Dynamic QoS-Aware Multimedia Service Configuration in Ubiquitous Computing Environments *

Xiaohui Gu, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
Email:{xgu},{klara}@ cs.uiuc.edu

Abstract

Ubiquitous computing promotes the proliferation of various stationary, embedded and mobile devices interconnected by heterogeneous networks. It leads to a highly dynamic distributed system with many devices and services coming and going frequently. Many emerging distributed multimedia applications are being deployed in such a computing environment. In order to make the experience for a user truly seamless and to provide soft performance guarantees, we must meet the following challenges: (1) users should be able to perform tasks continuously, despite changes of resources, devices and locations; (2) users should be able to efficiently utilize all accessible resources within runtime environments to receive the best possible Quality-of-Service (QoS). In this paper, we propose an integrated QoS-aware service configuration model to address the above problems. The configuration model includes two tiers: (1) service composition tier, which is responsible for choosing and composing current available service components appropriately and coordinating arbitrary interactions between them to achieve the user's objectives; and (2) service distribution tier, which is responsible for dividing an application into several partitions and distributing them to different available devices appropriately. Our initial experimental results based on both prototype and simulations show the soundness of our model and algorithms.

1. Introduction

Ubiquitous computing [15] has extended the computer system to the whole physical space and leaded to a more *dynamic* distributed system than ever before, with many devices and services coming and going frequently. Moreover, nowadays a single user often possesses multiple heterogeneous devices ranging from desktop, laptop, to PDA. The user may use any of those devices as the portal, with the help of other personal devices and/or proxy hosts, to perform tasks. Many quality sensitive distributed multimedia applications, such as *video-on-demand* and *visual tracking*, are being deployed in such a ubiquitous computing environment. Thus, a big challenging problem is to *provide a dynamic service configuration model to enable seamless delivery of distributed multimedia applications, in the ubiquitous computing environment, with best possible Quality-of-Service (QoS) guarantees.*

The problem of service configuration has been addressed in different research work [8, 12, 17]. However, most of the proposed approaches do not meet the expectations of the user community and fall short of the potential for ubiquitous computing. We identify two key problems as follows:

- The first problem is brought by the inflexible way service components are composed to form a distributed application delivery. Dynamic insertion of mediating services [12] provides certain adaptability, but also unnecessary overhead when the server or client service itself can be dynamically changed. Specifying a polymorphic distributed application [17] using multiple service paths provides a more flexible solution. However, those service paths are often predefined and fixed. They lacks the adaptability to accommodate changes that are unknown at design time. Moreover, all of the above approaches can only handle linear service compositions which have serious limitations to support complex distributed multimedia applications.

- Second, although ubiquitous computing environments provide more abundant resources than ever before, most of them are under-utilized. Putting all mediat-

*This work was supported by the NASA grant under contract number NASA NAG 2-1406, NSF grant under contract number 9870736, 9970139, and EIA 99-72884EQ. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, NASA or U.S. Government.

ing services in a single host makes them easy to maintain, but also vulnerable to malicious attacks. It is highly desirable that those services can dynamically bind available resources in the runtime environment. On the other hand, a single user often possesses multiple heterogeneous devices and many proxy hosts are also available to users everywhere (e.g. office, conference room, hotel). Aggregating those resources efficiently can definitely help to overcome resource limitations of mobile devices and provide better QoS for users. However, dividing a distributed application delivery appropriately and binding services to suitable available devices is a challenging problem and has not been systematically addressed yet. [4]

In this paper, we address the above challenges by proposing an integrated model to support dynamic QoS-aware service configuration. The service configuration model includes two tiers: (1) *service composition*, and (2) *service distribution* tiers. The former is responsible for choosing a set of suitable services, discovered in the current environment, to compose a customized application delivery to any client device. We also provide QoS consistency check to discover and correct inconsistencies of QoS parameters between any two interacting service components. The latter is responsible for dividing a distributed application into several partitions and dispatching them to different devices according to the current distributed resource availability.

We have implemented a prototype of our service configuration model as part of the Gaia OS [1], an enabling infrastructure for the smart spaces. Due to the scalability requirement, we structure the smart spaces hierarchically by grouping devices into different domains. Each domain contains one domain server, which provides the key infrastructure services for the entire domain space, in the same way as today's operating systems do for a single desktop. The *service configuration model* is implemented as part of the domain server. It cooperates with other domain services, such as the *event service*, to dynamically configure distributed applications for the user.

