Dynamic Service Composition in Pervasive Computing

Swaroop Kalasapur, *Member*, *IEEE*, Mohan Kumar, *Senior Member*, *IEEE*, and Behrooz A. Shirazi, *Member*, *IEEE*

Abstract—Service-oriented architectures (SOAs) promise to provide transparency to resource access by exposing the resources available as services. SOAs have been employed within pervasive computing systems to provide essential support to user tasks by creating services representing the available resources. The mechanism of combining two or more basic services into a possibly complex service is known as service composition. Existing solutions to service composition employ a template-matching approach, where the user needs are expressed as a request template, and through composition, a system would identify services to populate the entities within the request template. However, with the dynamism involved in pervasive environments, the user needs have to be met by exploiting available resources, even when an exact match does not exist. In this paper, we present a novel service composition mechanism for pervasive computing. We employ the service-oriented middleware platform called Pervasive Information Communities Organization (PICO) to model and represent resources as services. The proposed service composition mechanism models services as directed attributed graphs, maintains a repository of service graphs, and dynamically combines multiple basic services into complex services. Further, we present a hierarchical overlay structure created among the devices to exploit the resource unevenness, resulting in the capability of providing essential service-related support to resource-poor devices. Results of extensive simulation studies are presented to illustrate the suitability of the proposed mechanism in meeting the challenges of pervasive computing—user mobility, heterogeneity, and the uncertain nature of involved resources.

Index Terms—Pervasive computing, dynamic service composition, graph models, middleware, heterogeneous devices.

1 Introduction

Service-oriented architectures are suitable for designing and deploying pervasive computing environments in which facilitating user tasks is a major focus. Typically, user tasks will require a number of resources possibly spread over the networked environment. If the user requirements are known a priori, the required resources can be tuned for user access when desired. However, such a static design of systems would limit their possible usage by other user tasks. Due to the growing applications of pervasive computing, there is a need to provide support to user tasks in the face of dynamic challenges such as heterogeneity, resource restrictions, and mobility. Service-oriented environments promise flexibility in terms of user support, as well as better resource utilization. By modeling the available resources as services and by designing a mechanism to access the available services, a pervasive computing environment can be effectively transformed into a service-oriented environment. Further, by creating access mechanisms that can dynamically utilize available services in an efficient way to provide the *best possible* support for user tasks, guaranteed results can be delivered.

- S. Kalasapur is with Samsung Research USA, 75 W. Plumeria Dr., San Jose, CA 95134. E-mail: s.kalasapur@samsung.com.
- M. Kumar is with the University of Texas at Arlington, PO Box 19015, Arlington, TX 76019. E-mail: kumar@cse.uta.edu.
- B.A. Shirazi is with Washington State University, PO Box 642752, Pullman, WA 99164. E-mail: shirazi@wsu.edu.

Manuscript received 9 Mar. 2006; revised 27 June 2006; accepted 20 Sept. 2006; published online 9 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0055-0306.
Digital Object Identifier no. 10.1109/TPDS.2007.1039.

Service composition mechanisms are being employed to deliver support to complex user tasks within service-oriented environments. The mechanism of combining two or more services together to form a complex service is known as service composition. Service composition mechanisms are classically treated as extensions to service discovery techniques. Typically, a service composition system accepts a complex user task as an input and attempts to meet the needs of the task at hand by appropriately matching the task requirements with the available services. The approaches defined in [1], [2], [3], and [4] have the requirement that there is a one-to-one correspondence between the required services and those that are available. If one or more of the required services cannot be located, existing systems either fail or enter an extended mode of discovery.

Typical pervasive computing environments are envisioned to embody a number of devices with a very rich set of functionalities. In the presence of such richness, it is desirable to dynamically *combine* available basic services (as building blocks) to create composite services. User and application service requests typically require support from multiple services. Current solutions are limited to a *one-to-one matching* between the requested services and those that are available. The dynamic service composition mechanism presented in this paper identifies possible ways to combine existing services to arrive at a service required by a user/application request. In the rest of the paper, the requests that need service support are also referred to as *tasks*.

The main contributions of this paper include: 1) a mechanism to model services, 2) a graph-theory-based service composition mechanism, and 3) a hierarchical service overlay in pervasive computing environments. The service

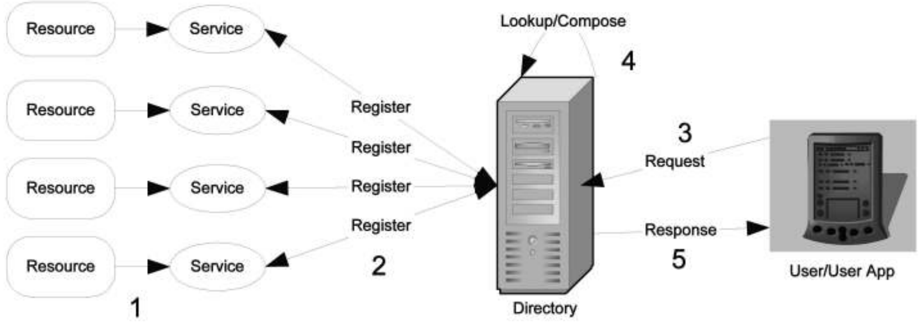

Fig. 1. Overview of operations in SOA. (1) Resources are *exported* as services. (2) Services are registered at the directory. (3) Users query the directory for service(s). (4) Directory performs lookup/composition on registered services. (5) Directory returns results to user.

model is aimed at capturing the features and useful characteristics of resources in a pervasive environment using the Pervasive Information Communities Organization (PICO) [5] middleware. Semantic descriptions of services are used to 1) organize services in a repository and 2) retrieve services required by tasks. The dynamic composition mechanism built using graph techniques has the capability of combining existing services to arrive at a complex solution. The hierarchical service overlay formed is capable of handling a number of challenges in pervasive computing such as resource heterogeneity, user and resource mobility, locality of service provisioning, and so forth. In an earlier work [6], we have introduced the basic composition mechanism and the hierarchical service overlay [7] mechanism. In this paper, we extend the initial models and present detailed discussions. A brief description of the overall architecture and the PICO middleware model is presented in Section 2. The model used to represent services and tasks is presented in Section 3. Section 4 presents the technique used to aggregate available services and the mechanism of task resolution. The hierarchical service composition mechanism based on a device overlay formed through the LATCH protocol is presented in Section 5. Section 6 details the capabilities of our composition mechanism in terms of its ability to handle the challenges involved. In Section 7, we lay out our future research directions and some open problems.

2 System Architecture

The overall system architecture in a service-oriented environment is abstractly represented in Fig. 1. Typically, resources that can be potentially used over the network are modeled as services. The services thus created are associated with metadata that describe the service and their usage pattern. Many such services available within the environment are *discovered* using one of the popular service discovery mechanisms [8], [9], [10]. The descriptions of discovered services are stored in a centralized directory, as in the Service Location Protocol (SLP) [9], or in a distributed directory structure such as that in Jini [8]. Users (applications) approach the directory to locate one or more services they need. The directory performs a query on the registered services and returns a matching service if found. In systems such as SpiderNet [2], Anamika [1], and the recent Web services infrastructure [11], where service composition is supported, the users (applications) are accommodated by locating complex combinations of simple services. Such composed services enable users (applications) to reach their goal without having to discover and coordinate among a number of services on their own.

The service composition mechanism presented in this paper employs the event-oriented middleware called PICO [5]. The middleware framework provides a transparent platform for applications and services alike to operate on and cooperate with each other in an efficient manner. The modeling constructs in PICO provide ways to extract the capabilities of resources and allow services to be built around the capabilities. By creating a cooperative structure among the designed services, a mechanism for achieving complex goals is realized.

