

# Towards a Context-Aware Service Directory

Christos Doulkeridis, Efstratios Valavanis and Michalis Vazirgiannis

Database Systems Laboratory  
Department of Informatics  
Athens University of Economics and Business (AUEB)  
10434 Athens, Greece.  
{cdoulk, valavani, mvazirg}@aueb.gr

**Abstract.** Advances in the areas of mobile computing and web services lead to new scenarios of use and innovative applications. Our approach involves mobile devices that act not only as requestors of data, but as data providers as well, providing access to their data through web services. Context plays an important role in such a scenario, when used to improve existing service discovery mechanisms, by finding the most appropriate service conforming to the search criteria. In this paper we propose a context-aware service directory and we investigate the process of service discovery with respect to contextual information. We adopt the MOEM model for representing contextual information within the service directory and we try to improve the performance of searching. Furthermore, we deal with issues related to the temporal dimension of context, namely prediction of service availability. We also present some preliminary experimental results concerning the search costs.

**Keywords:** Service directory, service discovery, context-aware computing, mobile computing, web services

## 1 Introduction & Motivation

Nowadays mobile computing comes into the everyday life more and more. Mobile devices, like 3G cellular phones, personal digital assistants (PDAs), digital cameras, laptops, keep getting widely accepted and their usage has become a commodity. As a result, a number of possible scenarios of use are presented every day. In contrast to desktop applications, mobile applications are characterized by dynamic changes of the actors' environment and the restrictions imposed by the size and capabilities of the actual devices [1]. Device and user context can provide valuable information that assists applications to adapt to the needs of specific situations, reducing the waste of resources and offering perspectives for the design of novel context-aware applications [2].

Mobile and pervasive systems make extensive use of heterogeneous data residing on diverse devices. This results in a dynamic system, which comprises numerous mobile information resources, e.g. databases, file repositories, media, etc. Accessing mobile data through web services offers two major advantages: it hides the heterogeneous nature of data scattered around the world, and it

provides a globally accepted, well-defined interface for data access. In such a dynamically evolving environment, service discovery plays a critical role in the system functionality.

The majority of web service architectures use service directories, i.e. registries of service descriptions, which facilitate service discovery based on parameters. However, when the entities involved (providers and requestors) are static, there exist alternative means of communication, such as direct connection or through an FTP server or even a Web site. In the case of more dynamic architectures, the role of a service directory is crucial, since it is responsible for enabling the discovery of available services by taking into account several parameters such as: location of the service, hosting device capabilities, type of the returned results.

Obviously a context-aware service directory would facilitate service discovery and increase its precision. So far, discovery mechanisms focus either on exact matching based on some service attribute or on semantic service discovery (more ambitious approaches). We argue that using the context of devices and services will make the existing methods more efficient. Thus, we argue in favor of a service directory that can answer queries like the following ones:

- "Which services (or devices) are available in location L at time T?"
- "Which services return results that the requestor device can represent?"
- "Which services were available at timestamp T?"
- "Which services are published by user U and by his device D?"

The innovative part of our work is embedding contextual information within a service directory, thus creating a context-based index for efficient retrieval of services. The contribution of our work is the enhancement of service discovery by taking into account the available contextual information.

The rest of the paper is organized as follows. In section 2, we present our approach for a context-aware service directory and we describe the process of searching in detail. In section 3, we examine the temporal domain by predicting service availability. In section 4, some preliminary experimental results are presented. Section 5 is about the related work, and finally section 6 concludes the paper.

## 2 Context-Awareness in the Service Directory

Recent surveys on context-aware computing, such as [1], recognize that:

- Only a few types of context (usually *location*) are actually used by applications, either because it is hard to collect and represent more complex types of context or because it is considered useless.
- Context histories are rarely used.
- Applications should tend to minimize user distraction [2] by avoiding to prompt the user to provide contextual information explicitly.

Regarding these points, our approach provides a context representation model capable of handling any type of contextual information, as long as this can be

described in terms of key-value pairs. Keys are called *dimensions* and they can take one or more discrete values, or they can range over a specific domain. We also explore context histories and we describe a couple of interesting use cases. Moreover, our approach deals with any number of context types, as we will show in the following sections. At the same time, we try to respect the user’s reluctance of providing information to the system.

