required="false"/>
    ...
   </operation>
</binding>
...
</description>
```

The abstract functionality specified by the *context-processing* WSDL feature can be realized through a respective SOAP module[4]. A SOAP module extends the behavior of a single SOAP node by imple-

[4]SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3.org/TR/soap12-part1/. World Wide Web Con-

menting one or more features. A feature is expressed in SOAP messages with a set of SOAP header blocks. The definition of the syntax and semantics of these header blocks is part of the *context-processing* SOAP module specification. We identify this module with the URI `http://soap-modules.org/context-processing/` and reference it in appropriate operations elements in the interface bindings section of the WSDL description document[5].

The "required" attribute of the feature element specifies whether support of SOAP module is required for a client in order to utilize the service operations. A value of 'false' states that feature implementation is not mandatory for a client in order to invoke the respective operations, allowing thus both context-aware or "unaware" mobile terminals to interact with the CASDs. The same holds for the *context-processing* module that realizes the WSDL feature and is referenced in the definition of the service interface bindings. We have identified four types of SOAP header blocks that are relevant to context management in the proposed P2P service discovery architecture. Our focus in this section is on the specification of the types and semantics of SOAP header blocks, while the syntax and structure of block information will be part of our future work. Context header blocks are introduced in the header part of a SOAP message by CASDs or user clients. Each header block is identified by a local name and the namespace of the *context-processing* SOAP module:

- *context-match*, that requires from the ultimate receiver of the SOAP request to perform a match against the given context while generating a response. The header block contains an element information item that specifies a pattern for context matching. In the case of a service discovery request, the header block is introduced by a user client and is processed by the CASD receiving the request. The CASD evaluates the request on the subset of its contents that matches the given context pattern.

- *context-update*, that updates the context of the initial sender to the ultimate receiver of the SOAP message. Context information is included in the header block as an element information item. A context-update header block may be sent as part of a service publish request sent from a user terminal to a CASD, specifying the context of the published services.

- *context-trace*, that is used by an initial SOAP sender to collect context information during the routing of a SOAP request to the ultimate receiver through various intermediaries. For instance, a context-trace header block sent from a user terminal to a CASD, is initialized with the mobile terminal's current context and is further enriched with relevant context information items from all SOAP intermediaries relaying the request to the target CASD. Such SOAP intermediaries could be other user devices acting as SOAP gateways in the user's Personal Area Network. We assume that these intermediaries may have a more accurate or complete view of the initial sender's context and are authorized to access and append context information. A context-trace header block includes a set of element information items (context parts, that correspond to contextual paths or fragments of them) representing context information collected from the various intermediaries. A context-trace header can be sent as part of a device registration request to a CASD, allowing thus the user terminal to "explore" its surrounding context.

- *trace-result*, that includes the result of a context-trace header block. A trace-result header block is returned to the initial node as part of a SOAP response. On the basis of trace-result information a user device updates its perceived context that is further used during service publishing.

4 Context-based Caching and Routing

Based on the aforementioned super-peer architecture, we will study issues related to context-based searching, i.e. routing of service requests (also called queries for simplicity). We will first give a general overview of how searching is performed and then describe how context can be used to improve the quality of web service discovery. Our main approaches towards this aim are *context-based caching* and *location-based routing*.

sortium Candidate Recommendation, 2006.

[5]Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts. `http://www.w3.org/TR/wsdl20-adjuncts`. World Wide Web Consortium Candidate Recommendation, 2006.

4.1 Basic Searching

The basic search approach is broadcast-based, i.e., based on flooding. It involves query forwarding from the local CAS to its neighboring peers on every user request. Initially the query is submitted to the local CAS (henceforth also called *originator*), which performs the following tasks: 1) forwards the query to its immediate neighbors, and 2) processes the query locally. The neighboring CASs follow the same procedure as long as the maximum hop count TTL (time-to-live) is not reached. On each CAS where there is a match, the results are sent back directly to the originator, i.e., not through the neighboring peers. The originator collects all results, and sends them back to the requesting device.

The time-to-live parameter determines the range of the search (query horizon), in terms of the number of steps that the query will be forwarded to a peer's neighborhood, starting from the originator. It is clear that increasing values of TTL considerably increase the number of peers that receive and process the query, and thus the probability of reaching multiple and high quality answers. On the other hand, a very large number of messages has to be exchanged due to the exponential nature of this strategy.

The criteria for query satisfaction vary according to the specific application. While locating a single service that is close may be enough for a taxi-booking application, in the case of more complex services that may consist of several simple services this is not enough. For example, a request for cheap and near restaurants obviously cannot be limited to the first service found. Similarly, there exist scenarios of use that request the best N services, where N is a user-defined parameter. The search strategy employed clearly depends on the type of application. In this paper, we are interested in the retrieval of as many contextually relevant services as possible, assuming exact context matching.

Yet another direction for query satisfaction is the case of partial context matching. This means that a query can be satisfied even if only part of its context is matched with the service context. In this case a subset of the query dimensions are prominent or a more elaborate technique is to allow weighting of different dimensions. Usually the location dimension is considered prominent in most applications.

4.2 Query Processing with Caching

We enhance the basic search mechanism with context-based caching in order to address the case of recurring queries and increase the efficiency of P2P web service discovery. We start with a general overview, then describe messages that will be used in the communication, followed by a study of issues related to caching query results.