The *device model* captures the characteristics and features of the available hardware resources. The identified features in the device model are provided as services over the network by creating software entities called *delegents* (intelli*gent dele*gates). Many such delegents can be combined together into a cooperative structure called *communities*. The communities offer a transparent service usage mechanism within the PICO framework.

3 Service Model

The capability of a service-oriented system to successfully operate in the face of the above-stated dynamic conditions depends greatly on the description of the service and the mechanism used to identify suitable matches to facilitate task support. There have been a number of service-oriented systems [12], [11], [13], [8], [4], [9] that allow service descriptions to include domain-specific information, enabling intelligent service selection. Typically, the metadata that represents the service includes descriptions about service capabilities. By employing semantics, formal declarative descriptions are attached to services. Computer programs can use these descriptions to find the appropriate services and use them correctly [14]. A *domain ontology* is used to conceptualize domain knowledge with commonly accepted vocabulary and to provide semantics to service descriptions [15]. By referring to such ontologies, the representation of entities can be interpreted. There have been a number of service composition mechanisms proposed in [16], [17], [18], and [19] for Web services that exploit the semantic information associated with service descriptions. Although much of the work involving semantics is still under research, it is expected that many

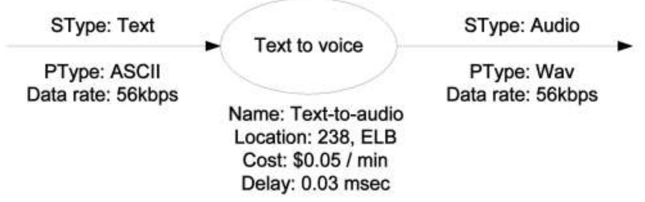

Fig. 2. Graph representation of a service.

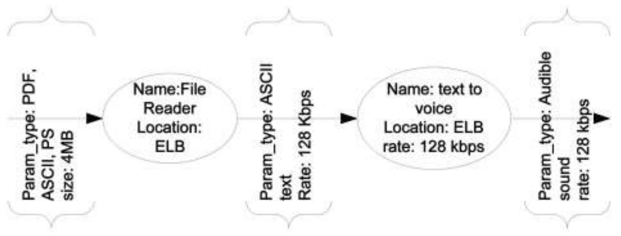
Param_type: PDF, ASCII, PS size: 4MB
Name:File Reader Location: ELB
Param_type: ASCII text Rate: 128 Kbps
Name: text to voice Location: ELB rate: 128 kbps
Param_type: Audible sound rate: 128 kbps

Fig. 3. Graph representing the task of reading out a file.

systems will benefit from semantics-based approaches in the near future. In Section 4, we describe a method that utilizes the semantic description of services.

3.1 Service Representation

Each delegent service is described as a simple graph $G_S = \{V_s, E_s, \mu_s, \xi_s\}$, with V_s being a single element set representing the service itself. The directed edge set E_s represents the inputs and outputs for the service. The vertex attribute function μ_s is responsible for embedding the attributes on the vertex, including the semantic description of the service, location, cost of utilization, promised throughput, advertised delay, and so forth. The edge attributes, including the semantic description of the parameter, the parameter type, the data rates, formats, and so forth, are represented by the function ξ_s.

Fig. 2 shows the representation of a service that performs text-to-voice conversion. The semantic attribute associated with the input is *text*, whereas the syntactic attribute specifies the expected form of text, which is *ASCII* in this case, along with other associated parameters such as the data rate. The descriptions on the vertex and other edges follow the same approach.

3.2 Task Representation

User tasks and applications that need additional support can exploit the presence of services. The model used to represent tasks contains descriptions of required services. Each task is modeled as a directed attributed graph $G_R = \{V_r, E_r, \mu_r, \xi_r\}$, where the vertex set V_r represents the services required to accomplish the task, and the edge set E_r represents the transition among the services. The attributes on the vertices are represented by μ_r, including preferences such as the location, cost, acceptable delay, and so forth. The attributes along the edges are represented by ξ_r, which takes care of attaching the semantic description of data, data formats, data rates, quality requirements, message formats, and so forth. Fig. 3 shows an example of a task graph, representing the task of *reading out* a file to a user.

There has been a number of initiatives [20], [21], [22], [23], [24] focusing on the extraction of request models from context information. In [24], a detailed framework for generating a task based on the target application has also been provided. Based on these existing works, we assume that the graph representing a task is already embedded into the applications by the designers or that the application has the ability to generate the requests based on the current context.

4 Service Aggregation

Typically, all the services available are collected in *directories*, where the metadata associated with services are stored. The directory is either maintained as a centralized structure, in SLP [9], or in a distributed fashion, as in Jini [8]. For simplicity, we first present a centralized directory scheme that supports service composition. The distributed structure of service aggregation and service composition is presented in Section 5.

In a centralized scheme of operation, all the services present within a pervasive computing environment are registered with a directory. At the directory, the service graphs (G_S) are parsed for storage into a two-layered aggregation graph G_P. For each registered service, the first stage of aggregation is based on the semantic parameters associated with the service. The second stage of aggregation is based on the syntactic type of the parameters. Algorithm 1 is the procedure for storing all the registered services into the aggregated graph G_P. Consider the service graphs shown in Fig. 4a. The first service $S_{a\sim b}$ provides a transformation from the semantic type A to the semantic type B. The actual syntactic types associated are $a1$ and $b1$, respectively. Similarly, the second service $S_{b\sim d}$ transforms from the semantic type B to D, whereas the syntactic parameters associated are $b2$ and $d2$, respectively.

Algorithm 1. Service aggregation algorithm

```
1: $InitG_P = null$
2: **for all** $G_S$ **do**
3:     $p_{in} = inParam(G_S)$, $p_{out} = outParam(G_S)$
4:     **if** $p_{in}^{semantic} \notin G_P^1$ **then**
5:         $addNode(G_P^1, p_{in}^{semantic})$
6:     **end if**
7:     **if** $p_{out}^{semantic} \notin G_P^1$ **then**
8:         $addNode(G_P^1, p_{out}^{semantic})$
9:     **end if**
10:    **if** $isNeighbor(p_{in}^{semantic}, p_{out}^{semantic}, G_P^1) = FALSE)$ **then**
11:        $addEdge(p_{in}^{semantic}, p_{out}^{semantic}, G_P^1))$
12:    **end if**
13:    **if** $p_{in} \notin G_P^2$ **then**
14:        $addNode(G_P^2, p_{in})$
15:    **end if**
16:    $addEdge(p_{in}^{semantic}, p_{in})$
17:    **if** $p_{out} \notin G_P$ **then**
18:        $addNode(G_p, p_{out})$
19:    **end if**
20:    $addEdge(p_{out}^{semantic}, p_{out})$
21:    $inNode = getNode(G_P^2, p_{in})$
22:    $outNode = getNode(G_p^2, p_{out})$
23:    $Edge\ e = addEdge(inNode, outNode)$
24:    $attribs(e) = attribs(G_S)$
25: **end for**
```

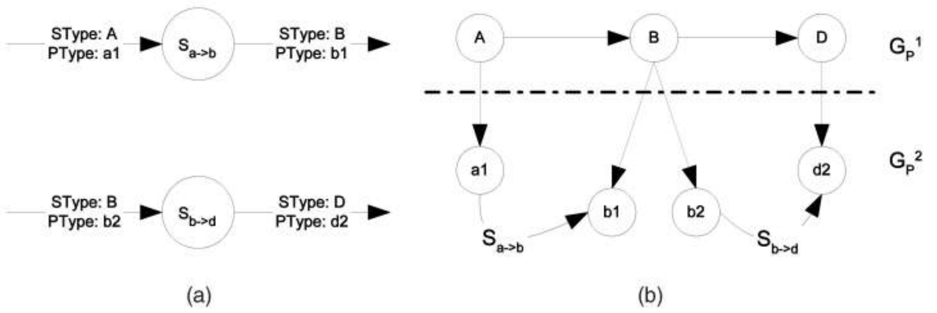

Fig. 4. Process of service aggregation. (a) Service graphs for services $S_{a\sim b}$ and $S_{b\sim d}$. (b) Resulting aggregation graph G_P.