We argue that context plays an important role in service discovery, as an additional set of criteria that enables locating the most suitable service, after having found the appropriate service category semantically [6]. Defining a global context model that applies to all services is both complicated and imposes several useless dimensions. Therefore we define context per service category.

In the following subsections, we will focus on a service category that includes file-sharing services, in order to demonstrate the feasibility of our approach. It is needless to say that the results of our study can be applied in any service category that is included within the service directory. The only obstacle is discovering the most interesting context dimensions for each service category.

## 2.1 Data Model

In order to introduce a context-aware service directory, we adopt a representation model that is capable of holding information presenting different facets under different contexts. We exploit this property and we represent service categories as multidimensional entities. Context is used to differentiate services that belong to the same service category.

Intuitively, the service directory is represented by a Multidimensional OEM graph [3], modelling services as atomic nodes (i.e. leaves of the graph). In fact every service category is represented by an (degenerate) (see 2.4) MOEM graph, which stores services in its leaves and uses multidimensional entities to differentiate these services based on the contextual information under which they hold.

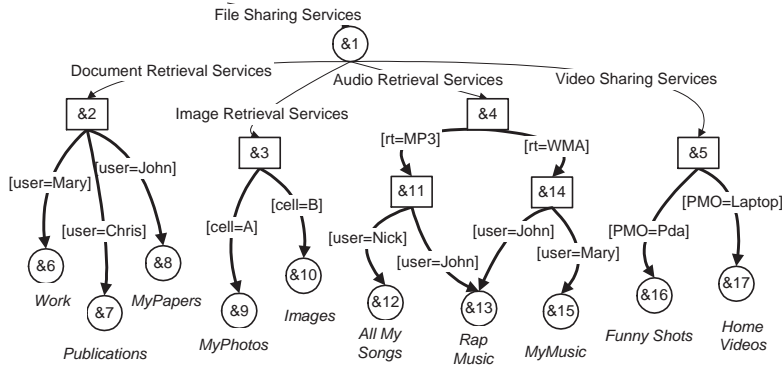


Fig. 1. An MOEM graph representing (part of) a service directory.

We will show the usage of an MOEM graph as a service directory through an example. Figure 1 shows (part of) a service directory that represents file-sharing services. The graph comprises two kinds of nodes: rectangular nodes, called *multidimensional nodes*, represent multidimensional entities that present different facets under different contexts, whereas circular nodes, representing entities, are called *atomic* or *context nodes*, depending on whether they are leaves of the graph or not. Atomic nodes are able to hold some kind of data and in our approach they represent web services. There are also two kinds of edges: *context* and *entity edges*. Context edges are represented by thick lines and they define the contexts (represented by labels called *explicit contexts*) under which the services hold. Entity edges are plain edges that represent relations between entities.

In figure 1, the file-sharing service category is further analyzed in four sub-categories, namely *document retrieval*, *image retrieval*, *audio retrieval* and *video-sharing* services. The actual services that belong to each of these (semantic) categories are distinguished by means of context. For example, services represented by nodes &6, &7 and &8 return documents, but they belong to different users. In a similar way, two image retrieval services are registered in the directory: service &9, which is located in cell A, and service &10, located in cell B (assuming a cell-based space model). We also notice that different dimensions of context can apply to the same services. This is the case with services &12, &13 and &15, which belong to different users and return different types of files (dimension: *rt* - return type). Service &12 returns MP3 files and service &15 returns files in WMA format, whereas service &13 returns both types of files. Finally, differentiating video-sharing services &16 and &17 is accomplished through the dimension PMO (Primary Mobile Object) [6] that defines the type of the device providing the service. Service &16 resides on a PDA, whereas service &17 on a laptop.

## 2.2 Searching for Services

Having explained the role of the MOEM graph as a model for representing a service directory, it is important to elucidate the service discovery mechanism. We introduce a breadth first search algorithm to traverse the graph and to spot the services that match the search criteria. This algorithm takes as input the identifier of the root and a context specifier, which is a syntactical construct representing the context under which the search is performed.