4.2.1 Overview

Local query evaluation is a hybrid approach that employs query forwarding and caching of results in order to enhance the overall performance of the P2P network. A CAS, henceforth also called peer (P_i), maintains data structures that store service descriptions, both local and cached ones, as well as relevant information needed to ensure cache consistency. Three data structures are identified as essential parts of a peer's internal architecture:

- A context-aware service directory (CASD) that is conceptually represented as a hierarchy of services. However, in the presence of many different contextual dimensions, it is not straightforward to combine them arbitrarily in a single way that makes sense, without replication of certain sub-hierarchies. Hence, context management becomes complicated. Instead, a slightly different approach is to assume different context hierarchies for different contextual dimensions (see Figure 1). This approach: 1) allows easier management of contextual paths and 2) supports higher context granularity. Nevertheless, both approaches present several similarities, due to their reliance on the notion of contextual paths, so the same algorithms and techniques apply in both cases. In the rest of this paper, we assume different hierarchies of contextual values for each dimension of context.

- A cache C used for services' descriptions storage. The structure of the cache is similar to CASD.

- A list L containing the identifiers of the peers P_j that currently maintain copies of P_i's services in their local cache.

4.2.2 Query and Cache Management Protocol

A detailed description of the message types introduced for query support and cache management is provided in this section. We highlight the role of each message type and specify basic attributes:

- Service Request (S_{req}): occurs when a peer (P_i) submits a contextual query Q_{cxt} for relevant services [6] to its neighbors $N(P_i)$. The message includes: Q_{cxt}, P_i's identifier, and a counter H (time-to-live) that indicates the number of steps that Q_{cxt} will be transmitted.

- Service Request Forwarding (S_{reqf}): a peer P_i that receives a S_{req} message sends S_{reqf} to neighbors in $N(P_i)$, if $H > 0$. In each message hop the value of H is reduced by one. The S_{reqf} message includes the peer identifier of the originator (P_{orig}), the query Q_{cxt} and the current hop count H.

- Service Response (S_{res}): sent by a peer P_i that received a Service Request S_{req}, back to the originator of the query. The contents of this message are the matching service descriptions retrieved both from the peer's CASD and its cache contents.

- User Moved (M_u): this message is send by a peer P_A to a peer P_B, whenever a user device u that offers services $u.S$ (registered in P_A's CASD) moves from the cell controlled by P_A to the respective cell of P_B. The two cells are assumed as spatially adjacent. The message contains the descriptions of the services $u.S$ and are appended in P_B's CASD.

- List L Moved (M_l): follows P_A's M_u message to P_B and transfers the list of peers kept in $P_A.L$ that keep cached content provided by u that migrated from P_A to P_B.

- Invalidate Cache (I_c): follows List L Moved (M_l) message and is sent by peer P_B (as defined above) to all the peers that cache services offered by u. The message contains the identifier of the peer that includes $u.S$ in its local CASD (P_B in our case) and the context of u in the new cell. The message triggers cache updates (i.e., changing peer id where service is now available - in this case changing P_A to P_B) and cache replacement in case the context of u is different in the respective cells of peers P_A and P_B.

- Invalidate List (I_l): is sent when a cache replacement occurs. When a peer P_A replaces an entry in its cache C, i.e., deletes service S originally from peer P_D with another service, it sends an I_l message to P_D causing deletion of the corresponding entry (i.e., P_A's identifier) from P_D's list L.

Contextual queries submitted to a peer P_A trigger retrieval of matching services and merging of parts of neighboring service directories ($N(P_A)$) to the originator cache, in order to enable local query processing of future service requests with similar context.

4.2.3 Message Exchange Example

Consider the case of a mobile device u located in a cell controlled by a peer A that issues a request Q for services. At the same time a contextual query Q_{cxt} is formulated incorporating the current context of the request (in terms of location, user profile, device capabilities, etc). Then the request is sent to the local service directory, say peer P_A and then forwarded to all neighboring directories ($N(P_A)$) using S_{reqf} messages. The matching services are returned to P_A via S_{res} messages and at a next step to the requesting device u. Finally, P_A merges the service descriptions with its current cache contents, for context Q_{cxt}, in order to serve further requests of the same type locally.

As a result of user movement, a mobile device can cross its current cell boundaries and thus enter another peer's area of responsibility. Figure 3 presents such a scenario, where a user device u moves from the cell controlled by peer A to the respective cell of peer B. In addition, it shows the message exchange sequence that is triggered by cell change. At first, a M_u message is sent from peer A to peer B which is now responsible for publishing the device's offered services $u.S$. This message is followed by M_l, that charges peer B with keeping consistent the caches of other peers (peers C and D) that store part of $u.S$. Peer B enforces cache consistency by sending I_c messages to C and D that update their cached copies of $u.S$.

[6] Here we remind the reader that the requestor's query Q is enriched with implicit contextual information forming thus Q_{cxt}

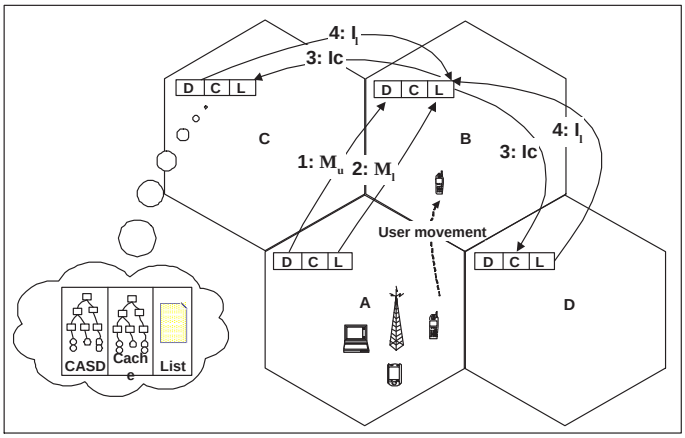


Figure 3: Message exchange resulting from user mobility in a P2P architecture of context-aware service directories.