When $S_{a\sim b}$ is registered at the directory, the following holds true:

- nodes A and B are created in the first layer, $G_P^1 \in G_P$,
- a directed edge is created between A and B in G_P^1, indicating that a service that can perform the transformation from A to B exists,
- nodes $a1$ and $b1$ are created in G_P^2,
- directed edges connect nodes from G_P^1 to G_P^2 (A to $a1$ and B to $b1$), indicating that the syntactic parameter $a1$ represents the semantic type A and the parameter $b1$ represents the syntactic type B, and
- a directed edge is created in G_P^2 between $a1$ and $b1$, and the attributes on the edge $a1 \sim b1$ are copied from $\xi_s(S_{a\sim b})$.

A similar process is carried out for the service $S_{b\sim d}$, also resulting in an aggregation as shown in Fig. 4b.

During the process of aggregation, the attributes available through the service definitions are maintained as the attributes on the edges of G_P^2. These attributes are essential to match the requirements of a task to the available service descriptions during composition. By aggregating all the available services into one single graph, we essentially maintain an aggregated knowledge base which can be queried for supporting complex tasks. In Section 4.1, we detail the basic process of extracting a possible composed service, that is, matching a task.

4.1 Task Resolution

A task is submitted to the directory for resolution. At the directory, the components of the task graph are matched against available basic services to generate possible composite services. If there is a direct match between an available service and the requirements specified in G_R, the user task is supported trivially. On the other hand, if such a direct match does not exist, a combination of available services can be utilized to meet the requirements. To accomplish this, we can combine multiple services in such a way that the resulting complex service provides the same functionality as the required service. Two services S_1 and S_2 can be combined with one another to form a complex service S_3 under the following conditions:

$$output(S_1) \cong input(S_2), \tag{4a}$$

$$\xi_s(S_2^{out}) \succ \xi_s(S_1^{in}). \tag{4b}$$

The first condition (4a) ensures that the output *produced* by S_1 can be *consumed* by S_2. The second condition (4b) states that the output attributes of S_1 can be *completely* accepted by S_2.

While (4a) restricts the services that can be combined based on the type of parameter, (4b) imposes the condition that the data produced by one service needs to be consumed by its successor with minimal losses.

The process of coming up with a possible composition for a given task is according to the procedure presented in Algorithm 2. Similar to service aggregation, task resolution is also performed in two steps. In the first step, one or more possible compositions are derived at the semantic level based on the aggregate graph G_P^1. If a composition is possible at the semantic level, the underlying services that can take part in the composed result are identified at the second level of aggregation G_P^2 during the second step. For each vertex in the task graph G_R, the semantic mappings of the input$(A = p_{in}^{semantic})$ and output$(B = p_{out}^{semantic})$ parameters to the nodes in the aggregation G_P^1 are identified. If both A and B are in G_P^1 and if a path exists from A to B, then a possibility of providing the requested service exists. Any path identified in G_P^1 is a composition in the semantic plane. To identify the corresponding services that can take part in the composition, a corresponding path needs to be identified in G_P^2, which is done by identifying the correct syntactic parameters (a and b) related to the request (available as links from G_P^1 to G_P^2) and locating the *shortest* path from a to b in G_P^2, as shown in Fig. 5. The procedure is repeated for all the nodes in G_R. The composition identified is stored in the composed graph G_C, which is returned as the result (for the query G_R). For each of the nodes, K (*predefined*) shortest paths are considered in G_P^1 and a feasible composition in G_P^2 is derived based on one of the K paths. The process of task resolution is depicted in Fig. 5. The resulting composed graph $G_C = \{V_c, E_c, \mu_c, \xi_c\}$ is also a directed attributed graph. G_C is returned as a result of composition. Based on the capabilities of the device where the query originated, the services are directly invoked by the device, or service utilization is achieved with a resourceful device acting as a *proxy*.

Algorithm 2. Request resolution algorithm

```
1: $G_R \leftarrow Request$
2: $G_C \leftarrow null$
3: $K \leftarrow number\ of\ paths\ to\ consider$
```

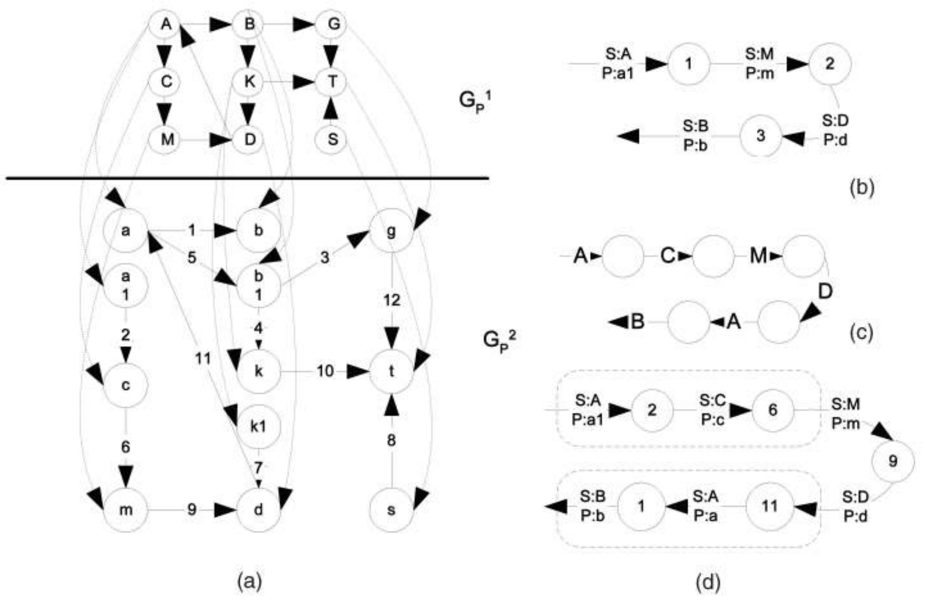

Fig. 5. Task resolution process. (a) Aggregate graph G_P. (b) Task graph G_R. (c) Composed result in semantic graph G_P^1. (d) Resulting composition G_C.

```
4: for all (node $v_r$ in $G_r$) do
5:   $p_{in}^S = inParam^{semantic}(v_r)$
6:   $p_{in}^T = inParam^{type}(v_r)$
7:   $p_{out}^S = outParam^{semantic}(v_r)$
8:   $p_{out}^T = outParam^{type}(v_r)$
9:   if $p_{in}^S$ AND $p_{out}^S$ in $G_p^1$ then
10:    for $k = 0$ to $K$ do
11:      $p_{in}^S \rightsquigarrow p_{out}^S = \text{shortestPath}(p_{in}^S, p_{out}^S, G_P^1)$
12:      if $\exists\, p_{in}^T \rightsquigarrow p_{out}^T \in G_P^2 \cong p_{in}^S \rightsquigarrow p_{out}^S$ then
13:        $G_C = \text{addToResult}(p_{in}^T \rightsquigarrow p_{out}^T)$
14:        BREAK
15:      end if
16:    end for
17:  else
18:    $G_C = \text{addToResult(null)}$
19:  end if
20: end for
21: return $G_C$
```

