A Flexible Awareness Measurement and Management Architecture for Adaptive Applications

Qiang Wang, Liang Cheng
Department of Computer Science and Engineering
Lehigh University, Bethlehem, PA
qiw3@lehigh.edu, cheng@cse.lehigh.edu

Abstract—**Advancements in heterogeneous and pervasive computing introduce many dynamic features that require applications to be adaptive. To become adaptive, an application first needs to be aware of its computing environment characteristics (awareness) as well as its peer's awareness. This paper presents flexible awareness measurement and management architectures for adaptive applications in a pervasive computing environment. We investigate how our system can be beneficial to adaptive applications by providing an integrated interface (adding, deleting, and querying) to different types of awareness, the measurement extensibility, and application controlled measurement behaviors. To efficiently manage awareness, we then discuss pull/pull method, and flexible consistency management. A hybrid architecture is used to efficiently distribute awareness for both wired and wireless networks.**

Keywords-Adaptation, awareness measurement, awareness management, heterogeneous environments.

I. INTRODUCTION

In today's decentralized Internet, end-to-end network characteristics exhibit great variance [9]. Available bandwidth between two hosts can vary from several gbps to few kbps. Round trip time varies from less than one millisecond to a few thousand milliseconds. Jitter may change significantly in one communication session, depending on traffic flows and congestion in the network. Above mentioned network characteristics can affect the quality of service of a distributed system. In contrast to resource reservation in ATM networks, applications in IP networks generally use built-in adaptation mechanisms to accommodate different network conditions.

Moreover, the advancement in heterogeneous and pervasive computing introduces more dynamic features that require applications to be adaptive. Today's IP networks are largely heterogeneous; wired and wireless connections coexist. Wired channels are typically more reliable and wireless channels are subject to interference and fading. The design of distributed applications therefore should consider the different features of wired and wireless networks. In a pervasive computing environment, almost any type of electronic device is connected, ranging from embedded sensors and handheld devices to traditional PCs. Computing devices and their applications have to be aware of their surroundings and peers in order to be capable of effectively providing services to and using services from their peers. Because of the wide range of possible characteristics in heterogeneous and pervasive computing, it is desirable for distributed applications to adapt to their computing environments, and make different adaptation decisions based on current, historical, and/or predicted characteristics of the computing environment.

To become adaptive, applications first need to be aware of computing environment characteristics and their changes. In this paper, awareness is defined as the information of computing environment characteristics that are needed to perform adaptations. The notation of awareness in this paper extends the concept of network-awareness [2] to incorporate device awareness, application awareness, end user awareness, and physical environment awareness.

Adaptive applications are not only interested in their own awareness information, but also the awareness of their peers. In a pervasive computing environment, the inequity of the devices and networks in a system requires an application to know its peers' awareness in order to make a proper adaptation decision. How to efficiently retrieve and distribute awareness information is one of our interests in this research.

Usually, an adaptation technique is specific to the application scenarios being implemented. To facilitate the generation of adaptive applications, we are developing an adaptation middleware called AwareWare [11], which provides common adaptation functions for component-based applications (CORBA in current implementation). This paper presents the early stages of development of a flexible awareness measurement and management architecture in AwareWare. Some important challenges are identified in the process, which we discuss, as well as possible solutions. The rest of this paper is organized as follows. Section II introduces the overall architecture of AwareWare. Section III introduces integrated awareness measurement for five types of awareness in AwareWare. Section IV presents a hybrid awareness management architecture for wireless and wired heterogeneous networks. We summarize related work in section V and conclude in section VI.

II. AWAREWARE MIDDLEWARE

AwareWare [11] is middleware situated between the Operating System and adaptive applications. The architectural design of AwareWare is depicted in Figure 1. Awareness measurement tools measure and collect network characteristics (i.e. awareness), device characteristics, end-users' preferences, applications' internal states and physical environments that are relevant to adaptation. The awareness manager organizes these tools and provides system independent query and notification interfaces for adaptive applications. To address issues in wireless and wired heterogeneous networks, awareness data is distributed across the network by using a hybrid architecture with flexible consistency controls. The adaptation decision module takes awareness as input and initiates adaptation directives to the application. It also provides a feedback control loop to the awareness manager, which in turn controls the behaviors of measurement tools. The awareness manager selects proper tools to accommodate application's requirement for the measurement. Adaptation decisions are driven by a script, which is written by programmers in an adaptation policy language. The adaptation policy defines rules to determine how the application changes its behaviors, by changing the application's component inter-connections and tuning parameters.