Assume that the context of $u.S$ is different in the new cell in terms of the location dimension, e.g. in cell B [location=Thisseio] while in cell A [location=Acropolis]. Moreover, peers C and D are interested only in services offered in the area of Acropolis. As a result, cache replacement is triggered and I_l messages are sent from C and D to peer B in order to remove them from its list L.

4.2.4 Contextual Query Result Caching in Practice

In the following, we discuss issues related to caching query results and their effect on web service discovery in the architecture described above. Such issues are: query workload distributions, caching models and cache replacement strategies. In the next section, we experimentally validate these issues.

Non-uniformly distributed query workloads motivate the design of algorithms that achieve better performance. In the case of successive occurrences of a request q, caching in principle reduces the cost induced by processing repeated occurrences of q. Determining the point (in terms of query workloads) when caching query results should start may be useful. In other words, assuming a discrete set of possible queries and a statistical distribution of the expected query workload, a *threshold* must be defined, above which the originator starts caching parts of directories that satisfy a specific query. An initial choice could be the mean value of the query distribution. However, we will elaborate more on this later.

Maintaining *cache consistency* is a major issue in every caching scheme. The most popular approaches are: push-based and pull-based, as well as some variations. In push-based approaches, the cached content is updated by the CASDs that currently manage the services that have been cached. This is achieved by each peer keeping a list L of those peers that have at some point requested and cached the specific service. In pull-based approaches, peers that have cached content periodically ask the responsible peers for updates, in order to avoid stale content. Our approach is push-based and enforces cache consistency through cache updates whenever a change occurs, for example whenever a user moves to a different cell.

Allowing *cache forwarding* is another interesting issue of the caching approach. Consider the case of a peer B that contains cached content from another peer A, retrieved for the contextual query Q_{cxt}. If a third peer, say peer C, requests from B services for context Q_{cxt}, then two options exist: 1) B evaluates the query and sends the relevant part of its directory, which contains only descriptions of local services, to C, or 2) B evaluates the query and sends the part of the directory containing both local and cached service descriptions. The former option prevents caching of service descriptions beyond neighboring directories, in other words a peer can contain only its local contents and contents of its immediate neighbors, defined by H. In the latter case, we enable cache forwarding, so a peer may eventually (after many steps) contain service descriptions from any service directory. However, this will result in caching large portions of service directories and frequent cache replacements, with unexpected performance degradation.

Cache forwarding enables the diffusion of popular service descriptions even in peers that cannot reach them due to the limited TTL. As a result popular sources become available to distant peers while the

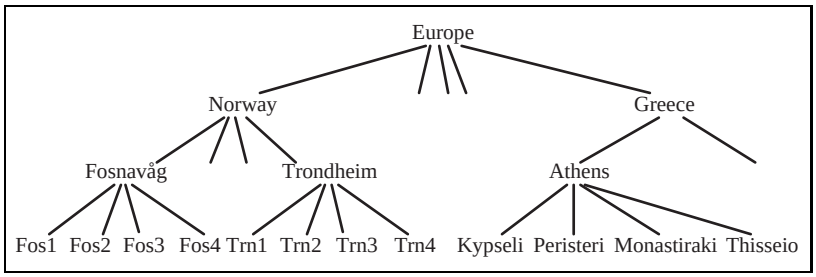


Figure 4: Location hierarchy.

message overhead due to frequent flooding with high query TTL is constrained. A negative effect of cache forwarding is the increasing cache consistency overhead in popular peers, e.g. peers located in an area of global interest (Acropolis in the case of Athens) or in any other area where a significant event takes place. With reference to the above example, peer A is responsible for the consistency of peer B and C caches, as regards context Q_{cxt}. In general, every peer that receives and caches forwarded cache content contacts the original source and registers for change notifications. Another issue that is raised from this approach is the latency in cache invalidation of distant peers, resulting in their use of stale results (even for a short time period). An amendment that can be introduced is the control of the horizon (hop distance) of cache forwarding by taking into account factors such as 1) change frequency of cached content, 2) available resources of source peer in terms of storage capacity or bandwidth.

Cache replacement strategy is a critical aspect of any caching scheme that greatly impacts the performance. In a P2P setting relatively large TTL values may return a high number of results thus causing frequent cache replacements. In this approach, we used least frequently used (LFU), which is deemed appropriate, since it penalizes services that are not requested frequently, thus replacing them with other more popular services.

The use of one hierarchy for each dimension looks like a case where a multidimensional index could be used. However, this is not the case, because unlike traditional multidimensional data where there is usually one value for each dimension, in our case for each service, there are just values for a small subset of possible dimensions. In this case, it is better having one hierarchy for each dimension (less space and less cost of updating). This approach is further motivated by the fact that service requests involve only a small subset of contextual dimensions, depending on the type of service. For example, a web service that aggregates and returns sensor readings, like temperature, heat or noise level, is dependent on the location, whereas a web service that provides file-sharing purposes depends on the file type and size dimensions of context. However it is highly unlikely that a web service may depend on most (or all) contextual dimensions.

4.3 Location Hierarchy

Intuitively, if we assume that the content in peers is location-specific, then upon receiving a query for location A, a node should route the query only to nodes residing in location A, rather than other locations. This motivates the need for location-based routing of service requests.

4.3.1 Limitations of Context-based Caching

While context-based caching can be particularly effective for recurring contextual service requests, as will be shown in Section 5, it does not suffice 1) in the case of inexact context match, and 2) for locating remote contents.