4.2 Analysis of Composition

By employing a layered approach $(G_P = G_P^1 + G_P^2)$, the search space for composition is reduced to polynomial time from a potentially exponential one. Based on the system domain, the vocabulary contained within the ontology is well known. Each service within the domain is represented with reference to the employed ontology. Considering an ontology with size N, if there are n services available within the system (generally, $n >> N$), all the n services are represented using the N members in the ontology. This results in improved response time during composition. Since the aggregation is maintained separately based on the semantic representations (G_P^1), the possibility of composition for any given task can be determined by finding if there exist paths in G_P^1. The *shortest path* in G_P^1 between any given pair of nodes can be determined in $O(|V_p^1| lg|V_p^1| + |E_p^1|)$ time, where $|V_p^1| \leq N$. Since we consider K shortest paths for each node in the task graph G_R, the resulting complexity for composition in the semantic plane for each node in G_R is $O(|V_p^1| lg|V_p^1| + |E_p^1| + K)$. For each of the K paths identified, a *valid* path needs to be determined in G_P^2, which can be performed in $O(K|V_C|)$ since the resolution is performed just by looking at the neighbors in G_P^2.

During the process of resolution, a candidate service in G_P^2 has to accommodate a number of requirements specified by the task. Requirements such as the permissible delay, allowable loss, cost of utilization, location, and so forth are some example constraints that need to be considered during composition. These requirements are specified within the task through the attributes on the nodes $(V_r \in G_R)$ as a constraint set $C = \{c^1, c^2, \ldots, c^n\}$. These parameters are retained as the edge attributes within G_P^2. While deriving the composite solution for the task, set C is treated as weights along the edges. The computation of the shortest path has to consider these multiple constraints along the edges as weights, as mentioned in (4b). These conditions can be treated as *weights* along the edges, thus turning the problem of finding a path between two nodes into a *multiconstrained path selection (MCPS)* problem, which is a well-known NP-complete problem [25]. The MCPS problem has been well studied in the context of quality-of-service (QoS)-aware routing [25], [26], [27], [28], with proposed solutions to achieve polynomial time bounds for restricted versions of the problem. In our approach, to achieve polynomial time performance for service composition, we limit the number of services used to compose each requested service within the task.

4.3 Performance Comparison

To analyze the performance of our graph-based service composition scheme, we conducted simulation experiments and compared our scheme against a generalized *discover + match* scheme. To measure the composition efficiency of our scheme, we have built a centralized directory, where all

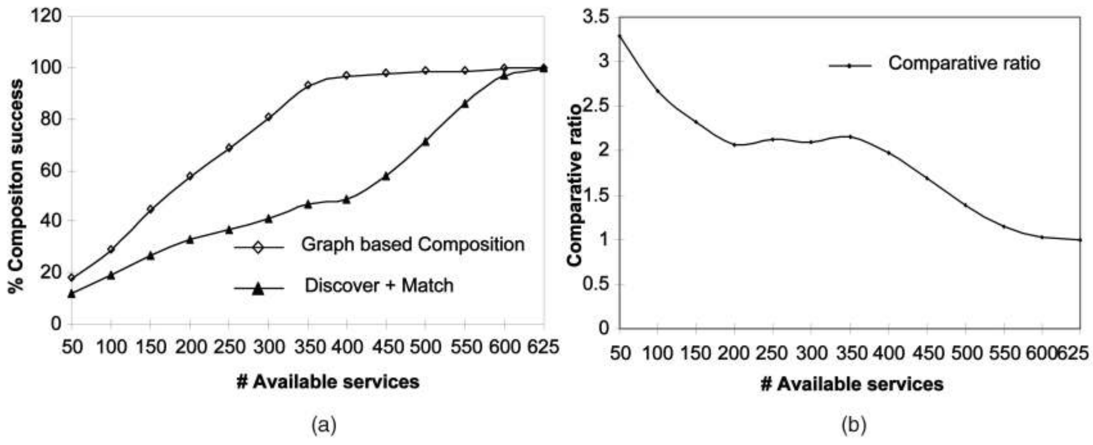

Fig. 6. (a) Composition success rate comparison. (b) Comparative ratio for composition.

the available services register. When a service registers with the directory, a graph (G_S) representing the service is provided as a part of the advertisement. At the directory, an aggregated graph (G_P) of all the registered services, which is generated according to Algorithm 1, is maintained. When a user device has a task to be accomplished, the task request is submitted to the directory in the form of a graph (G_R). The tasks are resolved according to the procedure detailed in Algorithm 2.

The $discover + match$ technique for service composition is implemented as follows. Similar to the previous scheme, all the available services register at a directory, and the directory maintains a list of all the available services. The user task is represented as a task graph (G_R) and is sent to the directory for resolution. At the directory, the services specified within the task graph (denoted by the vertex set V_r) are matched against the services that are registered to identify available matches.

For the purpose of simulation and to demonstrate the power of our scheme, we have considered services that perform transformation between two alphabets. A service s_i is represented as $a \rightsquigarrow b$ if s_i accepts an input of type a and produces an output of type b. By considering all possible combinations of alphabet transformations $(a \rightsquigarrow b, a \rightsquigarrow c, \dots, a \rightsquigarrow z, \dots, z \rightsquigarrow y)$, there are 625 unique services. Within the simulation, we control the service density by varying the number of registered services at any given time. The composite service requests also contain a combination of letters, each of which represents a service. The results shown below are averaged over 10 simulation runs with varying service densities. For each run, there are 25 requests. The plot shown in Fig. 6a is a comparison of the composition efficiency of our graph-based composition approach against the $discover + match$ scheme. The comparative ratio of composition (C_{CR}) is also shown in the plot in Fig. 6b. As can be seen from the results, our graph-based service composition mechanism can perform better due to the flexible approach to service composition. The comparative ratio of composition is given by the following formula:

$$C_{CR} = \frac{\eta_{GSC}}{\eta_{DM}}, \tag{4c}$$

where η_{GSC} is the number of successful compositions using our graph-based composition scheme and η_{DM} is the number of successful compositions based on the traditional $discover + match$ approach.

5 Hierarchical Service Overlay for Composition

The service composition mechanism presented in Section 4 assumes the presence of a directory, where information about all available services is *collected* and queries about services are *resolved*. Such an architecture can typically be mapped onto *managed environments* with well-defined nodes in the network that can act as directories. Depending on the number of nodes involved, the directory itself can either be centralized or distributed. The presence of a managed infrastructure enables designers to identify and deploy specific nodes that can act as directories. Similarly, other essential operations such as name resolution [29], proxy operation [30], and so forth can be deployed on dedicated (possibly distributed) nodes within the network. However, within pervasive computing environments, it is not always possible to assume the support of an infrastructure. User tasks need to be supported even in completely *ad hoc* environments with no infrastructure support. Also, due to the diverse nature of the involved devices (and, hence, the resources), it is a challenging task to create a transparent platform for executing user tasks.

To achieve transparent operational support, we employ a device classification mechanism in our approach. Resources with relatively higher degrees of availability and with minimal resource restrictions such as the servers, clusters, and so forth can be classified into the highest end of the spectrum, which is a level-3 device. Under level 2, user devices such as laptops and PDAs are classified. The mobility associated with such resources differentiates them from level-3 devices. Resources that can host the middleware and accommodate native support for delegents to execute but cannot act as proxies for other resource-poor devices are classified into level 1. Level-0 devices are resources with no native support for additional configuration. In such cases, a proxy is assigned to a level-0 device to make its features available as services in

TABLE 1
Device Classification Chart

LEVEL (α)	Middleware Version	Features	Examples
0	$NONE$	Features exported through delegents on Level-2 or Level-3 devices, No native personalization support	sensors, legacy printers
1	Minimum	Supports personalization, cannot be a proxy, possibly mobile	Cell phone, Mote sensors, smart printer
2	Complete	Supports personalization, Can act as a proxy, possibly mobile, Resource rich	Laptop, PDA
3	Complete	Supports personalization, typically a node in the infrastructure, can act as a proxy, not mobile, Resource rich	Servers, PCs, clusters, Grids

the environment. In Table 1, we present a summary of the resource classifications.

