2

Anywhere/Anytime Software and Information Access via Collaborative Assistance

Ren-Song Ko
National Chung Cheng University
Taiwan

1. Introduction

The development of computing has recently been dominated by three trends: the emergence of a wide range of embedded systems with diverse architectures and purpose; the rise of relatively high-speed mobile communication devices such as smart phones, personal digital assistants (PDAs), portable media players, ebook readers, etc; and the development of cloud computing, offering virtually unlimited data storage and computing resources, which may extend the capabilities of resource-constrained mobile devices. As a consequence, these devices and infrastructures have begun to pervade our daily life, creating a new paradigm in the interaction between people and computing environments along the lines of that envisioned by Weiser (Weiser, 1991), and thus opening up the potential of many novel applications.

For instance, the ubiquitous presence of computers allows people to carry with them only a minimal amount of computing hardware and software, depending on ambient computers to boost performance as needed. A smart phone may not have sufficient computation power to playback a high-definition movie but,rather than running the media playback software on a single device, one could look for available computers nearby and connect them together to constitute an *ad-hoc system* (Ko et al., 2008). The software can then utilize the resources of all participating devices to accomplish the execution collaboratively. Such a system is unplanned and organized on a temporary basis, usually to execute a specific task.

Another possible application is the timely information query. Even with sophisticated search engines available nowadays, the answers to some questions (e.g., "is the department store crowed now?" or "can someone take a snapshot of the car race now?") are time-sensitive and may become less significant if they cannot be obtained immediately; only people who are currently near the store or the race track can provide the appropriate responses. Similarly, with the ubiquity of mobile network-connected devices, it is highly possible to find such people and thus applications requiring timely information become feasible.

1.1 Problems

However, the scale of ubiquitous computing is huge, and so is the need to enable interoperability among mobile devices and infrastructures. Thus, realizing a system for ubiquitous applications as described above may face the following research challenges.

1.1.1 Software reuse

Given the number and variety of mobile and embedded devices, software development is, in most cases, a complex and time-consuming process since heterogeneous environments raise problems above and beyond source code level portability, which has a significant impact on ease of software reuse. With careful coding, the software may be compiled into native code for various platforms. Nevertheless, the software will probably be difficult to deploy and use. The source code has to be compiled for the target platform, either by vendors or users, and the computing environment needs to be correctly configured to the required hardware and shared libraries. While this problem may be alleviated by platform-independent intermediate bytecode and virtual machines (e.g., Java), a higher level of portability vis-a-vis performance also needs to be considered (Ko & Mutka, 2002). For instance, while a multimedia application may run perfectly on a desktop, it may run so slowly on a mobile device as to be unusable. One way to improve performance would be to reduce the output quality. An alternative would be to improve software to detect and exploit special the functionality of particular devices.

To achieve a higher level of portable performance, information on the resources available on the target devices is necessary but, unfortunately, not available until run time. Therefore, instead of making assumptions regarding the capabilities of target devices, developers specify performance and resource requirements during the software development stage. The software may then determine at run time whether it can run on the target platform, or, even better, adapt itself to the computing environment by adjusting performance and resource requirements.

1.1.2 Dynamic computing environment

These computing environments are open and dynamic; i.e., usage is not confined to a fixed location, and people carrying computers may join and leave the environment at will. Thus, the environment may change during the execution of an application, leading to problems such as resource variability, system errors, and changing requirements.

To improve system dependability, robustness, and availability, software execution has to be aware of environmental changes and take appropriate actions to accommodate these changes. Traditionally, such auto-adapting mechanisms may be implemented at the code level, e.g., explicit coding of environmental conditions and corresponding remedial actions. However, such approaches are specifically targeted and the adaptive capabilities are often limited in scope, brittle, costly to change, and difficult to reuse. Furthermore, they lack a global view of software systems, so they usually only detect the symptoms but not true sources of environmental changes and thus may not be able to determine the most suitable response.