The rest of the paper is organized as follows. Section 2 introduces our application service model. Section 3 presents the dynamic QoS-aware service configuration model. Section 4 presents the experimental results based on both prototype and simulation. Section 5 discusses related works. Section 6 concludes this paper.

2 Application Service Model

We consider a generic component-based model to characterize the structure of distributed (multimedia) applications which are expected to run in the ubiquitous computing environment. All application components are constructed as *autonomous services*, which perform independent operations, such as transformation, synchronization, filtering, on the data stream passing through them. Services can be connected into a directed acyclic graph (DAG), which is called a *service graph*.

For supporting QoS, we assume that each component accepts input data with a QoS level Q^{in} and generates output with a QoS level Q^{out}, both of which are vectors of application-level QoS parameter values, such as data format (e.g., MPEG, JPEG), resolution (1600*1200 pixels), and others. In order to process input and generate output, a specific amount of resources R is required, which is a vector of different *end-system* resource requirements (e.g., memory, cpu). The network bandwidth requirements are associated with edges between two communicating components. Formally, we define the vectors Q^{in}, Q^{out}, and R as follows: $Q^{in} = [q_1^{in}, q_2^{in}, ..., q_n^{in}]$, $Q^{out} = [q_1^{out}, q_2^{out}, ..., q_n^{out}]$, $R = [r_1, r_2, ..., r_m]$.

Intuitively, if a component A is connected to a component B, the output QoS of A (Q_A^{out}) must "match" the input QoS requirements of component B (Q_B^{in}). In order to formally describe this *QoS consistency* requirements, we define an inter-component relation "$\preceq$", called "*satisfy*", as follows: $Q_A^{out} \preceq Q_B^{in}$ if and only if

$$\begin{aligned} &\forall i, 1 \leq i \leq Dim(Q_B^{in}),\ \exists j, 1 \leq j \leq Dim(Q_A^{in}), \\ &\quad q_{Aj}^{out} = q_{Bi}^{in},\ \ if\ q_{Bi}^{in}\ is\ a\ single\ value; \\ &\quad q_{Aj}^{out} \subseteq q_{Bi}^{in},\ \ if\ q_{Bi}^{in}\ is\ a\ range\ value. \end{aligned} \tag{1}$$

The "Dim(Q_A)" represents the dimension of the vector "Q_A". The *single value* QoS parameters include media format, resolution, and others. The *range value* QoS parameters can be frame rate ([10fps,30fps]).

3 Dynamic QoS-aware Service Configuration Model

In this section, we present the dynamic service configuration model, based on the above application service model. The configuration model includes: (1) *service composition* and (2) *service distribution* tiers, to address the two problems identified in Section 1, respectively. We first state a number of key assumptions made by the service configuration model and prove that those assumptions are valid in practice. We then present the design and algorithms for the two tiers in detail.

3.1 Assumptions

First, we assume that service components, that a ubiquitous application needs in order to run, are not explicitly named, but rather specified in an abstract manner. The developer should specify the application service at a high level of abstraction in order to accommodate unexpected runtime variations [4]. Several programming environments and specification languages have been proposed to allow developers to provide such abstract descriptions [5, 10]. Second, we assume that a service discovery service is available to find the service instances that are closest to the abstract service descriptions. Such a discovery service has been provided by different research work as well [6, 16].

In order to support dynamic partition of applications and relocation of service components in the *service distribution* tier, we further make the following assumptions. First, we assume that system services are available for saving and restoring application checkpoints and for migrating components with their data between nodes. Several research work [14, 9] has addressed and provided sound solutions for application checkpointing and migration. Second, we assume that profiling or monitoring services are available to automatically measure the resource requirements for all application services. Such online profiling techniques are investigated and provided in [2, 13].

3.2 Service Composition Tier

We now present the design and algorithms for the *service composition* tier, represented by the *service composer*. The major protocol steps, carried out by the *service composer* to generate a QoS consistent *service graph*, include the following:

- **Acquire the abstract service graph.** As we mentioned above, the developer should provide high level descriptions for ubiquitous applications. We call this high level application description *abstract service graph*. It is structured in the same way as the *service graph* and includes abstract specifications about each service component the application needs in order to run, and also the interactions/dependencies between these components. The developer can also abstractly specify *optional* services that, if present at runtime, enhance the application.