Intuitively, we note that a path starting from the root must exist for every service that is supposed to belong to the result set of the search. This path should only consist of context edges with explicit contexts consistent with the context specifier representing the search criteria.

The algorithm starts its execution from the root of the graph, and examines all edges that depart from the root. For each of these edges, its explicit context is compared to the context specifier, which determines the context under which the search is performed, and if their *intersection* [3] is equal to the *empty context* [-] [3], then the subgraph pointed by this edge is ignored. Alternatively, the

node (pointed by the edge) is kept for further processing. When the algorithm finds an atomic node, this node represents a service that is in accordance with the search criteria, and it will be included in the search results. The algorithm ensures that only the atomic nodes, which should be returned, will be examined.

### SEARCH ALGORITHM

```

INPUT:  nRoot      // the identifier of the root
        strContext // context specifier describing the context
OUTPUT: SelNodes  // list with the ids of the selected services
SEARCH ALGORITHM:
STEP1: Mark node (&nRoot), and add it to queue
STEP2: While the queue is not empty...
    STEP2(A): Get the next node (&n) from the queue
    STEP2(B): If (&n) is an atomic node, add it to SelNodes,
    and go to STEP2
    STEP2(C): Get all edges that start from (&n) and put
    them in a list
    STEP2(D): While the list is not empty...
        STEP2(D)(i): Take the next edge from the list
        STEP2(D)(ii): If the intersection of the explicit context of
        the edge and strContext, is different than the empty set, and
        if the node (&p) pointed by the edge is not marked, then add
        (&p) to the queue and mark (&p)
        STEP2(D)(iii): Remove the edge from the list
        STEP2(D)(iv): Go to STEP 2(D)
    STEP2(E): Go to STEP 2
STEP3: Return SelNodes

```

For example, consider the query: "Find all services provided by user B", which is issued at the service directory depicted in figure 2(a). The context specifier representing the query is:  $[user=B]$ . It is easy to understand that applying the search algorithm on the graph, returns service &12. Note that the subgraph with node &5 as its root, is completely excluded from the search, since it holds under the context  $[user=A]$ , thus not conforming to  $[user=B]$ .

### 2.3 Improving Search

We argue that the same graph can be represented by numerous equivalent graphs, by simply changing the hierarchy of the dimensions that define context. We will show that the time required to perform a search depends on the morphology of the graph that represents the service directory.

Assume that context is defined for document retrieval services by means of the following dimensions: PMO, user, and rt - Return Type. We summarize the contextual information in a table, called *State Table*.

PMO	User	Return Type	Service
PDA	A	pdf	&10
Laptop	A	pdf,ps	&11
Laptop	B	ps	&12

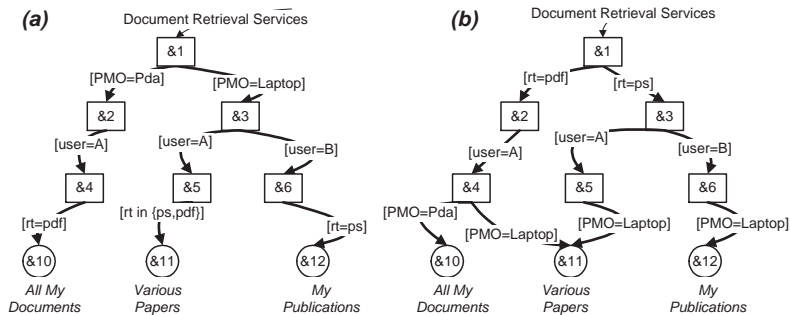
**State Table**

We will construct an MOEM graph that represents the information held in the *state table*, with the help of a construction algorithm. Let us choose the dimensions with respect to their order on the *state table*, i.e. at first dimension *PMO*, then *User*, and finally *rt* (return type). Consider the root (&1) of the graph depicted in figure 2(a). In order to apply the algorithm on the *state table*, we start with service &10, and we create and then draw the path under which the service holds, namely:  $PDA \rightarrow A \rightarrow pdf \rightarrow \mathcal{E}10$ . Similarly, for service &11 we create two paths:  $Laptop \rightarrow A \rightarrow pdf \rightarrow \mathcal{E}11$ , and  $Laptop \rightarrow A \rightarrow ps \rightarrow \mathcal{E}11$ . At last, the path that is produced for service &12 is:  $Laptop \rightarrow B \rightarrow ps \rightarrow \mathcal{E}12$ . The graph that is constructed by these paths is depicted in fig. 2(a).