1.1.3 Resources and information location

Dramatic improvements in the quality of and accessibility to networks makes it increasingly feasible to use collections of networked commodity devices as computational resources. Thus, the widespread use of dynamically-assembled collections of computers, located on local and even wide area networks, as powerful computational resources is becoming possible. For instance, a computational task might be initially mapped to available computers within a workgroup, but then extended or migrated to other resources provided by a commercial computational services provider because of changes in computational characteristics or resource availability. However, locating computational resources or information that are relevant to users' interests can be challenging for ubiquitous computing since it needs to

determine which resource is the best candidate at minimum cost given the heterogeneous, dynamic nature of the resources involved.
For example, searching for devices to constitute an ad-hoc system, a good resource discovery mechanism may need to consider many factors including user interfaces, architectures, or physical locations to minimize the costs of interoperability among participants. In addition, it is preferable that all ad-hoc system participants be in geographic proximity to promote ease of interaction. It is also highly likely that people currently close to the store would have an answer to the query, "is the department store crowed now?". Therefore, a range of physical locations must be incorporated within a resource discovery mechanism; e.g., routing mechanisms using geographic parameters such as Greedy Forwarding (GF) (Finn, 1987) may be used to locate participants and initiate communication among them.

1.2 Possible solutions

Due to the problems described above, the rapid the development of ubiquitous applications usually requires identifying appropriate middleware abstractions and organizing successful protocols, algorithms, and software modules into generic middleware platforms. An ideal platform should allow applications to handle the resource constraints of the ubiquitous devices but, at the same time, exploit their unique features such as availability of location information, embedded sensors, mobility, and spontaneous interaction.

1.2.1 Adaptive systems

To cope with dynamic computing environments, the concept of reflective systems that have the capability to reason and act autonomously was proposed (Maes, 1987). Such a system provides a representation of its own behavior which is amenable to inspection and adaptation, and so is able to observe its current state and alter its behavior at run time. Reflection has been added into various systems, including languages (*Java Platform Standard Edition API Specification*, n.d.), operating systems (Jr & Kofuji, 1996; Yokote, 1992), and middleware (Blair et al., 1998; Kon et al., 2000; Wang & Lee, 1998). These systems allow users to inspect internal states and modify several aspects of implementation, execution, and communication at run time and can adapt themselves flexibly to heterogeneous and dynamic environments.
Many projects have proposed to implement reflection from various perspectives to provide possible solutions for adaptive software development in ubiquitous computing. The paper (da Costa et al., 2008) describes the fundamental issues such as heterogeneity, dependability and security, privacy and trust, mobility, transparent user interaction, etc. A number of software architectures have provided solutions for particular classes of systems and specific domains of concerns to allow users simultaneously interact and collaborate using multiple heterogeneous devices.
For instance, the projects outlined in (Cheng et al., 2006; Garlan et al., 2004; Oreizy et al., 1999) adopt an architecture-based approach in which system architectural models are maintained at run time and used as the basis for system adaptation. External control mechanisms, considered in separable modules, allow system adaptation to become the responsibility of components outside the system which can thus can be analyzed, modified, extended, and reused across different systems. The architectural models usually provide a global view of the system, allowing one to better identify the sources of environmental changes.
BEACH (Tandler, 2001) provides a software infrastructure for synchronous collaboration with many different devices, supporting different forms of interaction and hardware configurations. The layered architecture of Aura (Garlan et al., 2002) can anticipate requests

from a higher layer by observing current demands, and adjust its performance and resource usage characteristics accordingly. HESTIA (Hill et al., 2004) provides a secure infrastructure for ubiquitous computing environments. It addresses the incompatible interoperation problem of securing critical information services in large-scale environments. Analysis of extensive surveys on software infrastructures and frameworks which support the construction of ubiquitous systems is given in (Endres et al., 2005).
Furthermore, many systems (de Lara et al., 2001; Flissi et al., 2005; Gu et al., 2004; Stevenson et al., 2003) adopt component-based software infrastructure for ubiquitous environments, which can take advantage of the exported interfaces and the structured nature of these applications to perform adaptation without modifying the applications. These applications are designed as assemblies of distributed software components and are dynamically discovered according to the end-user's physical location and device capabilities. With this approach, an application can add new behaviors after deployment. In addition, the system will dynamically partition the application and offload some components to a powerful nearby surrogate. This allows for delivery of the application in a ubiquitous computing environment without significant fidelity degradation. McKinley, et al. (Mckinley et al., 2004) provide a review of technologies related to compositional adaptation.