The classification of resources into one of the above levels is performed at the time of configuring the resources and installation of the middleware. The level of the device α is made available through the device specification document to be used at runtime.

The device classification derived according to the guidelines in Table 1 is useful in deriving an organized hierarchical structure in the pervasive environment. The basic principle of organization is for the resource-poor devices to *depend* on their resource-rich counterparts for their operations. The computing resources that are a part of the infrastructure that have relatively lower constraints in terms of resource restrictions are typically mapped into a higher level. Resources such as PCs and servers can support essential operations such as service discovery, service composition, and proxy operation for their resource-poor counterparts. This results in a hierarchical relationship as shown in Fig. 7. The organization of resources into a service overlay is facilitated by a process we refer to as *latching*. The basic principle of latching is that a device with lower resource availability *latches* itself to another device with higher resource availability. The process is repeated until a level-3 device is reached. The result of the latch process is a hierarchy with all the level-3 devices being mapped at the highest level in the tree.

The process of *latching* is detailed as a protocol, and the timing diagram of the *LATCH* protocol is depicted in Fig. 8.

5.1 The LATCH Process

When a device A enters the environment, it attempts to join the environment by broadcasting a *LATCH_HELO* message as an advertisement, along with the information about its device level ($\alpha(A)$). Another device B that is already in the environment upon receiving the message from A inspects $\alpha(A)$ and compares it with its own α. If $\alpha(A) > \alpha(B)$, then B sends a *LATCH_REQ* message to A, requesting A to be B's parent. If $\alpha(A) < \alpha(B)$, then B sends a *LATCH_INVITE* message to A, inviting A to be B's child. If $\alpha(A) = \alpha(B)$, then B sends a *LATCH_SIBLING* message to A and adds A as a sibling. When A receives a response from B, it inspects the response and adds B as either a parent, child, or sibling based on $\alpha(A)$. A *LATCH_ACK* acknowledgment is sent from the parent to the child once the child is registered at the parent. When registering itself with a parent, a device A sends the local parameter graph G_P if it already has some children. Otherwise, A sends service graphs (G_S) of all the services it offers.

The resulting hierarchical overlay is maintained with periodic *LATCH_HELO* messages, along which the current status of the child nodes is relayed to the parent. The parent inspects the *LATCH_HELO* message and keeps the child's status as *alive* and also updates the attributes of the services provided by the child based on the information that is piggybacked along with the *LATCH_HELO* message.

With the hierarchical structure created among the devices, those at a lower level will *rely* on their higher level counterparts (parents) for essential operations such as service discovery, composition, proxy operation, and so forth. By maintaining such a hierarchy, the differences in the capabilities of different devices involved can be made transparent to the applications that operate within the created environment.

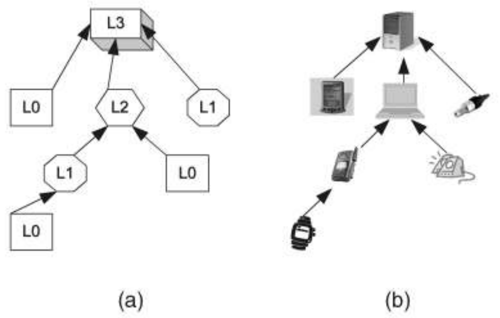

Fig. 7. Hierarchical organization of devices. (a) Device hierarchy based on α. (b) An example hierarchy.

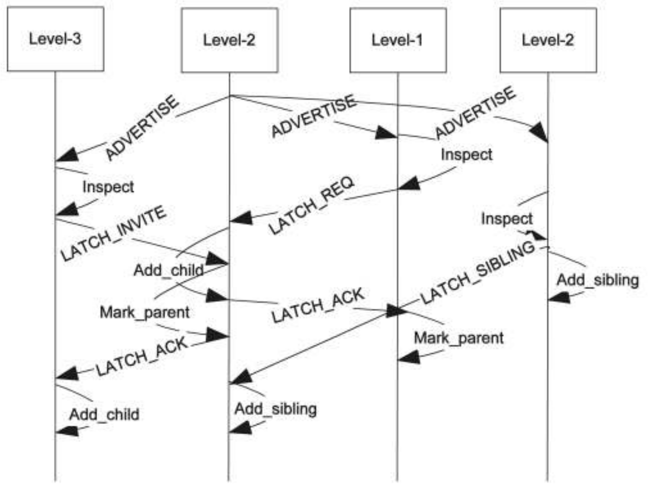

Fig. 8. Timing diagram of the LATCH process.

5.2 Hierarchical Aggregation of Services

Due to the dynamic nature of resources involved in pervasive computing environments, the structure of the hierarchy can change frequently. The change in the structure can be largely attributed to either a node moving from one location to another or node unavailability due to resource limitations. In either case, the service overlay will be modified to reflect the current status of resources.

When a new device A joins the hierarchy, the parent B will collect the graphs for all the services provided by A and updates the local service aggregation G_P^B. Algorithm 3 details the procedure used to admit a new node into the hierarchy. The updated information is sent up the hierarchy to B's parents with a *LATCH_ADD* message. Similarly, when a parent B observes that its child A is unavailable (through the missing *LATCH_HELO* message), it deletes all the services associated with A and sends the update message *LATCH_REM* to its parents along with the information about services offered through A.

Algorithm 3. Hierarchical service aggregation algorithm

```
$G_P^b \leftarrow local\ aggregation$
$G_P^a \leftarrow child's\ aggregation$
set $visited = false$ for all $v_p^i \in G_P^a$
**for all** $(v_p^i)$ in $G_P^a$ **do**
  **if** $visited(v_p^i) = false$ **then**
    addNode($v_p^i$)
  **end if**
**end for**
Procedure addNode($p_1$)
**if** ($p_1 \notin V_P^b$) **then**
  add $p_1$ to $G_P^b$
**end if**
**for all** $neighbor(p_1) = p_2$ **do**
  addNode($p_2$) {$neighbor(p_1)$ is connected by a directed
  edge originating at $p_1$}
**end for**
addEdge($p_1, p_2$) {Copy all edges between $p_1$ and $p_2$ in $G_P^a$
and their respective attributes}
Also add an attribute $reachableFrom$ to $p_1$, $p_2$ and set its
value to $a$.
```

The algorithm considers each vertex p_i in the child's aggregation G_P^A and inspects to see if it is already a vertex in the local aggregation G_P^B. If p_i is not present in G_P^B, then p_i is added to G_P^B. Each edge originating at p_i (which represents an individual service) is then incrementally added to the local aggregation upon inspecting all the nodes in G_P^A. Although the service information is replicated all along the hierarchy, the fact that the nodes higher up in the hierarchy are more capable than those at lower levels substantiates the design.

When a device A leaves a parent B, either due to A being mobile or due to resource limitations, it may notify B of its departure. Otherwise, B decides that A is no longer available due to the missing periodic *LATCH_HELO* messages. Once A departs, all the services that were reachable through A should no longer be contained within B's aggregation. For this, B examines each edge $e_p^i \in G_P^B$ to see if $reachableFrom(e_p^i) = A$ and deletes all edges that were reachable through A.