Consider the location hierarchy in Figure 4. Let us further assume location-based content assignment, meaning that the P2P topology is put on top of the geographical map, so that neighboring peers reside in nearby locations. If a contextual service request issued by a peer in *Trondheim* is for location *Monastiraki* and this leads to a local cache miss, then the alternative is to search using the basic mechanism, i.e., flooding, which would probably return few results. Actually in many cases, for sufficiently large networks

and queries for far locations, no results at all. However, using the hierarchy the service request could be extended to search for ancestor nodes of *Monastiraki* in the hierarchy, in this example *Athens*. If this entry is in the cache, then the query can be forwarded to the cached peer and then local flooding will find the requested location-specific content with high probability. Still, the query extension does not guarantee that the requested location will be found, since this is dependent on the contents of the cache or, in other words, on the query horizon. So the challenge is to exploit the location hierarchy (and generalizing, any context hierarchy) in a way that leads searches to remote peers, thus extending the query horizon.

Notice the advantages and differentiating factors of location-based routing compared to context-based caching. First, while context-based caching is beneficial in the case of recurring contextual queries, where the exact same context request is submitted, location-based routing can provide a direction for the search even in the case of partial context matching, when some (not all) of the query's contextual dimensions match the cached contextual description. In that case, the cache cannot be helpful. Second, context-based caching is totally dependent on the query horizon of the basic search mechanism. In the case of queries for local contents, context-based caching suffices, however supporting context-based service discovery in large-scale networks requires a different approach. Towards this end, we focus on location-based routing in the next paragraphs.

4.3.2 Location-based Routing

Location is a contextual dimension that is among the most important. In addition it also has the property that it is often related to the physical location of the web services. For example, the highest probability of finding pictures from Athens will probably be at the *location* Athens. It is also the case that often requests will be performed for a location that is very far from the location of the query originator, and that none of the service providers at a nearby location will be able to satisfy the request. An example is a person sitting on the airport in Trondheim wanting to access services in Athens, maybe to see pictures from today or book a taxi from Athens airport.

Unfortunately, in the case of large P2P networks, a contextual request for very remote contents will not be successful by traditional flooding approaches, and as indicated in the previous paragraphs, it will probably not be even be successful in the case of contextual caching. To overcome the limitations of the basic search, mechanisms are needed that enable access to remote contexts. One possible solution to the problem is to use a hierarchical overlay network based on the unstructured P2P network using the DESENT algorithm (Doulkeridis et al., 2007). Another solution is to organize the CASs in a DHT-based structured P2P network (in addition to the unstructured P2P network they already belong to) and use this for storage of location information. Using a hierarchical network involves possible problems with load-balancing and complex maintenance, so that given the stability and the very low churn rate of the CASs in our architecture, the DHT solution is most appropriate. We emphasize that in this case each CAS actually participates in both an unstructured P2P network and a DHT at the same time. Also note that services are not in any way indexed in the DHT, because of the high churn of the mobile devices and their services that would induce a very high maintenance cost (in contrast to the location of a particular CAS which can be expected to be extremely stable).

The maintenance of location-based in information is done as follows: The location of a CAS is represented as a contextual path based on a location taxonomy. For example, the location of a CAS near Acropolis would be *L=/Europe/Greece/Athens/Acropolis*. Each CAS has an identifier CID, in practice an IP address/port number. Key-value pairs (L_i, K_i) are inserted into the DHT. Thus given a query for a particular location L, a search in the DHT given L would give CID as the result. Often a location query is not performed for the most specific location, so we store in the index also the possible prefixes of the contextual (area) path. Thus in this particular example, 4 key-value tuples are inserted into the DHT. This extra information enables searches with different level of granularity, as well as extending queries to more general or more specific contexts. Note that a search for a location may return more than one CID, because the requested area contains more than one CAS. A search for a particular location requires only $O(\log n)$ messages, where n is the number of CASs in the network.

Number of peers N	25 or 100
Number of neighbors n	4
TTL value H	$1-4$
Dimensions of context D	3
Values per dimension V	$4-5$
Service contexts C_s	100
Services per directory S	1000
Cache size $c \times S$	$0 < c < 1$
Caching threshold t	$3-10$

Figure 5: Experimental setup default parameters for implementation of context-based caching.

5 Experimental Results

In this Section, we present the outcome of the experiments performed in order to evaluate the efficiency of our approach. We study: 1) the efficiency of context-based caching of web service descriptions, and 2) the effect of location-based routing of web service requests, as potential enhancements of traditional web service discovery in a P2P environment.

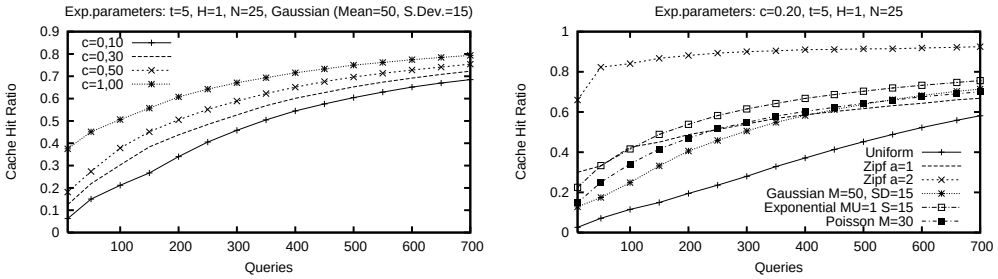
5.1 Context-based Caching

We implemented the proposed architecture of distributed service directories using the JXTA framework[7] for P2P applications. Each peer is represented as a JXTA peer that contains a list of neighboring peers. Neighbors are assigned to peers at startup to simulate coverage of a geographical region (each peer corresponds to a cell). Direct communication is achieved through exchange of messages between neighbors. Each peer contains a number of web services for various contexts and a cache for storing locally service descriptions that belong to web services outside its cell. The contextual service requests (Q_{cxt}), issued by mobile devices, are simulated by means of random query distributions in each peer. There are two functional modes: plain query forwarding or caching. Upon receiving a S_{req}, the peer forwards the query to its immediate neighbors and processes it locally. The originator peer retrieves the results (sets of contextual paths) of the query and, if caching is enabled, caches them, in order to process further identical requests locally, in an autonomous way.