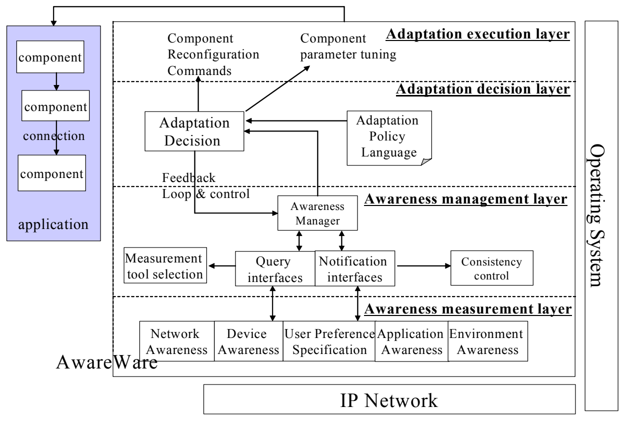

Figure 1. AwareWare middleware architecture

Dynamic reconfiguration [1] is the basis of application adaptation in AwareWare. Adaptation can be achieved by the reconfiguration of the application's components in response to environment changes and/or operator's commands. Programmers develop their applications using a set of separated components connected via CORBA. These components interact with others only via their interfaces, without explicitly binding to any other components, unlike normally do in CORBA applications. (Otherwise, the inter-connection can not be reconfigured at run time.) Each component is then turned into a dynamic reconfigurable component by applying a CORBA packaging template. After packaging, each component has a set of standard reconfiguration primitives, e.g., blocking/unblocking a connection, adding/removing a connection, serializing/restoring internal states, etc. Packaging codes respond appropriately to reconfiguration commands initiated from the adaptation decision module. Since the packaging codes are automatically created by the packaging template, the complexity of reconfiguration is handled transparently to the programmers.

III. AWARENESS MEASUREMENT

A. *Awareness for Adaptation*

To incorporate adaptation, applications first need to be aware of the computing environment characteristics (awareness). There are some existing awareness measurement tools that collect network and device information. However, these tools are not integrated, system-independent APIs for developers. One of our goals in this research is to provide an easy integrated interface to allow adaptive applications to query awareness information. In table I, we summarize what aspects of awareness are to be measured and how they are measured in AwareWare. We then provide more detailed explanations for network awareness and environment awareness.

TABLE I. AWARENESS AND DETECTION APPROACHES

Awareness types	**Awareness Measurement**	
	what to measure	***how to measure***
Network awareness	End to end capaticy, available bandwidth, latency, jitter.	Active probing tools, explained latter in this section.
Device awareness	CPU usage, display size, memory usage, display refresh rate, battery consumption (for mobile devices).	Operating System APIs and by integrating with existing tools, e.g. vmstat and uptime for CPU and memory under Unix/Linux systems, and Performance Data Helper Library provided by MS Windows.
User awareness	A user's high level expectations of a service.	A user can specify their preferences through Graphic User Interfaces (i.e. by selecting menus/dialog boxes).
Application awareness	Internal states of local and remote applications.	Exported by the local application to a shared memory, which is accessible by the middleware.
Environment awareness	Physical and environmental data (e.g. temperature).	Measured by WSNs (wireless sensor networks) and integrated into the Internet/intranet.

Among five types of awareness supported in the middleware, network awareness exposes greater challenges and it has been a continuing interest for research and industry communities to provide reliable network-awareness measurement tools. Our end-to-end capacity detection tool uses packet probing techniques, and it is discussed in detail in [4]. The essence of the algorithm is a variation of multi-packet model: in order to measure the bottleneck link between two network nodes A and B, A sends a train of *back to back* packets to B. The bottleneck capacity from A to B can be estimated from the time interval of the received packets at B and the probe packet size. The probe packet train in [4] is generated by a smooth traffic generator, which efficiently eliminates the side effect of traffic shaping in xDSL network.