Based on the hierarchy formed, we make the following definitions:

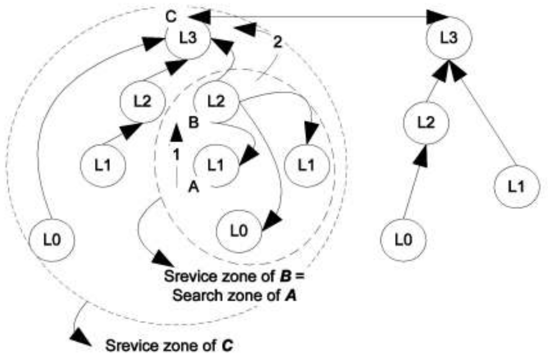

Fig. 9. Illustration of service zone and search zone in the hierarchy.

- *Service zone*. The service zone of a device D includes all the children of D within the hierarchy and itself. The service zone of a device defines the zone for which d is responsible with respect to service-related operations such as discovery, composition, and failure recovery. All the children of D *depend* on D to provide service-related support. If a particular task can be completely supported within the service zone of a device D and not within the service zone of any of its children, then D will *manage* the composed service.
- *Search zone*. The search zone of a device D is its own service zone if $\alpha(D) > 1$. If $\alpha(D) < 1$, then the search zone of D is the service zone of $parent(D)$. When D has a task to be composed, the aggregation in the local search zone is first inspected. If any of the required services for the task cannot be met, the search zone is then expanded to the search zone of $parent(D)$, and so on.

Fig. 9 illustrates the service and search zones for an example hierarchy.

5.3 Hierarchical Service Composition

When a particular task needs to be accomplished, based on the α of the device generating the request, the task resolution process begins either at the same device or at its immediate parent. Assuming that D generates a task request ($G_R^D = \{V_r^D, E_r^D, \mu_r^D, \xi_r^D\}$) and $\alpha(D) = 1$, the task will be *forwarded* to D's parent, A, for resolution. Upon receiving the request, A will run the request resolution algorithm presented in Algorithm 2 and attempt to generate a possible composition. Suppose A can compose a set of nodes $v_r^1 = \{v_i^D\}_{i=1}^k \in V_r^D$. It sets the composition for G_R^D as G_C^D, populates G_C^D with the successful partial compositions, and forwards only those nodes $((V_R^D)' = V_R^D - v_r^1)$ to its own parent for resolution. The process repeats until the highest node (say, B and $\alpha(B) = 3$) in the hierarchy is reached. At B, all the available services within the environment are inspected upon consultation among the other devices with $\alpha = 3$ to arrive at a possible composition.

6 Analysis of Dynamic Service Composition

The hierarchical service overlay formed among the resources available within a pervasive computing environment is used in supporting essential service-related operations such as service discovery, composition, and service management.

Based on the hierarchical operation, we can observe support for the following service-related factors.

6.1 Locality of Service

Locality of service refers to the requirement that the distance between the requesting entity and the services used in composition will be as small as possible. Locality of service is an important factor when the tasks include services which interact directly with users. Also, since each of the individual services can possibly reside on different nodes within the network, there will be an inherent network-related *cost* involved in combining them. To minimize such a cost of operation in a composed service, all involved services need to be located as close to each other as possible. The resulting hierarchical organization of devices also contains nodes that are a part of the infrastructure. In such devices, it is possible to embed additional location information that can contribute in refining the locality of services [12]. Consider a hierarchy similar to the one shown in Fig. 9. If a request for composition arises at a device A in the hierarchy, as shown in Fig. 9, an attempt for composing all the required services is first done at the device B, which is the parent of A over the service zone of B. Since the service zone of B includes all the services around B, any successful composition within B's service zone is guaranteed to be the closest *possible* composition for the request.

Since the hierarchy is formed based on the *network* distance between devices, the *locality* of services refers to the network locality. Many of the devices in a pervasive computing environment are typically equipped with short-range wireless interfaces for communication. In such devices, the network locality can also reflect the physical proximity of devices. Moreover, as a request is forwarded through the hierarchy, more powerful devices residing higher in the hierarchy can contribute to the refinement of location information.

6.2 Quality-Aware Composition

Typical user requests contain a number of parameters that need to be satisfied while the composition is being carried out. The parameters can be user preferences such as the desired location, a particular service provider, and so forth and also classical QoS parameters such as required bandwidth, permissible delay, acceptable cost, and so forth. In [31], an effective mechanism for attaching quality parameters to resources has been presented. We utilize this scheme to embed quality-related information to services. Essentially, the quality parameters associated with each service are embedded as attributes of the service. The node attribute function μ_s of each service graph is responsible for attaching and maintaining the quality attributes $q^s = \{Q_{in}, Q_{out}\}$ for each service. The quality requirements of a request are also specified as node attributes within the request graph $G_R = \{V_r, E_r, \mu_r, \xi_r\}$. Therefore, $\forall v_i \in V_r, \ni \{q^s_i\}^n_{i=1}$.

The result of a composition for $v_i \in V_r$ is $S_C = \{s_a\}^k_{i=a}$, that is, all the services that form the path $(p_{in} \rightsquigarrow p_{out})$ in G_P.

While computing the path S_C, we need to make sure that $q^s(S_C) \geq q^s(v_i)$ and $q^s(S_C) = \sum^k_{a=1} q^s_a$.

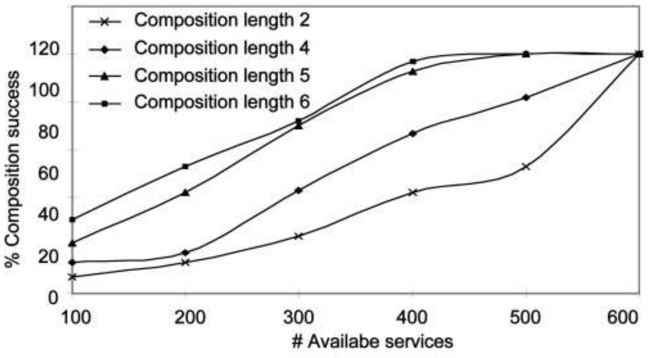

Fig. 10. Composition success ratio for different lengths of composition.

The composed service can be ensured to meet the quality requirements of the request by using the attributes on the edges of G_P to compute the *shortest* path $(p_{in} \rightsquigarrow p_{out})$ in G_P.

As mentioned in Section 4.2, the problem of finding a path in a multiconstrained graph is NP-complete. There have been a number of schemes [25], [26], [27], [28] proposed to achieve polynomial time bounds for the problem. As proposed in [27], by maintaining an average cost function reflecting values of all the specified requirements, the problem can be reduced to simple path selection problem. However, due to the varied nature of metrics and dynamically changing values, it is hard to maintain such a cost parameter within the pervasive computing domain. We employ a limited version of MCPS by limiting the length of composition, which is essentially the number of services that can be used to compose a single service. Although this limits the number of possible solutions, it provides an acceptable bound on the delay during service composition. Fig. 10 shows the effect of limiting the length of the composition to a predefined number. If the service density is higher, even with a lower value of composition length, a successful composition can be achieved. However, at lower service densities, it might be necessary to allow higher composition lengths for better composition.

6.3 User and Resource Mobility

One of the major challenges in pervasive computing applications is the issue of mobility. In any pervasive computing environment, once the initial composition is identified and a service session is established, the mobility of the user can change the composed solution. For example, the user might walk out of a room where a video stream is being displayed or move to a more capable terminal. Also, there is a possibility that one of the services currently being used in the composition becomes unavailable. For example, a handheld device being used to present the video stream might no longer be available due to power limitations. In such situations, the challenge is to reconfigure the session under progress as quickly as possible by considering the current resource availability around the user. With the hierarchical service overlay, it is possible to ensure that the request can be recomputed with minimal interruption of the session under progress.