1.2.2 Service composition

An approach similar to that of component-based software systems is to combine multiple primitive programs for a complex task. For example, shell pipes in Unix provide useful data I/O mechanisms, employing multiple programs to work together, and complex tasks are accomplished by coordinating sequences of primitive programs. These programs are "connected" by pipes which facilitate exchange of data among them. Truong and Harwood (Truong & Harwood, 2003) extended this concept and proposed a shell that provides distributed computing over a peer-to-peer network and is characterized by good scalability.
Another related topic is the composition of web services. Programmers may use the Web Services Description Language (WSDL) to specify characteristics and access of web services, and thus web services can be composed to provide a complicated service. However, WSDL does not support semantic descriptions; thus, a composition always requires intervention by a programmer. To enable web services to perform a dynamic composition by themselves, Martin et al. (Martin et al., 2004) proposed the OWL-S approach, in which a client program and web services may have a common consensus on the semantics of the terms in WSDL by a third-party ontology, and thus a web service can automatically interact with another web service by a priori setting the rule for the semantics. Mokhtar et al. (Mokhtar et al., 2007) extended the OWL-S approach and proposed a conversation-based service composition method named COCOA that aims for the dynamic composition of services to complete a user task. With COCOA, a service as well as a user task is transformed to an automata, and an algorithm is proposed to combine the automata of different services.
QoS-aware composition is another important issue of web service composition. For example, Li et al. (Li et al., 2001) propose a hierarchical adaptive QoS architecture for multimedia applications. A multimedia service is delivered by multiple service configurations, each of which involves a different set of service components. Each service component is executed as a process. Components cooperate through protocols over network communication. Usually, the composition of web services needs to satisfy given optimization criteria, such as the overall cost or response time, and can be formulated as a NP-hard optimization problem (Canfora et al., 2005). Canfora et al. (Canfora et al., 2005) proposed a genetic algorithm for the NP-hard

QoS-aware composition problem. In addition, Wada et al. (Wada et al., 2008) proposed a multi-objective genetic algorithm to deal with optimization criteria with trade-offs, and Berbner et al. (Berbner et al., 2006) proposed a fast heuristic that was 99% close to optimal solutions in most cases. Furthermore, various middleware and frameworks (Issa et al., 2006; Yu & Lin, 2005; Zeng et al., 2004) have been proposed to realize QoS-aware web service compositions.

1.2.3 Resource discovery

As mentioned earlier, design of such a resource discovery mechanism becomes increasingly difficult under ubiquitous computing conditions in which useful information servers are not known a priori. Porter and Sen (Porter & Sen, 2007) classified two approaches for resource discovery mechanisms, namely referral (Candale & Sen, 2005; Sen & Sajja, 2002; Singh et al., 2001; Yolum & Singh, 2005) and matchmaker (Albrecht et al., 2008; Iamnitchi & Foster, 2004; Jha et al., 1998; Ogston & Vassiliadis, 2001). In the referral approach, the resource providers provide both services and referrals to other providers. Providers which provide high quality services are likely to be recommended by many providers. Providers must, however, ascertain the trustworthiness and expertise of other providers to measure the value of a recommendation. For example, in (Candale & Sen, 2005), the performance of a provider is measured by the satisfaction obtained by its clients. This mechanism requires learning both the performance levels of different service providers as well as the quality of referrals provided by other providers by exchanging information.

Another possible solution to this problem is to use a matchmaker: a dedicated resource discovery server that arranges the connections. Assuming clients are truthful in their interactions with the matchmaker, optimal matches can be found. For example, in the SWORD architecture (Albrecht et al., 2008), a resource query will be processed by a distributed query processor to find candidate nodes whose characteristics match the specified requirements. Then, the optimized subset of the candidate nodes will be determined by the optimizer component accounting for desired device characteristics, such as load and network location, and inter-device characteristics, such as latency and bandwidth.

Furthermore, several efforts (Albrecht et al., 2008; Balazinska et al., 2002; Huang & Steenkiste, 2003) have explored resource discovery mechanisms in large-scale environments. The system must scale to thousands of devices and be highly available. It also has to support high rates of measurement updates from participating devices, from static characteristics such as operating system, processor speed, and network coordinates to more dynamic characteristics such as available CPU capacity, memory, and disk storage.