The physical environment is measured by integrating with WSNs (wireless sensor networks). A WSN consists of many tiny sensors with sensing capability of a physical environment, and sensors communicate with each another through wireless links. The ability to detect changes in a physical environment is useful for some adaptive applications. For example, in distributed fire fighting planning, the adaptation has to be made by using on-site temperature. By integrating WSNs with the Internet, AwareWare serves as a gateway for traditional applications to query awareness from a WSN [5]. In our current system, environmental awareness collected by sensors include temperature, acceleration of a particular sensor node, and the distance between two sensor nodes.

B. Extensibility

Besides the awareness described above, AwareWare has the ability to add new types of awareness. Similar to [12], awareness is organized in a hierarchical architecture and can be easily added, retrieved, and modified according to a naming convention. This hierarchical organization also conforms to the SNMP [3] MIB standard for simplicity, extensibility, and compatibility, providing a flexible way to integrate more measurement tools, if needed in the future.

C. Application-Controlled Measurement

A measurement tool needs to be controlled by the adaptive application. In this session, we use bandwidth measurement as the example to specify our design consideration of application controlled measurement, since measuring bandwidth is particularly complex compared to others (e.g., measuring the screen size), and the interaction between bandwidth measurement tools and the application is complicated.

An active network bandwidth measurement tool consumes a certain amount of network resources. Active probing tools, such as our measurement tool [4], Packet Bunch Mode [9], Pathchar [6], and Packet tailgating [8], usually send several kilobytes into the network to generate a single bottleneck bandwidth measurement. Active probing traffic competes with application traffic. Too much active probing traffic will consume bandwidth, therefore degrading an application's performance. One goal of measurement, however, is to efficiently utilize the existing bandwidth and to improve application performance.

On the other hand, a bandwidth measurement tool can acquire bandwidth awareness with differing accuracy and overhead. Generally speaking, sending more probing packets into the network and measuring the network more frequently can generate more accurate results. However, the degree of accuracy may not be beneficial to an adaptive application. For example, in a network-aware application, if the application only needs to adapt to significant bandwidth change (e.g., a 50% bandwidth drop), measurement tools should reduce the measurement frequency and the amount of probing traffic if a relatively stable bandwidth is predicated based on measurement history. To address this problem, feedback loop control and interaction from the application is needed. AwareWare integrates a set of bandwidth measurement tools and lets the application specify measurement accuracy. The accuracy specifications for the adaptation decision module will help AwareWare to select the proper tool and proper parameters to perform the measurement.

IV. Awareness Management

In addition to the application's own awareness information, the application may also need to know the awareness information of its peers. Therefore awareness information needs to be distributed to whatever other applications that are interested in. There are two basic methods for querying and distributing awareness information. One is pull, another is push. An interested application can explicitly query awareness sources using a pull method. In contrast, with the push method, an awareness source pushes information to interested applications without the need of an explicit inquiry from them.

A. Pull method

Existing distributed awareness architectures generally use peer-to-peer and client-server architectures, as shown in Fig 2.A and Fig. 2.B. In Figure 2, a circle represents an end host, and a filled rectangle represents a piece of awareness information. Node 1 represents a wireless client, and others are all wired clients.

In a pervasive environment, communication is made across both wired and wireless domain. A wireless link is a very constrained resource, especially when used by handheld devices. Existing architectures to manage the awareness are not efficient in this environment, as analyzed below.

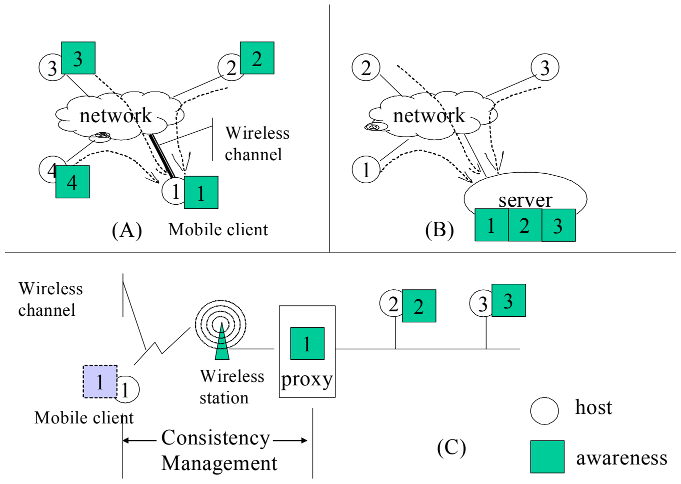

Figure 2. Distributed awareness architectures

The pure peer-to-peer architecture (Fig 2.A) doesn't scale well – when many hosts request awareness information 1 from mobile host 1, query and response traffic will all go through the wireless channel. Even worse, many repeated queries for the same awareness from many peer applications will consume a large portion of wireless bandwidth.