### CONSTRUCTION ALGORITHM

INPUT: State Table (ST)  
 OUTPUT: MOEM graph (G) representing the service directory  
 CONSTRUCTION ALGORITHM:  
 (foreach service in ST) begin // i.e. each line of the state table  
 - create all possible paths to the service  
 - draw the produced paths on G (if a subpath of a path already exists, extend it to point to the specific service)  
 end

Let us now try to construct an equivalent graph. This graph is depicted in fig. 2(b), which is constructed by selecting the dimensions with a different order. At first, we choose dimension *rt*, then *User*, and finally *PMO*. Applying the construction algorithm, results in the following paths:  $pdf \rightarrow A \rightarrow PDA \rightarrow \mathcal{E}10$  (for service &10),  $pdf \rightarrow A \rightarrow Laptop \rightarrow \mathcal{E}11$  and  $ps \rightarrow A \rightarrow Laptop \rightarrow \mathcal{E}11$  (for service &11), and  $ps \rightarrow B \rightarrow Laptop \rightarrow \mathcal{E}12$  (for service &12).



**Fig. 2.** Two equivalent graphs holding contextual information.

Regarding search performance, we claim that search is faster when the queries concern dimensions that are closer to the root. In many cases we can optimize service search by restructuring the directory. Performance is improved due to a reduction of the number of nodes and edges examined by the algorithm.

For example, consider the query: *"find all services that return documents of type pdf"*, which is issued against each of the graphs depicted in fig. 2(a) and fig. 2(b). When the search algorithm is applied to the graph in fig. 2(a), the following nodes will be examined: &1, &2, &3, &4, &5, &6, &10, &11, as well as all the edges of the graph, and it will finally retrieve nodes &10 and &11. The same algorithm applied on the graph in fig. 2(b) would examine the nodes: &1, &2, &4, &10, &11, as well as five (5) from the total nine (9) edges, and would retrieve nodes &10 and &11, too. Obviously, when the query is issued against the graph in fig. 2(b), the performance of the search is improved, and the difference is significant even in the case of graphs of relatively small size.

## 2.4 Complexity Analysis

This section examines the space requirements and time complexity that characterize the service directory. Our aim is twofold: first, to identify the search cost, and second, to determine the space cost induced by this structure.

We can make some interesting observations based on the description of the graph. First, the MOEM graph is actually a degenerate MOEM graph, since it is mainly constructed by successive context edges that point either to multidimensional nodes or to atomic nodes representing the actual services. There are practically no entity edges (except from those defining service categories) and the only context nodes that exist in the structure are those that represent the available services in the directory. Moreover, the morphology of the graph is similar to a tree, with the only difference that there can exist leaves, which belong to different subtrees (i.e. service nodes pointed by more than one edge, like service &11 in fig. 2(b)). Based on these observations, we find that the graph presents several similarities with multiple way (m-way) search trees. This point will be clarified by the analysis that follows.

Consider a service directory that contains  $n$  services, and  $k$  dimensions (defining the context). Assume that each dimension can take up to  $m$  distinct values. Further on, let us assume a 'complete' graph, i.e.  $m$  edges starting from every node, which is the worst-case scenario both in terms of space and time complexity. If we think of the graph as a tree, then its height is:

$$h = k + 1$$

since  $k$  is the number of dimensions and each level corresponds to context edges describing one dimension. The fan-out of the tree is equal to the number of values  $m$  that a dimension can take, so:

$$fo = m$$

which is the number of edges starting from a node. The number of nodes for a complete tree of height  $h$  and fan-out  $m$  is:

$$N_m(h) = 1 + m + m^2 + \dots + m^k = O(m^k)$$

The mean value of the number of node accesses for a successful search (for exactly one service, i.e. node) is:

$$E = m/2 * k + 1 = O(mk)$$

since  $m/2$  is the mean value of outgoing edge comparisons in each node in order to find the edge that matches the search criteria, and the search is performed in  $k$  levels. When searching for all services that hold under a specific context:

$$E = m/2 * k + n_i = O(mk + n_i)$$

with  $n_i \leq n$  representing the number of services under this context. Generally:

$$\sum n_i = n, i = 1..m^k$$

and if a uniform distribution is presumed, then:

$$n_i = n/m^k$$

### 3 Temporal Aspects of Context and its Applications

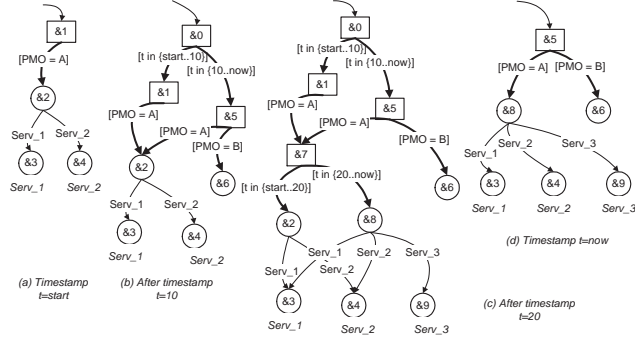
This section deals with issues that arise when the temporal domain is examined. So far, we have avoided explicit references to the temporal dimension within the service directory, for this dimension distinguishes itself from the usual dimensions of context. This is, mainly, due to the fact that the frequency of modifications concerning time is expected to be enormous compared to other dimensions, since the latter are usually not affected by frequent changes. In addition to that, when temporal information is taken into account, some interesting applications and some new scenarios of use come up, that highlight its distinguishing nature.

The basic motivation of this part of our work is to predict the availability of services, in terms of time and geographical space, i.e. when and where a service is expected to be available in the future. In many cases this could be achieved using the current state or a sequence of states of the device hosting the service. By analyzing the past behavior of a user, we claim that it is possible to discover interesting patterns, thus allowing the prediction of future appearances of devices, and therefore the prediction of service availability.

We try to exploit the fact that service availability is directly dependent on regular habits of users offering services. For example, consider the case of a user that provides access to his services only when he is at work, from Monday to Friday during working hours. A request, for such a service, issued at night would normally get no results, whereas in our case the requestor would be informed that the service will probably be available the next morning. We are in favor of a system that returns approximate results, rather than no results at all.

In [4], the authors show that MOEM is capable of representing temporal changes in semistructured databases. We take advantage of this property of MOEM to maintain time-dependent information about services and their availability. We will explain the usage of such a graph in our case with the help of an example.

Figure 3(a) shows the initial state of the MOEM graph that keeps information about the PMOs, within the boundaries of a single cell, and about the services they publish. At timestamp  $t=start$ , only one PMO, named **A**, is registered and it publishes two services: **Serv1** and **Serv2**. At timestamp  $t=10$ , a new



**Fig. 3.** The service directory describing the devices and the services published, at various time instances.

PMO named B, which provides no services, enters the cell and is registered in the system (fig. 3(b)). A new multidimensional node with id &0 is created, which groups together nodes &2 and &6 that represent the registered PMOs in the cell before and after  $t=10$  respectively. Notice that the new PMO is accessible from the root, only for timestamps before  $t=10$ . Then, at  $t=20$ , PMO A publishes a new service: Serv3 (fig. 3(c)). A new multidimensional node with id &7 replaces node &2, and points to node &2 that represents the services published before  $t=20$ , and to node &8 that encapsulates all published services after  $t=20$ . Finally, fig. 3(d) shows a snapshot of the current state of the directory, containing both PMOs and all the services they publish.