1.3 Organization of this article

The remainder of this article is organized as follows. Section 2 describes the idea of ad-hoc systems in detail and how it may be realized by the adaptive software framework, FRAME, implemented as part of the adaptive software architecture project, ASAP. Under FRAME, software components can be discovered, loaded, combined, adapted, and executed on the target platforms in accordance with available resources and performance constraints. Software may have a list of specifications, specified during the development stage, to gather information about its environment. During execution, the software may check the list for environmental changes, and then respond accordingly. Therefore, an ad-hoc system can be constructed and executed without human intervention.

Section 3 introduces the Distributed Shell System, or DISHES, in which a mobile user can issue a shell script of a task, and DISHES will automatically locate the required programs and retrieve the necessary data. The required programs will be dispatched and executed on their host computers. Intermediate results will be piped between the host computer through networks, and final result will be I/O redirected to the user-specified location.
In Sec. 4, we present the basic idea of mutual assistant networks (MANs) which combine social networks and wireless sensor networks (WSNs) to query local and timely information. The proposed infrastructure uses routing protocols commonly adopted in WSNs, such as GF, to forward the query to someone who may have an answer. Conceptually, people in MANs serve the role played by sensors in WSNs; they may accept queries from others, gather information based on queries, and then respond. Such a new social network application will promote the sharing of knowledge and bring people closer together. Finally, a summary is given in Sec. 5.

2. Ad-Hoc systems

2.1 Overview

As mentioned earlier, the ubiquity of computers makes it possible to combine several resource-limited devices as an ad-hoc system to complete a complex computing task. Imagine a scenario in which a user watches a movie on his smart phone. Referring to Fig. 1, to completely execute on the smart phone, the media player software needs to decoding the multimedia stream, output video and audio, and interact with the user. Due to limited computing capability, the performance or quality of the video and audio may be unacceptably poor. In addition, the small size of the phone may lead to an unpleasant interaction experience.

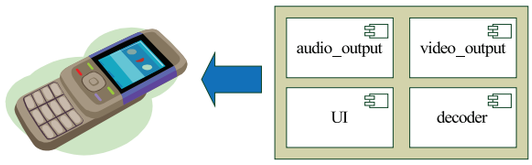


Fig. 1. The media player software needs to decode the multimedia stream, output video and audio, and interact with the user. Due to limited computing capability, the performance or quality of the video and audio may be unacceptably poor.

Alternatively, the user may search ambient devices for their hardware features. For example, he may find a TV for its big LCD display, a Hi-Fi audio system for its stereo sound quality, and a PC for its computing power. He can connect these devices together to form an ad-hoc system as shown in Fig. 2. After the media player software is launched, the appropriate part of the code will be distributed to each device, i.e., the code for audio processing to the Hi-Fi audio system, the code for video processing to the TV, and the code for decoding the multimedia stream to the PC. As a consequence, instead of watching the movie on the smart phone, the user may enjoy the smoother movie on the ad-hoc system with a larger image on the TV and better sound from the Hi-Fi audio system.
One challenge to realizing ad-hoc systems is the diversity of participating devices. It is impossible to know the performance of components on each device in advance to determine the appropriate component distribution. In Fig. 2, before distributing the video processing component to the TV, one must first know if the TV has the appropriate resource for video processing. Such performance information can only be known after the ad-hoc system is formed. Manually probing the performance of each participating device is difficult for the