The client-server architecture (Fig 2.B) also suffers problems of scalability, where the server is the central depository of all awareness information from all hosts. While the client-server architecture may simplify the consistency management algorithm, a single server is a "single point of failure" of the whole system. In addition it may be the bottleneck for system performance.

We address these problems by using a hybrid architecture (Fig 2.C). A proxy is used to separate the wireless from the wired domain. The proxy is a high end computer attached to a wireless access point. In a wired network, the architecture is peer-to-peer while in a wireless network the architecture is client-server. Advantages of using a hybrid architecture are two fold:

First, some awareness detection methods can be directly performed at the powerful proxy, instead of in the less powerful handheld mobile devices. For example, the awareness information of bottleneck capacity from a mobile device (A) to another computer (B) can be measured in the proxy instead of in the mobile device: The proxy first measures the bottleneck capacity from the proxy to A, which equals to the bottleneck capacity from A to the proxy, given the fact that upward and downward communication channels generally are symmetric (one exception is aDSL, which is asymmetric). The proxy then measures the bottleneck capacity from the proxy to B. The minimum of two measures is the bottleneck capacity from A to B. A to B available bandwidth can also use a similar approach.

Second, awareness information inquiries to mobile devices will be satisfied directly from the proxy. Repeated and large volume queries do not need to go through wireless channels. In case the awareness is produced at the mobile client side, the wireless channel is only used to maintain consistency between the awareness and its replica at the proxy (section IV.B).

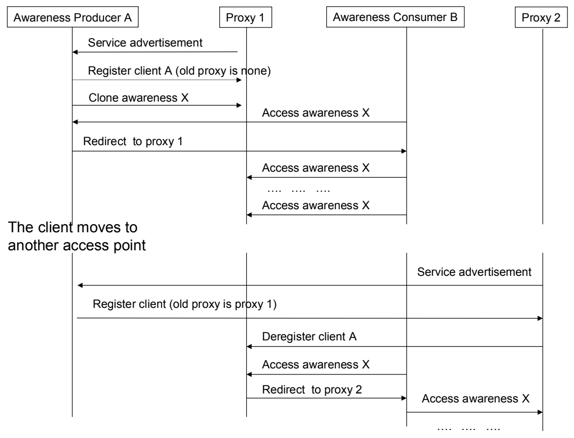

Figure 3. Message exchanges for allocating awareness

By introducing the proxy, several message exchanges are needed to allocate the appropriate awareness. As shown in Figure 3, when a proxy (proxy1) is active, it automatically advertises its services through a well-known port to all mobile clients covered by the wireless access point to which proxy 1 is attached. After receiving the advertisement, mobile client A (awareness producer) registers itself to proxy1. Assuming A produces awareness X at its local side instead of at the proxy side, A then clones a replica of X in proxy1. When B (an awareness consumer) wants to get awareness X from A, since B only knows A's IP address, therefore it first queries A for X. A returns the value of X to B, along with a redirect message containing proxy1's information, notifying B that awareness X needs to be queried from proxy1. In this approach, if B only needs to query for X once, it is satisfied by A's return message. However, later access to X from B will be directly satisfied from proxy1.

When mobile client A moves to a new location covered by proxy2 (proxy1 now is out of range), A will register itself to the new proxy, and tells the new proxy the information of the old proxy. Since A can not communicate with proxy1, the new proxy, proxy2 then deregisters A from its old proxy (proxy 1). Proxy1 also redirects all incoming query message of awareness X to the new proxy (proxy 2).