- **Discover service instances in the current environment.** Once the abstract service graph is acquired, the discovery service is invoked to find suitable service instances in the current environment, according to the abstract descriptions. It also takes into account the user's QoS requirements and properties of the client device (e.g., screen size, computing capability). The returned component should be the one closest to the service's abstract descriptions. It is possible that no discovered component is returned for a particular service.

- **Check QoS consistencies and coordinate ad-hoc interactions.** The service instances, returned by the second step, are concrete service components discovered in the current environment. They include more detailed and specific information than their abstract descriptions (e.g., resource/platform requirements). Moreover, the discovered components' specifications may not be exactly the same as their abstract descriptions. For example, the discovery service can only find a JPEG player in the current environment although an MPEG player is requested. Thus, the *service composer* needs to check the QoS consistencies between discovered service instances and automatically correct the inconsistent interactions, if possible.

- **Generate the QoS consistent service graph and deliver it to the *service distribution* tier.** After the third step, a QoS consistent service graph is generated and is then delivered to the *service distribution* tier.

Among the above four steps, the third one forms the key part of the *service composer*. It tackles the following major problems brought by the dynamic service composition in ubiquitous computing environments: (1) failed discovery of a service instance; (2) fast and efficient QoS consistency check among discovered service instances; and (3) automatic correction of inconsistent interactions. We will explain in detail how these problems are addressed by the *service composer* in the next paragraphs.

For the first problem, if the service that cannot be discovered is optional, then the *service composer* may simply neglect it. Otherwise, the service composer can either recursively apply the service composition algorithms to the missing service or send a notification to the user. In the former approach, the service composer tries to find the service graph that can perform the same task as the missing service does [1]. In the latter approach, the user can either download and install an instance for the missing service into the current environment, or simply quit the application.

For the second and third problem,we propose an *Ordered Coordination* (*OC*) algorithm to perform QoS consistency check and automatic correction on the *service graph*. It includes the following major operations, illustrated in Figure 1: (1) topologically sort the instantiated *service graph*; (2) check the QoS consistency, in the reverse order of topologically sorting, between each node and its predecessors, according to the inter-component relation "*satisfy*" defined by equation (2) in Section 2; (3) If any inconsistency is found, possible automatic corrections are performed. In the general case, developers should decide how to correct QoS inconsistencies. However, if components' output QoS parameters can be dynamically configured, we can adjust the output QoS of the current node's predecessor to make it "*sat-*

[1] In order to avoid infinite recursive service compositions for the missing service, we limit the depth of recursion to 2 in the practical implementation.

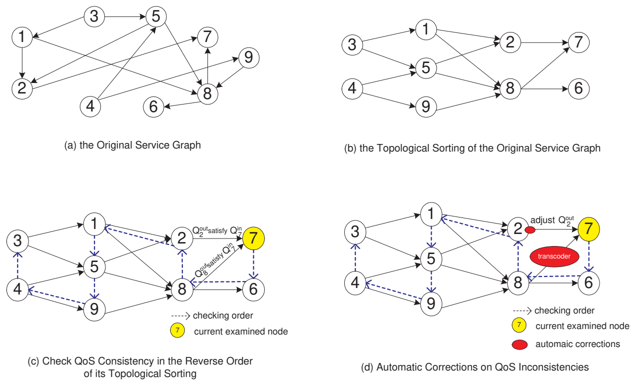

Figure 1. Major operations in the ordered coordination algorithm.

isfy" the input QoS requirements of the current node. Then the input QoS requirements of the predecessor need to be adjusted accordingly and so on. Hence, the *OC* algorithm not only preserves the QoS consistency but also best supports the user's QoS requirements because the output QoS parameters of the first examined nodes [2], which often correspond to the user's QoS requirements, are preserved. We may also insert transcoders in the middle to solve the type mismatches or insert buffer component to alleviate performance mismatches. The computational complexity of *OC* algorithm is *O(V+E)*, where *V* and *E* are the numbers of service components and edges in the final *service graph*, respectively.