We performed experiments on the implemented system. The basic experimental setup (see relevant table in Figure 5) comprises $N = 25$ or 100 peers, with $n = 4$ neighbors per peer in both cases. Each peer was assigned $S = 1000$ services at startup, uniformly distributed under $C_s = 100$ different contexts (if we consider $D = 3$ contextual dimensions, having each 4 or 5 discrete values). Unless mentioned otherwise, the cache size is 20% of the directory size, the caching threshold t is set to 5 query occurrences and the TTL is 1. We simulated query workloads by means of statistical distributions that show the frequency of occurrence for each of the 100 different contextual queries. A total of 2000 queries were generated in most of the cases. Unless explicitly stated, we allow cache forwarding in all experiments presented here.

The cache hit ratio (*CHR*) is the primary metric in this approach, because it defines the percentage of the queries that can be processed locally, without sending any messages to neighbors, thus influencing the overall system performance. A *CHR* value of $x\%$ means that out of 100 service requests x can be answered locally, by retrieving the service provider's identifier from the cache and then contacting it directly. As a result, the cost of processing these x requests in terms of number of messages is minimum (just 1 message), compared to the flooding alternative. Our goal is to measure the *CHR*. We start recording *CHR* results after 80% of the peer cache has become full to eliminate the cache warm-up effect. In particular, we measure the *CHR* of each peer, for every 10 queries. We compute the average *CHR* of all participating peers and present comparative charts (see Figure 6(a), 6(b)) for varying: 1) cache sizes and 2) query distributions respectively. We also performed a set of experiments for varying caching threshold values, and found out that it does not influence the *CHR* significantly.

[7] JXTA technology is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner, see http://www.jxta.org/



(a) Effect of cache size on cache hit ratio (b) Effect of different query distributions on cache hit ratio

Figure 6: Cache hit ratio dependency on cache size and query distribution.

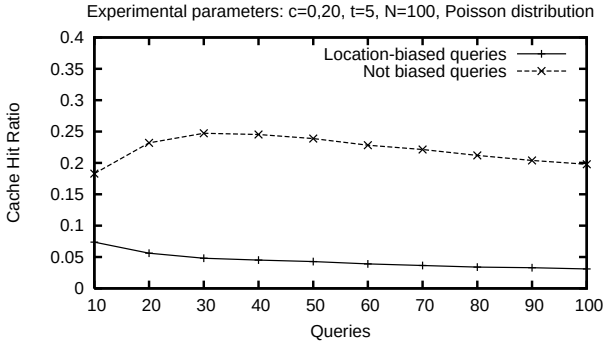
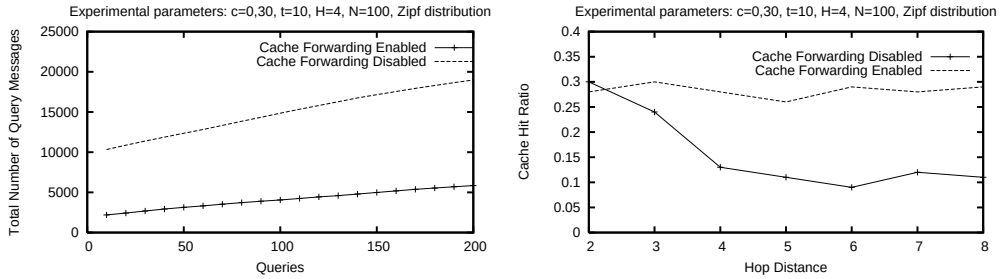


Figure 7: Cache hit ratio degradation vs query selectivity.

At first (see Figure 6(a)), we study the effect of different cache sizes on *CHR*. We tested different cache sizes ($c = 10\% - 100\%$), as a percentage of the directory size, for a Gaussian distribution (mean=50, st.deviation=15) of 100 distinct queries. Generally, increasing the cache size, results in *CHR* augmentation. However, after 650 queries and in order to increase the *CHR* from 68% to 80%, the cache size must increase from 10% to 100% of the directory size, so we educe that a relatively small benefit comes with a substantial cost for high *CHR*s. Also, notice that small cache sizes (10%) result in sufficiently good performance on the long run. Nevertheless, if a high *CHR* is required, then one should choose the largest cache size available.

Query distributions play an important role in the performance of caching. In Figure 6(b), we compare the query workloads generated by various random distributions. Besides the uniform distribution, we chose three distributions for query workloads that can represent very frequent occurrences for some queries, as well as two variants of the zipfian distribution. The zipf law implements $p(i) = C/i^a$ for $i = 1$ to n where C is the normalization constant (i.e., $\sum p(i) = 1$). As mentioned before, in our application scenarios, contextual queries tend to present high frequency at certain locations and times. The chart shows that the zipfian query distribution (with parameter $a = 2$) performed better, in terms of *CHR* (nearly 90%). As expected, the uniform distribution performed worse than the rest, however for the particular experimental setup the achieved *CHR* can be considered satisfactory. Generally, the more skewed the distributions, the higher *CHR* values they presented.