B. *Consistency management*

In our hybrid architecture, the proxy maintains replicas of awareness for mobile clients. A consistency policy is assigned to each individual awareness replica at the time when the replica is initially cloned. Since different awareness may require a different level of consistency, several consistency policies are necessary. Figure 4 shows five consistency mechanisms included in the middleware, and their interaction protocol among awareness producers, replicas, and awareness consumers (i.e., applications that query the information).

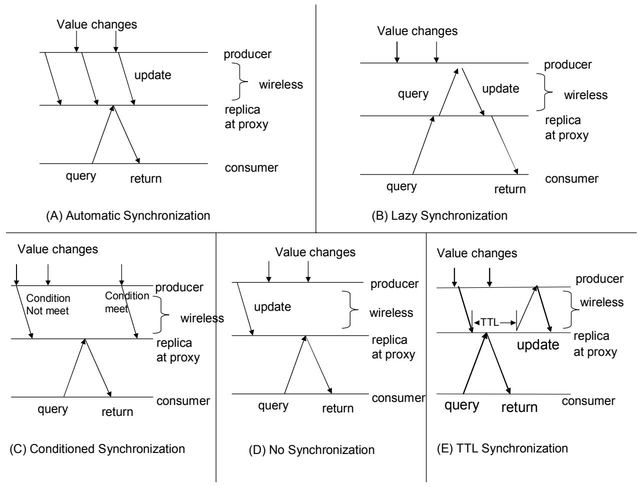

Figure 4. Consistency mechanisms

1) Automatic synchronization (Fig. 4.A): awareness producers at a mobile client synchronize the replica whenever any awareness change occurs. Automatic synchronization provides timely update of the awareness to its consumers. However, the wireless bandwidth between mobile clients and the proxy (i.e., wireless channel) is not fully utilized, since not every update is useful if access from consumers is infrequent.

2) Lazy synchronization (Fig. 4.B): The proxy synchronizes a replica with an awareness producer only when a consumer looks at the replica. This lazy consistency mechanism is beneficial for systems with limited network access, and it reduces wireless bandwidth usage.

3) Conditioned synchronization (Fig. 4.C): An awareness producer only needs to synchronize the replica when certain conditions are met. One example of the condition is: only when the change of bandwidth exceeds a predefined value, the producer updates the replica.

4) No synchronization (Fig. 4.D): The proxy only maintains a snapshot of awareness without any update mechanism. It is used for static awareness information, e.g., the screen size of a device (assuming that it is fixed).

5) TTL synchronization (Fig. 4.E): As in Web caching, an awareness replica is cloned in the proxy when awareness is first accessed. A time to live (TTL) property is associated with the replica. For each query arriving within the TTL, the cached object (replica) is returned without accessing the awareness producer. When the TTL expires, the replica is validated through an update.

C. *Push Method*

One problem for distributing awareness information using the pull method is that sometimes it is difficult for an adaptive application to decide when to pull the awareness source. One example of an adaptation policy in an application B is shown as followings:

```
If awareness Y in a remote application A < a
predefined value, then …
```

If B continuously pulls for Y from A, it may not be efficient. Instead, application B needs to register to a particular event at A, and the adaptation rule in B is the trigger function. The awareness manager in A then checks these registered event conditions every time the measurement tools finish a measurement. An event notification will be sent to interested applications if conditions are met. The push method is essentially same as the publish/subscribe paradigm, where the publisher pushes certain messages to subscribers who have registered an interest.

V. Related Work

Simple Network Management Protocol (SNMP) [3] is the most widely used and deployed network management framework. SNMP also supports queries and notification. However, event notifications from an SNMP agent are generally very simple. Events are detected by device-drivers and only include cold or warm start of a device, up or down of a link, and the loss of a neighbor. Another event category is the authentication failure. There is no quantity notification in SNMP; for example, a bandwidth utilization of a certain link exceeds a predefined value. Clearly, SNMP is used not just to monitor network-awareness, but also to control network devices. However, the control is initiated from the SNMP manager without further consideration of the adaptive application's specific requirements.

Mark Stemm et al. have proposed SPAND [10], a shared passive network performance discovery architecture and adaptive application framework. SPAND is essentially a client-server architecture, where Performance Server is a central server that contains performance reports generated from clients and Packet Capture Hosts. SPAND provides extensibility to integrate new applications and new metrics. The concept of application specific metrics in SPAN is similar to application-awareness in our system. In this paper, we specifically consider the case when the number of possible peers is large and wireless link is involved. We also consider the interactions between an adaptive application and its measurement tools.