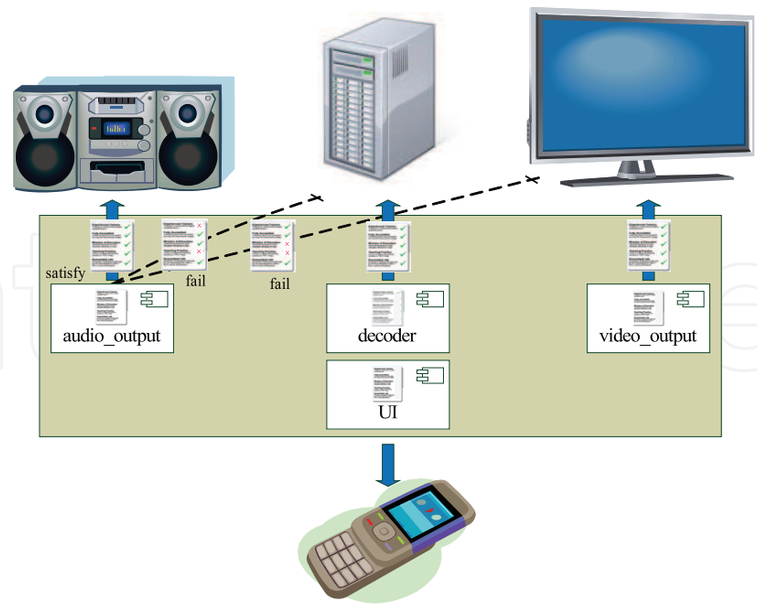


Fig. 2. The media player software executes on an ad-hoc system consisting of a smart phone, a TV, a Hi-Fi audio system, and a PC. The components are distributed to the appropriate participating devices in which all specifications are satisfied for a better movie watching experience.

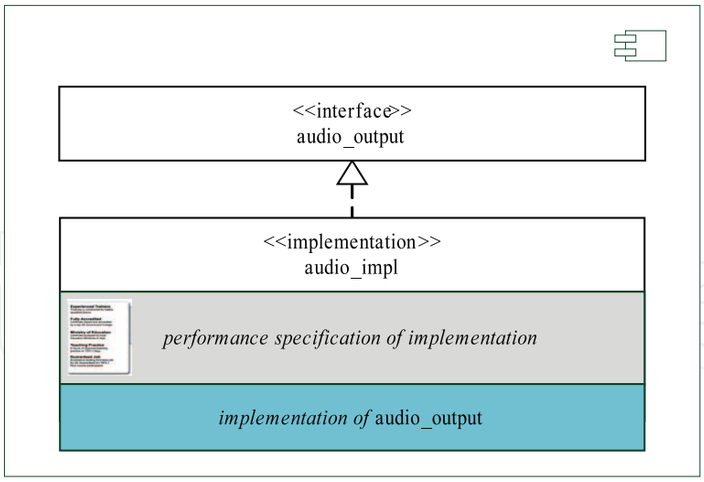
average user, but the adaptive software framework, FRAME (Ko & Mutka, 2002), may provide a better solution to alleviate this configuration problem.

2.2 The adaptive software framework, FRAME

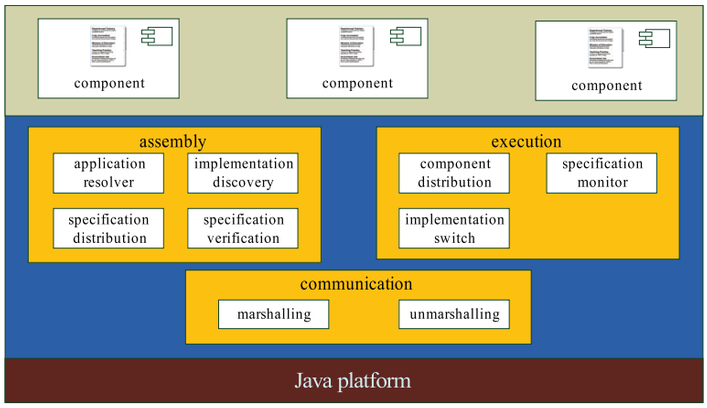
In many mass production industries, such as automobiles and electronics, final products are assembled from parts. The parts may be built by various vendors, but they are plug-in compatible if they have the required functionality. It is technically impossible for vendors to develop parts that may work perfectly in every environment. As a consequence, vendors usually specify how well the parts may perform in certain environments, and users can select appropriate parts based on these specifications. If, under certain conditions, a part fails or does not perform as required, it can be replaced with an appropriate part. For example, regular tires are designed for normal weather conditions, but may be prone to skidding on snow and thus may not meet safety requirements. Consequently, special snow tires may be used to achieve better safety.

A similar idea was adopted in FRAME. In addition to function implementation as normal components, specifications for required resources were added as shown in Fig. 3(a). The specifications allow FRAME to identify an appropriate participating device for execution.