Then we studied the effect of query selectivity on *CHR*. The intuition is that for queries of low selectivity (i.e., many results returned) we expect that due to the high volume of the results frequent cache replacements take place. We conducted an experiment, shown in Figure 7, where we used 100 peers to represent 4 neighboring geographic locations (25 peers per location) and each peer held services having as contextual value the corresponding peer's location. We performed two measurements for: 1) location-biased queries for each peer and 2) uniformly distributed queries, in order to have low and high query selectivity respectively. The results show that location-biased queries exhibited a very poor performance (small *CHR*), due to the low query selectivity. In other words, each query resulted in a large number of services, which was comparable to the cache size, thus resulting in very frequent cache replacements (prac-



(a) Total number of query messages per peer (b) Cache hit ratio related to distance from popular sources

Figure 8: Studying the cache forwarding mechanism.

tically the result of each query caused cache replacement). The lesson learned here is that our approach regarding context-based caching in a P2P architecture requires careful parameter tuning, especially in the case that cache size is comparable to query selectivity.

In order to mitigate the impact of high query selectivity we conducted the rest of our experiments with a modified version of the query forwarding algorithm. Specifically, instead of forwarding each query that has not reached the maximum hop count TTL, a peer first checks its capability of satisfying the query. Thus, queries that can be responded by the peer's local directory are no further forwarded to its neighbors. On the basis of this query forwarding approach, a flooding query reaches the minimum number of peers that have relevant results and lie in its TTL horizon. However, in a real setting, decision making for query forwarding could be based on the number or quality of results that are omitted here for simplicity reasons.

We studied the impact of cache forwarding in the congestion of the peer network, as well as on the average CHR with an experimental setup of 100 peers deployed in a quadratic grid of side 10. The grid was divided in four regions, each one comprising 25 peers and representing a respective geographical area. The content of the peers of a region is characterized by the same value in the context dimension representing the location of the directory. The queries generated in each peer follow the zipf distribution with their frequency of occurrence depending on the location dimension. The workload pattern generated in each peer comprises the following types of queries with descending frequency of occurrence: 1) queries related to a specific popular region (location=Acropolis), 2) queries related to the current peer's region, 3) queries for services available in neighboring regions. Figure 8(a) depicts the average number of query messages processed by a peer in cases that cache forwarding is enabled or disabled. It is evident that in the cache forwarding case the number of messages processed by each peer is substantially lower due to the diffusion of popular results in all regions of the network of peers. Regarding Figure 8(b), it presents the fluctuation of the average *CHR* in each peer with respect to the hop distance from the popular region. Having cache forwarding enabled contributes to a more smooth variation in the *CHR* as the distance from the popular content increases. On the other hand, when cache forwarding in not allowed, *CHR* undergoes significant degradation as query TTL does not suffice for reaching the popular region.

5.2 Location-based Routing

The aim is to illustrate that for large P2P networks a contextual request for far contents (e.g. a person sitting on the airport in Trondheim wanting to access services in Athens, maybe to see picturea from today, etc.) will not be successful by traditional flooding approaches, and probably will not be even be successful in the case of contextual caching (this works well basically for queries for nearby locations)[8].

To address this problem we organize representative peers (one for each location) in a DHT overlay network, using as key the location path and as value the peer identifier responsible for the particular location. This means that a peer P_A within location A can always be found with logarithmic cost. Then P_A can

[8]Context-based caching with cache forwarding extends the query horizon of the query, so it improves flooding techniques in this sense, however this approach also imposes a limit to the extended query horizon. Thus for really large P2P networks, we propose location-based routing as an alternative approach.

Topology type 1	$N = 1600$, $n = 4$
Topology type 2	$N = 1600$, $n = 8$
Topology type 3	$N = 6400$, $n = 4$
Topology type 4	$N = 6400$, $n = 8$
TTL value TTL	10
Percentage of querying peers	20%
Zipfian query distribution parameter a	1.2
Services per directory S	50
Location hierarchy levels	4
Location hierarchy fan-out	4

Figure 9: Experimental setup default parameters for simulations of location-based routing.

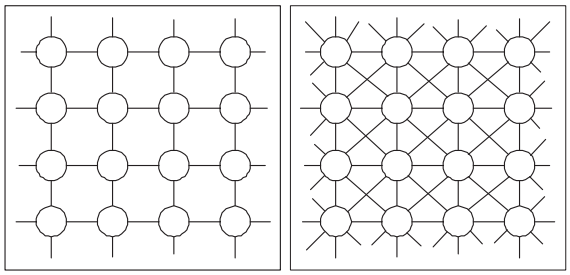


Figure 10: Illustration of the SQUARE topology with connectivity degree 4 (left) and degree 8 (right).

retrieve the services in its surrounding area even using a naive technique like flooding. We stress here that the DHT is used only for keeping the location of representative peers.

In order to test location-based routing we need a larger scale environment to test the feasibility of our ideas. Therefore we study location-based routing through simulations. We have built a simulator in Java for studying different search strategies in unstructured P2P networks.

The simulation setup (see also Figure 9) consists of a grid-like P2P network topology called SQUARE, which organizes peers in a square topology with each peer having 4 or 8 neighbors. This topology resembles a random graph with average connectivity degree d (4 or 8 in our case), with the difference that it's more dense than a random graph. Each pair of neighboring peers may share up to 4 common neighbors. This is necessary for ensuring connectivity of peers, even in the case of peer failure. For an illustration of this topology, see Figure 10. We spread this topology on top of a geographical area, so that each actual location is assigned to a set of peers that are near each other (see Figure 4 for an example of location hierarchy used). In this way we are able to simulate location-based content assignment on peers, by having each peer keeping service descriptions only of web services that reside in its area of coverage. We consider 4 different types of topologies, practically the combinations of 1600 and 6400 peers with connectivity degree 4 and 8.