The effect of user mobility while a service session is in progress can lead to a complete recomposition of the service. Typically, however, it is sufficient to identify a new

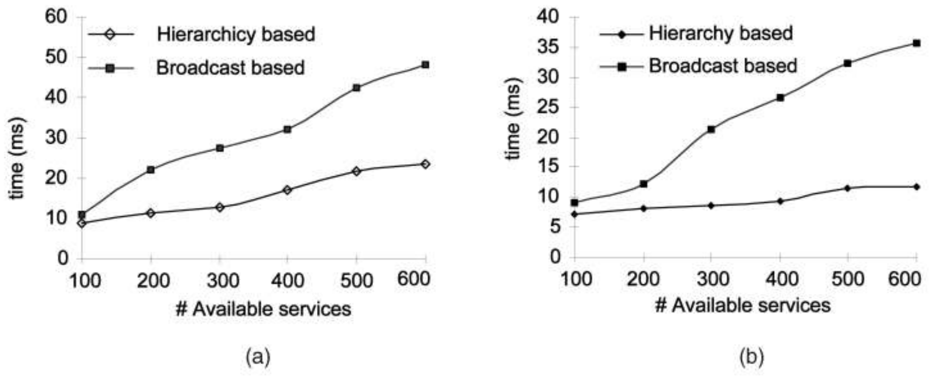

Fig. 11. Composition reconfiguration time (a) due to user mobility and (b) due to resource mobility.

interactive service closer to the user such as a display unit, a speaker, and so forth. When a session is in progress, there are a set of services $\{s_i^c\}_{i=1}^n$ that are initially a part of the composition. When the user moves to a new location, the first task is to identify a suitable service s_k that can directly interact with the user. Based on the service selected to interact with the user, the composition needs to be changed accordingly. Through simulations, we have observed that, in a majority of the cases, it is sufficient to only recompose a part of the original composition.

When a resource being used in a composition is mobile or is no longer available due to resource restrictions, we need to identify an alternative service to fulfill the part that was being played by the missing service. Therefore, when a resource either moves or is no longer available, all the services that were available through that particular device become unavailable. For a service session in progress, which uses one or more services of the mobile device, it is now necessary to recompute a portion of the composition. Recall that the result of a successful composition S^C is a set of services $\{s_i\}_{i=1}^n$, where a subset of the composed services is used in matching the needs of each node in the request graph G_R. So, $\forall v_k \in V_r, \ni \{s_i\}_{i=1}^k$. If a service $s_j \in \{s_i\}_{i=1}^n$ present on the device that moved out of a particular service zone was a part of an ongoing composed session (that is, a node in the request G_R), only that part of the request needs to be recomposed. Based on the hierarchy, a new composition can be generated to meet the requirements of the affected node $v_k \in V_R$ in at most two search zone expansions and another lookup on G_P among the level-3 devices.

Since the maximum depth of the hierarchy is three, it is sufficient to expand the search zone twice to reach the Level-3 device and, if required, another lookup can be made among the Level-3 devices. Fig. 11a shows the comparison of reconfiguration times between the discover-and-match scheme and the hierarchical composition scheme due to user mobility. Fig. 11b shows the comparison in case of a mobile resource. Since, at each level of the hierarchy, the available services are already aggregated, the reconfiguration is a matter of a new lookup in the graph G_P. The simulation results show that the hierarchical scheme outperforms schemes based on discover-and-match techniques.

7 Conclusions and Future Work

The presence of a number of personal devices with users and a variety of communication mechanisms make it possible to realize a pervasive computing environment where essential support to user tasks can be provided. In this paper, we have described a model to build service-oriented environments using the PICO middleware framework. The features available through a number of resources can be offered as services by designing and deploying software entities called *delegents*. Typical user tasks require support from a number of services. Service composition techniques offer advanced support to user tasks by identifying all the required services for a particular task and achieving coordination among the identified services. The service composition mechanism presented in this paper performs beyond the traditional $discover + match$ approaches by *constructing* possible compositions based on their semantic and syntactic descriptions. We have also presented a hierarchical service overlay mechanism based on the proposed *LATCH* protocol. The hierarchical scheme of aggregation exploits the presence of heterogeneity through device cooperation. Devices with higher resources assist those with restricted resources in accomplishing service-related tasks such as discovery, composition, and execution. Through simulation results, we have shown the performance improvement achieved by the proposed scheme as compared to the traditional $discover + match$ schemes.

The proposed work is aimed at exploiting the features of devices in the environment to provide support to user tasks. The current framework assumes the availability of information such as the device capabilities made available at design time. In reality, due to the dynamic nature of devices and varying workload, the capabilities of each device can vary over time. It is thus necessary to dynamically capture information regarding the state of a device while the device is operational. Such a dynamic mechanism will ensure uniform resource consumption, timely support, and fairness in resource utilization. The changing states of resources result in a dynamic hierarchical overlay that accurately reflects the device state. The development of a dynamic scheme to capture the device state will also lead to the possibility of ensuring that a service session in progress is supported in an unobtrusive manner by monitoring the involved services and the resources they execute on. We are currently working toward developing such a model that would capture information about the available resources in a dynamic fashion.

Acknowledgments

The work presented in this paper was completed under US National Science Foundation grants ANI 0129682 and IIS 0326505 and was a part of Swaroop Kalaspur's PhD dissertation while at the University of Texas at Arlington. The authors would like to thank the anonymous reviewers for their constructive comments to improve the quality of this paper.

References

[1] D. Chakraborty, F. Perich, A. Joshi, T.W. Finin, and Y. Yesha, "A Reactive Service Composition Architecture for Pervasive Computing Environments," *Proc. Int'l Federation for Information Processing (IFIP) TC6/WG6.8 Working Conf. Personal Wireless Comm. (PWC '02),* pp. 53-62, 2002.

[2] X. Gu, K. Nahrstedt, and B. Yu, "Spidernet: An Integrated Peer-to-Peer Service Composition Framework," *Proc. 13th IEEE Int'l Symp. High Performance Distributed Computing (HPDC '04),* pp. 110-119, 2004.

[3] J. Robinson, I. Wakeman, and T. Owen, "Scooby: Middleware for Service Composition in Pervasive Computing," *Proc. Second Workshop Middleware for Pervasive and Ad Hoc Computing,* pp. 161-166, 2004.

[4] S. Helal, N. Desai, V. Verma, and C. Lee, "Konark—A Service Discovery and Delivery Protocol for Ad Hoc Network," *Proc. Wireless Comm. and Networking (WCNC '03),* vol. 3, pp. 2107-2113, Mar. 2003.

[5] M. Kumar, B.A. Shirazi, S.K. Das, M. Singhal, B. Sung, and D. Levine, "Pervasive Information Communities Organization PICO: A Middleware Framework for Pervasive Computing," *IEEE Pervasive Computing,* vol. 2, pp. 72-79, 2003.

[6] S. Kalasapur, M. Kumar, and B. Shirazi, "Personalized Service Composition for Ubiquitous Multimedia Delivery," *Proc. Sixth IEEE Int'l Symp. World of Wireless Mobile and Multimedia Networks (WoWMoM '05),* pp. 258-263, 2005.

[7] S. Kalasapur, M. Kumar, and B. Shirazi, "Seamless Service Composition (SESCO) in Pervasive Environments," *Proc. First ACM Int'l Workshop Multimedia Service Composition (MSC '05),* pp. 11-20, 2005.