Bjorn Knutsson et al. [7] designed scalable mechanisms to distribute states of massively multiplayer games to participating players by using peer-to-peer overlays. Their approach can maintain consistency of a replica in the face of node failures. The scalability is achieved by organizing game participants into several self organized groups. Their peer-to-peer architecture treats all nodes homogenously and works well in massively multiplayer games, where all participants are high-end PC users. In our research, however, we consider a more realistic scenario in pervasive computing where the networks across both wired and wireless domain and device capabilities are diversified.

VI. Conclusions

This paper addresses the important topic of measuring and managing awareness in the context of adaptive applications in a pervasive computing environment. Awareness measurement module within AwareWare provides an integrated interface to different types of awareness, the ability to extend, and application controlled measurement. We believe our systematic approach can be beneficial to adaptive application development. Awareness management in AwareWare employs a hybrid architecture. Compared to client-server and peer-to-peer architectures, the hybrid architecture has some advantages for distributed systems across both wired and wireless domains. Our system provides pull and push interfaces for distributing the awareness information. Several consistency mechanisms provide some degree of flexibility to meet the requirements of different adaptive applications.

AwareWare is now under active development. Our ultimate goal is to provide a high performance and flexible middleware for component-based adaptive applications. Our continuing work will demonstrate the performance of the architecture through experimental tests and real-world applications.

Acknowledgments

We thank Dr. Brian D. Davison for his help to improve the presentation of the paper, and anonymous reviewers for their insightful comments.

References

[1] B. Agnew, C. Hofmeister, and J. Purtilo. *Planning for Change: a Reconfiguration Language for Distributed Systems*. Distributed Systems Engineering, 1(5):313-- 322, Sept. 1994.

[2] J. Bolliger and T. Gross. *A Framework-Based Approach to the Development of Network-aware Applications*. IEEE Trans. on Software Engineering, 24(5):376-390, May 1998.

[3] J. Case, R.Mundy, D.Partain, B.Stewart. *Introduction to Version 3 of the Internet-standard Network Management Framework*. *RFC 2570*, 1999.

[4] L.Cheng. and I. Marsic. *Accurate Bandwidth Measurement in xDSL Networks*. Computer Communications,25(18), pp.1899-1710.Nov. 2002.

[5] L.Cheng, Y. Zhang, T. Lin, and Q. Ye. *Integration of Wireless Sensor Networks, Woireless Local Area Networks, and the Internet*. In Proceedings of 2004 IEEE International Conference on Networking, Sensing and Control, Taipei, China, Mar 21-24, 2004

[6] V.Jacobson. *Pathchar: A Tool to Infer Characteristics of Internet Paths.* ftp://ee.lbl.gov/pathchar, 1997.

[7] B. Knutsson, H. Lu, W. Xu and B. Hopkins. *Peer-to-Peer Support for Massively Multiplayer Games*. In Proceedings of IEEE INFOCOM 2004, March 2004, Hong Kong, China.

[8] K.Lai and M. Baker. *Measuring Link Bandwidths Using a Deterministic Model of Packet Delay*. In Proc. of the ACM SIGCOMM, pages 283--294, Stockholm, Sweden, August 2000.

[9] V.Paxson. *End-to-End Internet Packet Dynamics*. IEEE/ACM Transactions on Networking, Vol. 7, No. 3, 1999, pp. 277-292.

[10] M.Stemm, S.Seshan, and R. H. Katz. *A Network Measurement Architecture for Adaptive Applications.* In Proc. of the IEEE INFOCOM 2000, Mar. 2000, pp. 285--294.

[11] Q.Wang and L.Cheng. *AwareWare: An Adaptation Middleware for Heterogeneous Environments.* In Proceedings of the 2004 IEEE International Conference on Communications (ICC), June 20-24, 2004.

[12] G.Welling and B.R.Badrinath. *An Architecture for Exporting Environment Awareness to Mobile Computing Applications.* IEEE Transactions on Software Engineering, 24(5), pp. 391-400, May 1998.