The *service composer* is activated whenever some significant changes are detected during runtime. For example, when the user moves to a new location, the previous service components may no longer be available. Or when the user switches to a different device (e.g., from PC to PDA), the previous service graph can no longer be supported. Under these circumstances, the *service composer* will generate a new *service graph* on-the-fly. Thus the user can continue to perform tasks, after the state handoff from the old service graph to the new one.

3.3 Service Distribution Tier

This section presents the design and algorithms for the *service distribution* tier, represented by the *service distributor*. After the *service composer* generates a consistent *service graph*, the *service distributor* is responsible for properly distributing service components so that the *service graph* can "fit into" the current available devices. As we mentioned in Section 2, each component is associated with an end-system *resource requirement* vector $R = [r_1, r_2, ...r_m]$, with r_i ($1 \leq i \leq m$) as the required amount of the ith resource type. Each edge *e = (u,v)* in the graph is assigned an integer weight "*c(u,v)*", representing the communication throughput from node u to node v. In addition, we use a vector RA to represent *resource availability* on each device. We assume that R and RA represent the same set of resources and obey the same order. We use $b_{i,j}$ to represent the end-to-end available network bandwidth from the ith to the jth device. Figure 2 illustrates the above concepts. The graph nodes in Figure 2 represent the minimum partitionable service components. Before we present the algorithm for the *service distributor*, we first introduce the following definitions.

[2] The first examined nodes are the last ones in the topological sorting order and thus usually correspond to client services.

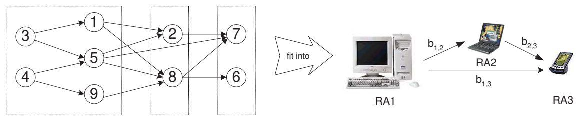

Figure 2. Illustration of service distribution problems.

DEFINITION 3.1 The addition of two resource vectors for service component A and B, $R^A = (r_1^A, r_2^A, ..., r_m^A)$ and $R^B = (r_1^B, r_2^B, ..., r_m^B)$, is defined as the following:

$$R^A + R^B = [r_1^A + r_1^B, r_2^A + r_2^B, ..., r_m^A + r_m^B] \qquad (2)$$

DEFINITION 3.2 Given resource requirement vector $R = [r_1, r_2, ...r_m]$ and resource availability vector $RA = [ra_1, ra_2, ...ra_m]$, $R \leq RA$ if and only if

$$\forall i, 1 \leq i \leq m, r_i \leq ra_i, \qquad (3)$$

DEFINITION 3.3 Let G = (V,E) be a directed graph. A *k-cut* in G is a partitioning of V into nonempty subsets V_1,..., V_k. An edge e belongs to the *k-cut* if its endpoints belong to different subsets of the partition V_1,...,V_k.

For instance, Figure 2 shows a 3-cut of the service graph. The edges belonging to the cut are $\{e_{1,2}, e_{1,8}, e_{5,2}, e_{5,8}, e_{5,7}, e_{9,8}, e_{2,7}, e_{8,7}, e_{8,6}\}$.

DEFINITION 3.4 Given the *service graph* $G = (V, E)$ and K ($2 \leq K \leq V$) available devices, we define that G can "*fit into*" those k devices, if and only if there exits a *k-cut* (V_1,...,V_k) of the graph, such that

- $\forall j, 1 \leq j \leq k, \sum_{v_i \in V_j} R^i \leq RA^j$, where R^i represents the resource requirement vector of component v_i, RA^j represents the resource availability on the jth device;

- $\forall i, j, 1 \leq i, j \leq (k-1), \forall e = (u,v) \in E, u \in V_i, v \in V_j, \sum_{e \in E, u \in V_i, v \in V_j} c(u,v) \leq b_{i,j}$, where $c(u,v)$ represents the communication throughput on the edge e that belongs to the *k-cut*, $b_{i,j}$ represents the available bandwidth between the ith and jth devices.