## ALGORITHM: DISCOVERING THE ONLINE TIME INTERVALS

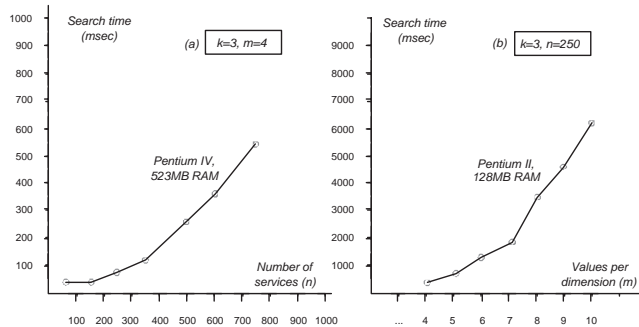
STEP1: Find all context edges with explicit context [PMO=X], since they point to the PMO in question  
STEP2: For each of these edges...  
STEP2(A): Find all starting nodes (father nodes)  
STEP2(B): Find all context edges pointing to these nodes  
STEP2(C): Extract the time intervals from these edges  
STEP3: Accumulate all intervals, apply pattern-matching techniques

In order to predict the future behavior of services, we need to accumulate the time intervals they were available, and try to apply pattern-matching techniques on them. We present a simple, case-specific algorithm for extracting the time intervals from the directory. The algorithm takes as input a specific PMO and finds all the related time intervals. The algorithm results in a time interval (or a union of time intervals), which is kept for every service, showing the expected time of availability for each service.

## 4 Experimental Results

We have performed a series of experiments concerning service lookup with respect to contextual information. Given a graph (of varying size) representing the service directory, we measured the time required to find *all services* that hold under the specified context. All experiments are performed in main memory.

Figure 4(a) shows the results when context is defined by  $k=3$  dimensions, with each dimension ranging over  $m=4$  discrete values. Figure 4(b) depicts the results produced from a directory that contains  $n=250$  services that hold under  $k=3$  dimensions of context. The number of values  $m$  (i.e. the fan-out of the graph) that each dimension can take is shown on the X-axis.



**Fig. 4.** Time required for service retrieval for:(a) $k=3$ ,  $m=4$ , and for various numbers ( $n$ ) of services, (b) $k=3$ ,  $n=250$ , and for various numbers of values for dimensions ( $m$ ).

The charts show the linear relationship between the required search time and (a) the number of services  $n$ , (b) the number of values per dimension  $m$ . Notice that this is in accordance with our complexity analysis (see 2.4), which shows that the search time exhibits a linear relationship with  $m$ ,  $k$ , and  $n$ , when we search for all services holding under a specified context.

## 5 Related Work

Some protocols support exclusively directory-less operation, based on multicasting (like UPnP [14, 19] and HP Cooltown [16]), in which all the participants can advertise or request service from each other in an ad-hoc manner. Others use service repositories (Jini lookup tables, SLP Directory Agents or Salutation SLMs [14, 15, 17, 18]). Service discovery is usually based on service types and attributes not taking context into account, except Cooltown. However, Cooltown's [16] concept of context is rather informal with descriptions that can be unstructured web pages.

The UDDI specification [20] provides a model used to describe services and facilitate service and business discovery. The discovery mechanism adopted is still

at an immature level, and service classification is based on *tModels*, which present certain limitations. The ebXML approach [21] shares some common features with UDDI, but goes a step further by trying to introduce an infrastructure for business-to-business communication. The data model treats all entities as objects, thus providing a generic model. Nevertheless, discovery mechanisms are rather limited and categorization is achieved through the use of *Slots*, a similar concept to *tModels*. Like UDDI, the ebXML specification does not address issues like mobility of services, semantic discovery and context-awareness.

The most popular model for representing raw contextual data are key-value pairs [8] and arbitrary annotations (tags) [9]. Object-oriented approaches [10, 13] model real world entities as objects with types, names, capabilities and properties to allow computer systems to share and use the user's perceptions of the real world (location, time, environmental condition). This approach is quite powerful, but involves an important overhead for constructing the objects. Finally, logic-based models [11] keep context data as facts in rule-based systems. Nearly all of these approaches use their own ad-hoc data structures to model contextual data. This results in many islands of contextual information that work well as stand-alone entities, but hinder communication and exchange of information.

The data formats used to model contextual information should be independent of any requirements concerning hardware, programming language or operating system [7]. Our approach for storing contextual information is based on the use of the MOEM model, which is a generic model that fulfills such requirements. Furthermore, in our case, exchanging information regarding context (just like in the case of stick-e notes [12]) is supported with the use of MXML [5], a markup language that incorporates the notion of context in XML.