[8] J. Waldo, "The Jini Architecture for Network-Centric Computing," *Comm. ACM,* vol. 42, no. 7, pp. 76-82, 1999.

[9] E. Guttman, "Service Location Protocol: Automatic Service Discovery of IP Network Services," *IEEE Internet Computing,* vol. 3, pp. 71-80, July-Aug. 1999.

[10] B.A. Miller, T. Nixon, C. Tai, and M. Wood, "Home Networking with Universal Plug and Play," *IEEE Comm. Magazine,* vol. 39, pp. 104-109, Dec. 2001.

[11] P. Rompothong and T. Senivongse, "A Query Federation of UDDI Registries," *Proc. First Int'l Symp. Information and Comm. Technologies (ISICT '03),* pp. 561-566, 2003.

[12] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The Design and Implementation of an Intentional Naming System," *Proc. Symp. Operating Systems Principles,* pp. 186-201, 1999.

[13] D. Booth, "Web Services Description Language (WSDL)," w3C working draft, http://www.w3.org/TR/wsdl20-primer/, Aug. 2005,

[14] R. Filman, "Semantic Services," *IEEE Internet Computing,* vol. 7, pp. 4-6, July-Aug. 2003.

[15] D. Bianchini, V.D. Antonellis, and M. Melchiori, "An Ontology-Based Architecture for Service Discovery and Advice System," *Proc. 16th Int'l Workshop Database and Expert Systems Applications,* pp. 22-26, Aug. 2005.

[16] B. Medjahed, A. Bouguettaya, and A.K. Elmagarmid, "Composing Web Services on the Semantic Web," *VLDB J.,* vol. 12, no. 4, pp. 333-351, 2003.

[17] J. Lui, C. Fan, and N. Gu, "Web Services Automatic Composition with Minimal Execution Price," *Proc. 2005 IEEE Int'l Conf. Web Services (ICWS '05),* pp. 11-15, July 2005.

[18] Z. Lu, P. Hyland, A.K. Ghose, and Y. Guan, "Using Assumptions in Service Composition Context," *Proc. 2006 Int'l Workshop Service-Oriented Software Eng. (SOSE '06),* pp. 19-25, 2006.

[19] K. Fujii and T. Suda, "Dynamic Service Composition Using Santic Information," *Proc. Second Int'l Conf. Service Oriented Computing (ICSOC '04),* pp. 39-48, 2004.

[20] R. Dick, D. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," *Proc. Sixth Int'l Workshop Hardware/Software Codesign (CODES/CASHE '98),* pp. 97-101, Mar. 1998.

[21] M. Cosnard and M. Loi, "Automatic Task Graph Generation Techniques," *Proc. 28th Hawaii Int'l Conf. System Sciences,* vol. 2, pp. 113-122, Jan. 1995.

[22] D. Wichadakul, X. Gu, and K. Nahrstedt, "A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications," *Proc. 10th ACM Int'l Conf. Multimedia (Multimedia '02),* pp. 631-640, 2002.

[23] S. Ponnekanti and A. Fox, "'Sword': A Developer Toolkit for Building Composite Web Services," *Proc. 11th World Wide Web Conf. (Web Eng. Track),* May 2002.

[24] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. Campbell, and M. Mickunas, "Olympus: A High-Level Programming Model for Pervasive Computing Environments," *Proc. IEEE Int'l Conf. Pervasive Computing and Comm. (PerCom '05),* pp. 7-16, 2005.

[25] T. Korkmaz, M. Krunz, and S. Tragoudas, "An Efficient Algorithm for Finding a Path Subject to Two Additive Constraints," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '00),* pp. 318-327, 2000.

[26] T. Korkmaz and M. Krunz, "Multi-Constrained Optimal Path Selection," *Proc. INFOCOM,* pp. 22-26, Apr. 2001.

[27] W. Xiao, B.H. Soong, and Y.L. Guan, "Evaluation of Heuristic Path Selection Algorithms for Multi-Constrained Qos Routing," *Proc. IEEE Int'l Conf. Networking, Sensing and Control,* pp. 112-116, Mar. 2004.

[28] G. Liu and K.G. Ramakrishnan, "A*Prune: An Algorithm for Finding k Shortest Paths Subject to Multiple Constraints," *Proc. INFOCOM,* pp. 743-749, 2001.

[29] S. Vinosky, "Corba: Integrating Diverse Applications within Distributed Heterogeneous Environments," *IEEE Comm. Magazine,* pp. 46-55, Feb. 1997.

[30] X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt, "Adaptive Offloading for Pervasive Computing," *IEEE Pervasive Computing,* vol. 3, no. 3, pp. 66-73, 2004.

[31] K. Nahrstedt and W. Balke, "A Taxonomy for Multimedia Service Composition," *Proc. 12th Ann. ACM Int'l Conf. Multimedia (Multimedia '04),* pp. 88-95, 2004.

Swaroop Kalasapur received the BE degree in instrumentation and electronics engineering from Bangalore Univeristy and the MS (2001) and PhD (2006) degrees in computer science and engineering from the University of Texas at Arlington. He is currently a research staff member at Samsung Research. His research interests include pervasive computing, distributed computing, mobile computing, and Web-related technologies. He has various publications in conferences and workshops in the field of pervasive computing. He has contributed to various journals, conferences, and workshops in the capacity of a reviewer. He is a member of the IEEE.

Mohan Kumar received the BE (1982) degree from Bangalore University in India and the MTech (1985) and PhD (1992) degrees from the Indian Institute of Science. He is a professor in computer science and engineering at the University of Texas at Arlington. Prior to joining the University of Texas at Arlington in 2001, he held faculty positions at the Curtin University of Technology, Perth, Australia (1992-2000), the Indian Institute of Science (1986-1992), and Bangalore University (1985-1986). His current research interests are in pervasive computing, wireless networks and mobility, active networks, mobile agents, and distributed computing. He has published more than 125 articles in refereed journals and conference proceedings and supervised several doctoral dissertations and master's theses in the above areas. He is one of the founding editors of the *Pervasive and Mobile Computing Journal* and is on the editorial board of *The Computer Journal*. He has guest coedited special issues of several leading international journals. He is a senior member of the IEEE. He is a cofounder of the IEEE International Conference on Pervasive Computing and Communications (PerCom), where he served as program chair (2003) and general chair (2005). He has also served on the technical program committees of numerous international conferences/workshops. Recently, he has developed or codeveloped algorithms/methods for service composition in pervasive environments, information acquisition, dissemination and fusion in pervasive and sensor systems, caching and prefetching in mobile, distributed, pervasive, and P2P systems, and active-network-based routing and multicasting in wireless networks.

Behrooz A. Shirazi is the Huie-Rogers chair professor and the director of the School of Electrical Engineering and Computer Science at Washington State University (WSU). Prior to joining WSU in 2005, he was on the Faculty of Computer Science and Engineering at the University of Texas at Arlington and served as the department chair from 1999 to 2005. He has conducted research in the areas of pervasive computing, software tools, distributed real-time systems, and parallel and distributed systems over the past 18 years. He is currently serving as the editor-in-chief for special issues of the *Pervasive and Mobile Computing (PMC) Journal* and has served on the editorial boards of the *IEEE Transactions on Computers* and the *Journal of Parallel and Distributed Computing* in the past. He is a cofounder of the IEEE International Conference on Pervasive Computing and Communications (PerCom). He has served on the program committee of many international conferences. He has received numerous teaching and research awards and has served as an IEEE Distinguished Visitor (1993-1996) as well as an ACM Lecturer (1993-1997). He is a member of the IEEE.