Each peer hosts a query generator that creates queries for locations. A random distribution for modeling queries can be tested. Most interesting is the case of zipfian query distribution, which is usually adopted to model user behavior. Queries consist of one location. We consider queries for all (as many as possible) results to test the quality of the search in terms of completeness of results.

A number of peers N_{Q_p} are randomly picked to act as querying peers, each of them producing N_q queries. The queries are processed sequentially, i.e., for each querying peer, send its first query, then its second query and so forth. Both strategies are evaluated using the same parameters (querying peers, queries, etc.), in order to have, as much as possible, comparable results. In our simulations we assume 20% of the super-peers to receive queries from simple peers.

We study the comparative performance of the following search strategies:

1. Flooding: typical broadcast-based search with an associated TTL parameter.

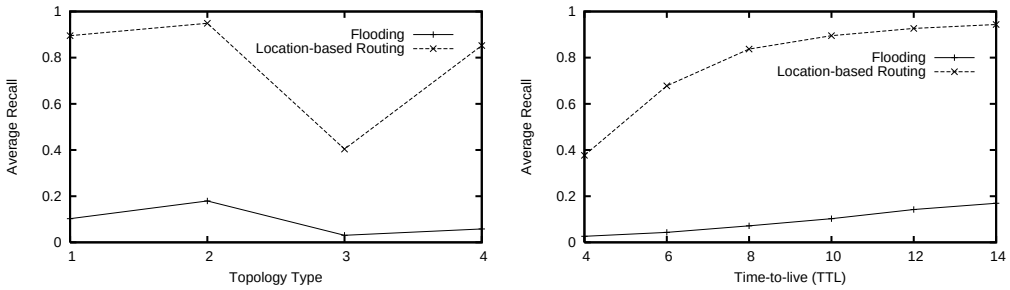


Figure 11: Average recall achieved for different topologies (left) and increasing TTL values for 1600 peers with avg.connectivity 4 (right).

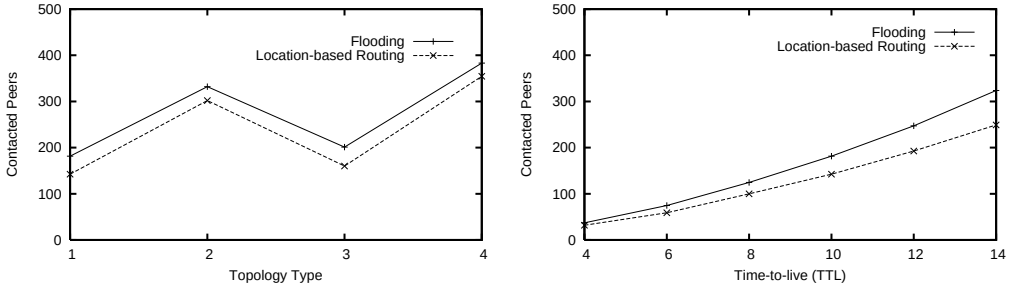


Figure 12: Number of contacted peers for different topologies (left) increasing TTL values for 1600 peers with avg.connectivity 4 (right).

2. Location-based routing: forward service requests to representative super-peers (based on location), using the DHT, then perform flooding in the surrounding area.

In Figure 11, we show the average recall achieved by the location-based routing approach compared to flooding using *the same number of messages*. Recall defines the number of relevant services retrieved as fraction of all relevant services. Higher recall values mean more complete results, which is an index of the search quality. The simulation results show that for a network of 1600 peers with $TTL = 10$, the recall of location-based routing is high (around $89.5 - 94.8\%$) compared to flooding, which achieves a poor $10 - 17.8\%$. This is expected since the content (services) is dependent on location, so using location-based routing the service request is directed to a peer in the queried location, then local flooding is effective, thus avoiding the limitations of blind flooding. We increase the network size to 6400 peers to test the scalability of our approach. Again, context-based routing outperforms the basic search mechanism. The smaller recall values obtained for the 6400 network are due to the fact that we keep the same TTL value, as in the case of 1600 peers. This is shown for 6400 peers when the peer connectivity is increased to 8 (topology type 4), where the recall values are closer to the ones for 1600 peers, since more peers are contacted by the service request.

In the right part of Figure 11, the increase in recall values with increasing TTL is depicted. This is an expected result, however the interesting part is that with location-based routing this increase is more significant and it comes with a higher rate.

In Figure 12, we show the number of contacted peers for each approach using the same number of messages. While location-based routing decreases the number of contacted peers, it manages to select peers to forward the service request, in such a way that they contain more relevant services, when compared to blind flooding. As mentioned before, the use of the same TTL value (10) keeps the number of contacted peers relatively the same for both settings. The effect of increasing TTL values is depicted on the right part of the figure.

6 Conclusions and Further Work

In this paper, we studied the integration of context-awareness in P2P service discovery, more specifically context-based caching and location-based routing for improving web service discovery in a P2P architecture of service directories. We used a context hierarchy as data model and presented how this work fits in the MobiShare architecture. Extended experiments, capitalizing on a JXTA-based prototype, illustrated the efficiency of caching. Via simulations we proved that location-based routing increases the quality and decreases the cost of web service discovery.

While in this paper neighbor proximity refers to geographical terms, future work will focus on extending the approach to study semantic, content-based proximity, where neighboring peers are considered based on similar content. This can be achieved by adopting semantic overlay networks (Crespo and Garcia-Molina, 2002) (Doulkeridis et al., 2007) that group together peers with similar content. We also intend to study how service churn, due to disconnections or user mobility, affects the efficiency of context-based caching and searching.

References

Akkiraju, R., Goodwin, R., Doshi, P., and Roeder, S. A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. In *Proceedings of the Workshop on Information Integration on the Web (IIWeb-03)*, 2003.