Usually, there exist multiple k-cut schemes that can fit the service graph into k devices. Thus, the *service distributor* needs to find the one with the *Minimum Cost Aggregation*. The concept of *Cost Aggregation* is defined as follows:

DEFINITION 3.5 Given a k-cut Φ = (V_1,...,V_k) for the service graph G = (V,E), its *Cost Aggregation (CA)* can be calculated in the following way:

$$CA(\Phi) = \sum_{j=1}^{k}\sum_{i=1}^{m} w_i \cdot \frac{r_i}{ra_i^j} + \sum_{1\leq i,j \leq k}^{i\neq j} w_{m+1} \cdot \frac{T_{i,j}}{b_{i,j}} \quad (4)$$

where $T_{i,j}$ = $\sum_{u\in V_i, v\in V_j} c(u,v)$ and w_i (1 $\leq$ i $\leq$ (m+1)) are nonnegative values so that $\sum_{i=1}^{m+1} w_i = 1$.

For any end-system resource type r_i (e.g., memory, cpu), $w_i \cdot \frac{r_i}{ra_i}$ is a normalized value between 0 and 1, where w_i represents the significance of this resource type. Generally, we assign higher weights for more critical resources. For the network resource type, $w_{m+1} \cdot \frac{\sum_{u\in V_i, v\in V_j} c(u,v)}{b_{i,j}}$ is a normalized value between 0 and 1, where w_{m+1} represents the significance of network resource. In both cases, the normalized value represents the cost the user pays for using a specific type of resource to perform his or her tasks. Intuitively, the more important (higher weight) and more scarce (smaller resource availability) the resource is, the larger cost (larger normalized value) it takes the user to use it. Minimizing the *cost aggregation* can help improve the total resource utilization and reduce the contention on critical resources. As a result, the user's QoS requirements can be better preserved and more applications can be supported simultaneously given the union of all resources. Thus, the goal of the *service distributor* is to find a *k-cut* for the given *service graph*, which can make the graph *fit into* the current k available devices and also minimizes the *cost aggregation* for the user.

However, we neglect several important practical issues in the above analysis. First, we assume that every service component can be instantiated on any device. But the assumption does not hold in reality and some services must run on certain device. For example, the display service in the video-on-demand application must run on the client device. Second, the model described so far is not *heterogeneity-aware*. In other words, the k available devices are assumed to be the same. To solve the first problem, we can first "*pin*" those special components on proper devices by inserting them into the corresponding subsets (V_i, Definition 3.3) of partitions. For the second problem, we need to normalize both the *resource requirement* and *resource availability* values on heterogeneous machines to those on a benchmark machine. For example, if we use a laptop as the benchmark machine and assume the resource availabilities of a PDA and a PC are RA^{PDA} = [32MB (memory), 100% (CPU)] and RA^{PC} =[256MB,100%], then the two normalized resource availability vector values on the benchmark machine (laptop) may become $N(RA^{PDA})$ = [32MB, 40%], $N(RA^{PC})$ = [256MB, 500%]. We assume that the memory availability values are not affected by device heterogeneity. However, the normalized CPU availability should be changed according to the speed difference between the heterogeneous device and the benchmark machine. Similarly, the *resource requirement* values also need to be normalized to those on a benchmark machines. In the general case, the above normalization functions can be derived through experimental measurements. For simplicity, we assume that the values in Definitions 3.1 to 3.5 are all normalized values. Thus, we can apply those definitions to the ubiquitous computing environment. We now show that the general problem of finding the optimal service distribution (*k-cut*) that makes the service graph *fit into* k devices and also minimizes the *cost aggregation* is NP-hard.

Theorem 1 *Finding the optimal service distribution(OSD) that makes the service graph fit into k devices and also minimizes the cost aggregation is NP-hard.*

Proof: We prove this by showing that the *minimum directed multi-way cut* problem which is known to be NP-hard [7] maps directly to a special case of our service distribution problem. The minimum directed multi-way cut problem is as follows: Let G = (V, E) be a directed graph and let c(u,v) be a non-negative capacity function associated with the edge e = (u,v).

$$Minimize \sum_{1\leq i,j\leq k}^{i\neq j} \sum_{u\in V_i, v\in V_j} c(u,v) \quad (5)$$

where V_1, V_2,..., V_k are k non-empty subsets of V and form a k-cut of graph G.