## 6 Conclusions & Future Work

In this paper, we presented a context-aware service directory, we explained the process of searching for services based on contextual information, and we presented some preliminary experimental results. We also discussed about prediction of service availability. The innovative part of our work is creating a context-based index for efficient service retrieval and our contribution is the enhancement of service discovery by taking into account the available contextual information.

Our future work will focus on the synchronization and communication of distributed context-aware service directories, and we will try to define formally the notion of context regarding services. We also intend to exploit context-awareness in order to improve existing service discovery mechanisms.

## References

1. G. Chen and D. Kotz *A Survey of Context-Aware Mobile Computing Research*. Dartmouth Computer Science Technical Report TR2000-381.
2. M. Satyanarayanan *Pervasive Computing: Vision and Challenges*. IEEE Personal Communications, August 2001.

3. Y. Stavrakas and M. Gergatsoulis. *Multidimensional Semistructured Data: Representing Context-dependent Information on the Web*. In Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering (CAISE'02), Toronto, Canada, May 2002.
4. Y. Stavrakas, M. Gergatsoulis, C. Doukeridis and V. Zafeiris. *Accommodating Changes in Semistructured Databases Using Multidimensional OEM*. In Proc. of the 6th East European Conference, Advances in Databases and Information Systems (ADBIS'02), Bratislava, Slovakia, September 2002.
5. Y. Stavrakas, M. Gergatsoulis, and P. Rondogiannis. *Multidimensional XML*. In the Proceedings of the Third International Workshop, Distributed Communities on the Web, DCW 2000, Quebec City, Canada, June 19-21, 2000, pp.100-109.
6. E. Valavanis, C. Ververidis, M. Vazirgiannis, G. C. Polyzos, K. Norvag. *MobiShare: Sharing Context-Dependent Data and Services from Mobile Sources*. To appear in the Proceedings 2003 IEEE/WIC International Conference on Web Intelligence, WI2003, Halifax, Canada, October 13-17, 2003.
7. J. Hong and J. Landay. *An Infrastructure Approach to Context-Aware Computing*. Human-Computer Interaction, 16:287-303, 2001.
8. G.M. Voelker and B.N. Bershad. *Mobisaic: An Information System for a Mobile Wireless Computing Environment*. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 185-190, Santa-Cruz, California, December 1994. IEEE Computer Society Press.
9. P.J. Brown, J.D. Bovey and X.Chen. *Context-aware Applications: from the Laboratory to the Marketplace*. IEEE Personal Communications, 4(5):58-64, October 1997.
10. N. Davies, K. Cheverst, K. Mitchell and A. Friday. *Caches in the Air: Disseminating Tourist Information in the GUIDE System*. In Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, Louisiana, February 1999. IEEE Computer Society Press.
11. J. Bacon, J. Bates and D. Halls. *Location-oriented Multimedia*. IEEE Personal Communications, 4(5):48-57, October 1997.
12. J. Pascoe. *The Stick-e note Architecture: Extending the Interface Beyond the User*. In Proceedings of the 1997 International Conference on Intelligent User Interfaces, pp. 261-264, Orlando, FL, January 1997, ACM Press.
13. A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. *The Anatomy of a Context-Aware Application*. In Wireless Networks 8(2-3): 187-197, 2002.
14. S. Helal, C. Lee. *Protocols for Service Discovery in Dynamic and Mobile Networks*. International Journal of Computing Research, vol.22, no.1, pp.1-12, 2002.
15. E. Guttman. *Service Location Protocol: Automatic Discovery of IP Network Services*. IEEE Internet Computing, 3(4):71-80, 1999.
16. T. Kindberg, et al. *People, Places, Things: Web Presence for the Real World*. Technical Report HPL-2000-16, Internet and Mobile Systems Laboratory, HP Laboratories Palo Alto, February, 2000.
17. Salutation Consortium, <http://www.salutation.org/>, 2003.
18. Microsystems Inc. Jini Network Technologies, <http://www.sun.com/jini/>, 2003.
19. Universal Plug and Play (UPnP), <http://www.upnp.org/>
20. UDDI *The UDDI Technical White Paper*. <http://www.uddi.org>
21. ebXML <http://www.ebxml.org>