FRAME is a middleware which provides APIs for Java applications to adapt themselves to heterogeneous environments. Figure 3 illustrates the component structure and its middleware architecture. A component provides abstract function interfaces without exposing detailed implementations. Similar to components in the automobile and electronic industries, a component may have multiple implementations developed by various vendors. Different implementations may require different resources and produce different quality of results; these are specified as specifications. The components of an application are not linked during the development stage, but are "assembled" at run time after determining the resources



(a) Component



(b) Middleware architecture

Fig. 3. The adaptive software framework, FRAME consists of three modules, namely assembly, execution, and communication. In addition to function implementation, specifications for required resources are added for components in FRAME.

of the computing environment with their specifications. If the application fails to work appropriately because of dynamic environmental changes, it may also use the FRAME APIs to replace component implementations for better performance or quality of execution without down-time. In summary, FRAME provides the following features:

- Developers may specify specifications for component implementations.
- Application components may be automatically distributed to single or multiple participating devices.
- Before execution, FRAME may probe the available resources from the computing environment and then adapt themselves to the computing environment via a special process called *assembly*.
- During execution, FRAME may detect run-time environmental changes and, if necessary, invoke the assembly process without down-time.

Referring to Fig. 3(b), these features are implemented as three modules, namely assembly, execution, and communication. The assembly module resolves the components of an application and discovers all possible component implementations, then distributes and verifies the implementation specifications for participating devices to determine the appropriate execution devices. The execution module distributes each component to the selected execution device following the assembly process. It also monitors the specifications during run-time and, if necessary, invokes the assembly process for more appropriate implementations without down-time. The communication module provides necessary data marshalling/unmarshalling mechanisms for cooperation among components on different devices.

The assembly process is worthy of special attention. The traditional approach to component implementation selection is to use condition statements such as `if-else` statements as shown in Table 1. There may be nested if-else statements and each is used to decide the appropriate implementation of a component. Once an implementation is selected, execution flow may go into the inner `if-else` statements to select the appropriate implementation of other components. However, from a software engineering perspective, the condition statements approach is primitive. As the numbers of components and their implementations increase, the code tends toward so-called "spaghetti code" that has a complex and tangled control structure and the software will become more difficult to maintain or modify. The most important limitation of this approach is that condition statements are hard-coded, so the availability of all implementations needs to be known during the development stage. It is impossible to integrate newly-developed implementations without rewriting and recompiling the code, and, of course, the down-time.

```
if (constraints of component 1 with implementation 1)
{ // select component 1 with implementation 1

  if (constraints of component 2 with implementation 1)
  { // select component 2 with implementation 1

    // check each implementation of component 3, 4,...
  }
  else if (constraints of component 2 with implementation 2)
  { // select component 2 with implementation 2

    // check each implementation of component 3, 4,...
  }
  ... // more else if blocks for other implementations of component 2
}
else if (constraints of component 1 with implementation 2)
{ // select component 1 with implementation 2

  // similar as the code in the if block of
  // component 1 with implementation 1
}
... // more else if blocks for other implementations of component 1
```

Table 1. `if-else` statement structure for the component implementation selection

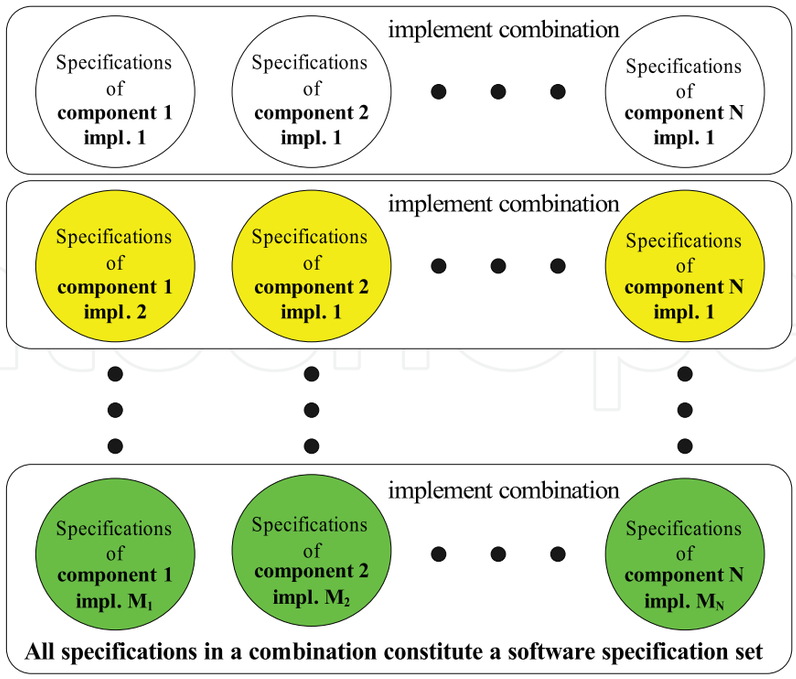


Fig. 4. All possible implementation combinations of an application: each row represents a possible implementation combination and the constraints of a combination constitute a software constraint set.