The above problem is identical to the following special case of our problem. Suppose each of the k available devices has infinite end-system resource availability. Thus, any k-cut of service graph G can satisfy the "fit into" constraints. We also assume that every service component can be assigned to any of the k available devices. In addition, we let (1) w_i (1 $\leq$ i $\leq$ m) be 0 and w_{m+1} be 1; (2) every available bandwidth $b_{i,j}$ be 1 (Gbps). An identity transformation makes the minimum directed multi-way cut problem a special case of our *OSD* problem, shown by equation (4). Thus, the *OSD* problem is also NP-hard. □

Since the number of service components can be large given fine service granularity and complex applications, we provide a polynomial heuristic algorithm for the *OSD* problem. It primarily involves the following steps: (1) insert

Event Label	Event Content	Service Configuration Results	Measured QoS
1	Start *"mobile audio-on-demand"* on the desktop1. User QoS request: CD quality music	Desktop 1 Audio Server → Desktop 2 Audio Player	40fps
2	Switch from desktop to PDA with a wireless link. Music continues from the interruption point.	Desktop 1 Audio Server → Desktop 2 MPEG2wav Transcoder → Jornada Audio Player	40fps
3	Switch back from PDA to another desktop3	Desktop 1 Audio Server → Desktop 3 Audio Player	40fps
4	Start *video conferencing* on the workstations. User QoS request: video(25fps), audio(6fps)	Work Station video recorder, audio recorder → Work Station2 gateway → WorkStation3 lipsync → video player, Audio player	25fps, 6fps

Figure 3. End-to-end QoS of different service configurations.

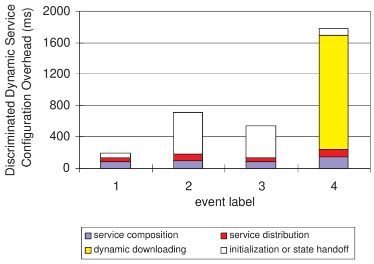

Figure 4. Overhead of each dynamic service configuration actions illustrated in Figure 3.

those service components, that cannot be instantiated arbitrarily, into their proper devices; (2) repeat sorting the k available devices in decreasing order of their resource availabilities and insert the next chosen service components to the current head of the sorted device list, namely the device that currently has the largest resource availability. If the head device contains a service component A, then the next chosen component is A's neighbor, which has the largest resource requirements [3]. We then insert the chosen component into the head device and merge it with A. If the head device is empty, then the next chosen service component is the one which has the largest resource requirements among all remaining service components. Repeat the above procedure until every service component has been inserted into a proper device.

The *service distributor* is invoked whenever some *significant* resource fluctuations or device changes happen during runtime. For example, if one of old devices crashes or a new device is added, the *service distributor* needs to calculate new service distributions for the changed resource availability. Thus, the user can continue his or her tasks with minimum QoS degradations.

4 Experimental Results

We have implemented a prototype of the service configuration model as part of the Gaia OS [1], an enabling infrastructure for smart spaces, and performed several experiments based on both the prototype and simulations.

Our first set of experiments is performed based on the prototype, using two distributed multimedia applications, *mobile audio-on-demand* and *video conferencing* implemented in our lab. The servers/gateways in our testbed are either Sun Ultra-60 workstations or Pentium III 900 PCs. The client devices are either PCs, laptops (IBM Thinkpad), or PDAs (HP Jornadas). we demonstrate that the service configuration model is able to provide soft QoS guarantees to the user in the ubiquitous computing environment. Figure 3 shows experimental results for both applications. The results from the *mobile audio-on-demand* application show that our dynamic service configuration can flexibly accommodate common runtime changes, such as user mobility and handoff between heterogenous devices. The experiment with the *video conferencing* application demonstrates the ability of our framework to handle on-demand service configuration for *non-linear* service graph. Figure 4 shows the dynamic service configuration overhead, during the above experiments. For the *mobile audio-on-demand* application, we assume that the required service components are already installed on the target devices in advance. Thus there is no dynamic downloading overhead involved. However, in the *video conferencing* application, we assume that all required service components need to be downloaded on demand from the component repository. The state handoff time includes the handoff protocol overhead and also the buffering time for the first frame at the interruption point. Since the PDA is connected with the wireless network while the PC is connected with the ethernet, the state handoff time from PC to PDA is longer than that from PDA to PC. Overall, the results show that the overhead of the dynamic service configuration is relatively small compared to the entire execution time of the application. Moreover, the dynamic downloading overhead, which occupies the largest proportion of the total overhead, can often be avoided if the required components are already on the target devices.