Arabshian, K. and Schulzrinne, H. GloServ: Global Service Discovery Architecture. In *Proceedings of Mobiquitous 2004*, 2004.

Baggio, A., Ballintijn, G., Steen, M. V., and Tanenbaum, A. S. Efficient Tracking of Mobile Objects in Globe. *The Computer Journal*, 44(5):340–353, 2001.

Chakraborty, D., Joshi, A., Yesha, Y., and Finin, T. Toward Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile computing*, 5(2):97–112, 2006.

Crespo, A. and Garcia-Molina, H. Semantic Overlay Networks for P2P Systems. Technical report, Stanford University, 2002.

Doulkeridis, C., Nørvåg, K., and Vazirgiannis, M. DESENT: Decentralized and Distributed Semantic Overlay Generation in P2P Networks. To appear in IEEE Journal on Selected Areas of Communications (J-SAC), 2007.

Doulkeridis, C., Valavanis, E., and Vazirgiannis, M. Towards a Context-aware Service Directory. In *Proceedings of the 4th VLDB Workshop on Technologies on E-Services (TES'03)*, 2003.

Doulkeridis, C. and Vazirgiannis, M. Querying and Updating a Context-aware Service Directory in Mobile Environments. In *Proceedings of the 2004 IEEE/WIC/ACM Int. Conference on Web Intelligence (WI'04)*, 2004.

Doulkeridis, C., Zafeiris, V., and Vazirgiannis, M. The Role of Caching and Context-awareness in P2P Service Discovery. In *Proceedings of MDM'05*, 2005.

Dustdar, S. and Treiber, M. A View Based Analysis on Web Service Registries. *Distributed and Parallel Databases*, 18(2):147–171, 2005.

Hu, T. H., Ardon, S., and Seneviratne, A. Semantic-laden Peer-to-Peer Service Directory. In *Proceedings of the 4th IEEE International Conference on P2P Computing (P2P'04)*, 2004.

Iyer, S., Rowstron, A., and Druschel, P. Squirrel: A Decentralized Peer-to-Peer Web Cache. In *Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

Jeckle, M. and Zengler, B. Active UDDI - an Extension to UDDI for Dynamic and Fault-tolerant Service Invocation. In *Proceedings of the International Workshop on Web Services Research, Standardization, and Deployment (WS-RSD'02)*, 2002.

Jensen, C. S. Research Challenges in Location-enabled M-services. In *Proceedings of the 3rd International Conference on Mobile Data Management (MDM'02)*, 2002.

Jensen, C. S., Kligys, A., Pedersen, T. B., and Timko, I. Multidimensional Data Modeling for Location-based Services. *VLDB Journal*, 13(1):1–21, 2004.

Lee, C. and Helal, S. Context Attributes: An Approach to Enable Context-awareness for Service Discovery. In *Proceedings of the 2003 Symposium on Applications and the Internet (SAINT'03)*, 2003.

Lee, D. L., Lee, W.-C., Xu, J., and Zheng, B. Data Management in Location-dependent Information Services. *IEEE Pervasive Computing*, 1(3):65–72, 2002.

Leontiadis, E., Dimakopoulos, V. V., and Pitoura, E. Cache Updates in a Peer-to-Peer Network of Mobile Agents. In *Proceedings of the 4th IEEE International Conference on P2P Computing (P2P'04)*, 2004.

Mostefaoui, S. K. and Hirsbrunner, B. Towards a Context-Based Service Composition Framework. In *Proceedings of the 1st International Conference on Web Services (ICWS'03)*, 2003.

Nørvåg, K., Doulkeridis, C., and Vazirgiannis, M. Taxonomy Caching: A Scalable Low-cost Mechanism for Indexing Remote Contents in Peer-to-Peer Systems. In *Proceedings of the 2nd IEEE International Conference on Pervasive Services (ICPS'06)*, 2006.

Patro, S. and Hu, Y. C. Transparent Query Caching in Peer-to-Peer Overlay Networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.

Pokraev, S., Koolwaaij, J., and Wibbels, M. Extending UDDI with Context-aware Features Based on Semantic Service Descriptions. In *Proceedings of the 1st Int. Conference on Web Services (ICWS'03)*, 2003.

Schmidt, C. and Parashar, M. A Peer-to-Peer Approach to Web Service Discovery. *World Wide Web*, 7(2):211–229, 2004.

ShaikhAli, A., Rana, O., Al-Ali, R., and Walker, D. UDDIe: An Extended Registry for Web Services. In *Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT-w03)*, 2003.

Steen, M. V. and Ballintijn, G. Achieving Scalability in Hierarchical Location Services. In *Proceedings of the 26th International Computer Software and Applications Conference (CompSac)*, 2002.

Valavanis, E., Ververidis, C., Vazirgiannis, M., Polyzos, G. C., and Nørvåg, K. Mobishare: Sharing Context-dependent Data and Services from Mobile Sources. In *Proceedings of IEEE/WIC International Conference on Web Intelligence (WI'03)*, 2003.

Wang, C., Xiao, L., Liu, Y., and Zheng, P. Distributed Caching and Adaptive Search in Multilayer P2P Networks. In *Proceedings of the 24th Int. Conference on Distributed Computing Systems (ICDCS'04)*, 2004.

Zheng, B., Xu, J., and Lee, D. L. Cache Invalidation and Replacement Strategies for Location-dependent Data in Mobile Environments. *IEEE Transactions on Computers*, 51(10):1141–1153, 2002.

Zhu, F., Mutka, M., and Ni, L. Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, pages 81–90, 2005.