To demonstrate the efficiency of the proposed heuristic service distribution algorithm, we conducted two sets of experiments based on simulations. First, we compare the relative performances of different heuristic algorithms (random and ours) with the optimal algorithm. The optimal algorithm uses exhaustive search for the optimal service distribution solution. Since the problem is NP-hard,

[3] Both resource availability and resource requirement are measured using the weighted sum of different resources. Due to the page limit, the detailed equations are not shown.

we limit ourselves to the special case of two-way cut. We assume two heterogeneous devices (PC, PDA) are used, with initial normalized resource availability vectors RA^1 = [256MB, 300%], RA^2 = [32MB, 100%], respectively. We consider service graphs with 10 to 20 service components. Each component has, on average, 3 to 6 outbound edges. Other parameters including resource requirement vectors, communication throughput on each edge and weight values are uniformly distributed. Table 1 summarizes the comparison results for 150 randomly generated service graphs. The first column in Table 1 is the average performance of each heuristic, measured by the ratio of cost aggregation between the optimal solution and the solution found by the heuristic, averaged over all 150 graphs. The second column is the percentage of 150 graphs for which our heuristic or the random algorithm was able to find the exact optimal solution.

Algorithms	Average	Optimal
Random	25%	0%
Our Heuristic	91%	60%
Optimal	100%	100%

Table 1. *Comparisons among different service distribution algorithms.*

Second, we compare the overall *success rate* achieved by our heuristic algorithm with *random* and *fixed* algorithms. A service configuration request is said to be successful if the service graph can *fit into* the current available devices. The *success rate* is calculated by the ratio of the number of successful service configuration requests to the number of total configuration attempts. We assume three heterogeneous devices (desktop, laptop, and PDA) are used, with initial normalized resource availability vectors RA^1 = [256MB, 300%], RA^2 = [128MB, 100%], and RA^3 = [32MB, 50%] respectively. The available bandwidths $b_{1,2}$, $b_{1,3}$, and $b_{2,3}$ are initialized to be 50Mbps, 5Mbps, and 5Mbps respectively. We randomly create 5000 application requests over 1000 hours period. Each request randomly selects a service graph from 5 predefined ones. Each graph has 50 to 100 nodes with on average 5 to 10 outbound edges. The length of each application is exponentially distributed from 5 minutes to 1 hours. Other parameters, including resource requirement vectors, communication throughput on each edge and weight values, are uniformly distributed. When a new application starts or an old application stops, both our heuristic and *random* algorithms make the re-distribution decisions, but the *fixed* algorithm does not. The success rate is calculated every 50 hours. Figure 5 shows the comparison results among the *fixed*, *random* and our heuristic algorithms. *Fixed* algorithm has the lowest success rate because it lacks dynamic service distribution considerations. *Random* algorithm provides better success rate since it benefits

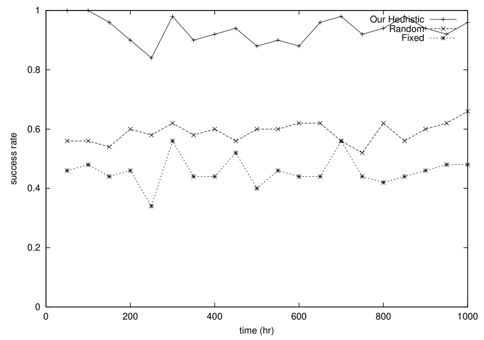

Figure 5. Success rate comparisons among the fixed, random and our heuristic algorithms.

from the flexibility of dynamic service distribution. Finally, our heuristic algorithm consistently maintains the highest success rate because it considers both resource availability and resource requirements during dynamic service distributions.

5 Related Work

The problem of dynamic service configuration has been addressed in different research works. In [12], the authors propose the concept of "path" to compose services on demand in heterogeneous environments. Although the concept of "path" can solve the type mismatches for ubiquitous multimedia streaming applications, it lacks the general QoS support and is limited to linear service graphs. In [17], the authors use a set of different compositions to represent the same application with different QoS levels so that the general QoS support is explicitly included into the service composition solutions. However, the service composition list for a particular application is predefined and thus cannot accommodate unexpected runtime changes which are a common case in the ubiquitous computing environment. Moreover, all of the above approaches only address *one* aspect of the service configuration problem, namely the service composition problem.

The idea of automatically partitioning and distributing applications is not new as well. In [3], the authors propose a hierarchical application partitioning approach to provide both potential for scalability and support for system heterogeneity. In the Coign [11] project, a system to automatically partition and distribute binary applications is proposed to ease the development of distributed applications. However, our work is different from the above similar works in the following primary respects. First, the goal of the service distribution in our work is not only to improve application performance but also to overcome the resource limitations of mobile devices. Second, we consider multiple resource types in finding the optimal service distribution for distributed multimedia applications. Moreover, the resources are differentiated so that the consumption of the most critical resource is minimized.

In conclusion, compared with the above similar works, the novelty of our work is to propose an *integrated* and *QoS-aware* model to address *both* aspects of the dynamic service configuration problem in the ubiquitous computing environment.

6 Conclusion

Ubiquitous computing brings new challenges to the dynamic service configuration research for the soft real time applications such as multimedia. In this paper, we present an integrated QoS-aware service configuration model to address the challenges. The major contributions of the paper are: (1) identifying two key problems, *service composition* and *service distribution* to support dynamic service configuration in ubiquitous computing environments; (2) introducing a two-tier integrated model to solve the above problems in a unified framework; (3) providing the design and polynomial algorithms for the *service composition* tier which includes automatic QoS consistency check and correction to support arbitrary interactions between service components; and (4) defining the *optimal service distribution* problem which is shown to be NP-hard and then providing a polynomial approximation algorithm for the problem.

7 Acknowledgment

We would like to thank Dejan Milojicic, Alan Messer, Ira Greenberg at HP Labs for part of the idea presented in this paper is inspired by the discussion with them. We are also grateful to Long Wang and Duangdao Wichadakul for their contributions in the prototype implementation. Finally, we would like to thank anonymous reviewers for their helpful comments.

References

[1] GAIA: Active Spaces for Ubiquitous Computing. *http://devius.cs.uiuc.edu/gaia/*.

[2] T. F. Abdelzaher. An Antomated Profiling Subsystem for QoS-Aware Services. *Proc. of IEEE Real-Time Technology and Applications Symposium*, June 2000.

[3] T. F. Abdelzaher and K. G. Shin. Period-Based Partitioning and Assignment for Large Real-Time Applications. *IEEE Transactions on Computers, vol.49, No.1*, 2000.

[4] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. Challenges: An Application Model for Pervasive Computing. *Proc. of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom2000)*, 2000.

[5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. *http:www.w3.org/TR/wsdl,*, Mar. 2001.

[6] S. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. *Proc. of Fifth Annual International Conference on Mobile Computing and Networks (Mobicom'99)*, Aug. 1999.

[7] N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. *21st ICALP*, 1994.

[8] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Computer Networks, Special Issue on Pervasive Computing*, 2001.

[9] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. Systems directions for pervasive computing. *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

[10] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based Quality of Service Enabling Language for the Web. *Journal of Visual Language and Computing, Special Issue on Multimedia Language for the Web*, 2002.

[11] G. C. Hunt and M. L. Scott. The Coign Automatic Distributed Partitioning System. *Proc. of the 3rd USENIX Symposium on Operating System Design and Implementation (OSDI'99)*, Feb. 1999.

[12] E. Kiciman and A. Fox. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. *Proc. of the Second International Symposium on Handheld and Ubiquitous Computing 2000 (Lecture Notes in Computer Science , Spring Verlag)*, Sept. 2000.

[13] B. Li and K. Nahrstedt. QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications. *Proc. of IFIP International Conference on Distrbited Systems Platforms and Open Distributed Processing (Middleware 2000)*, Apr. 2000.

[14] D. Milojicic, F. Douglis, and e. R. Wheeler. Mobility: Processes, Computers, and Agents. *ACM Press. Addison-Wesley*, Feb. 1999.

[15] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communication of the ACM, 36(7), pp. 74-84*, 1993.

[16] D. Xu, K. Nahrstedt, and D. Wichadakul. QoS-Aware Discovery of Wide-Area Distributed Services. *Proc. of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, May 2001.

[17] D. Xu, D. Wichadakul, and K. Nahrstedt. Multimedia Service Configuration and Reservation in Heterogenous Environments. *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS 2000)*, Apr. 2000.