FRAME uses a different approach than condition statements to select appropriate component implementations. By identifying all component implementations of an application via service discovery, the assembly module builds all possible combinations of the application as shown in Fig. 4. Each combination has a set of specifications, called a *software specification set*, which consists of all the specifications from the involved component implementations. Since no two implementations should have same set of specifications, the mapping between combinations and software specification sets is one-to-one. By solving which specification set is *feasible*, i.e., all specifications in the software specification set are satisfied, the corresponding feasible combination will be found.

To realize ad-hoc systems, the assembly module distributes components to participating devices prior to constructing the software specification sets. There might be more than one possible distribution of components, and we call each possibility a *distribution*. For each distribution, the assembly module constructs all possible software specification sets and determine whether a feasible specification set exists. A distribution is *feasible* if it has a feasible software specification set, and then the application is assembled from the feasible distribution. In other words, the assembly process for an ad-hoc system application is to find a feasible distribution from all possible distributions. For the example of the media player software in Fig. 2, there may be an implementation for the audio component with good sound quality, and all its specifications are satisfied on the Hi-Fi audio system, but not the PC and the TV. Thus the audio component implementation will be executed on the Hi-Fi audio system. As a consequence, an ad-hoc system for the media player is constituted without human intervention.

3. Distributed Shell System

This section introduces another approach for accomplishing a resource intensive task by using ambient computing resources. The approach is based on the concept that many single machine systems provide a command line interface (e.g., shell) which allows a user to issue a command consisting of many primitive programs to accomplish a complex task. These programs are coordinated by the *pipe* mechanism and the result may be stored as a file via the *I/O redirection*. DISHES (DIstributed SHEll System) (Lai & Ko, 2010) extends this idea to ubiquitous computing environments, in which a mobile device user can issue a command specifying the data location and a sequence of programs to process the data. When a mobile device receives the command, it will seek out appropriate ambient devices which have the required programs and send tasks-to-do to these devices. Each device may retrieve the data from the specified location and execute the designated program to process the data. Once finished, it will send the result to other devices for further processing or back to the user's mobile device. Thus, a complicated task can be achieved by the sequential cooperation of multiple primitive programs.

Imagine a scenario in which a student in a school library is looking for research literature with the keywords "ubiquitous computing" in descending order of the year published. He may issue the following command with his smart phone:

```
grep ``ubiquitous computing'' http://myschool.edu.tw/reference_list |
sort -k 2,2 -r > sorting_result
```

Based on the command, the student's smart phone will find devices providing the required programs (`grep` and `sort`). Suppose computer `A` provides `grep` and computer `B` provides `sort`, as depicted in Fig. 5. Then `A` will retrieve the client information from the specified data location, `http://myschool.edu.tw/reference_list`, execute `grep "ubiquitous computing"` to pick up the literature containing the phrase "`ubiquitous computing`", and send the intermediate result to `B` via the pipe. After receiving the intermediate result from `A`, `B` will execute `sort -k 2,2 -r` to sort the literature by publication year. The final sorted result will be sent back to the student's smart phone and stored as the file, `sorting_result`, via the I/O redirection.

In the above example, the student only specifies the data location and a sequence of programs. DISHES will automatically seek out appropriate computers with the required programs to process the data retrieved from the specified location. Moreover, a complicated task may be accomplished by gluing multiple primitive programs (`grep` and `sort`). These primitive programs do not have to be stored in or executed on the student's smart phone. They can be executed on the other computers (`A` and `B`) for better performance, and the results will be returned to the user. With this approach, the hardware and software of a mobile device may be kept as simple as possible, allowing the device volume, weight, and cost to be minimized. Thus, DISHES boosts people's mobility. Besides, gluing together multiple primitive programs to perform a complicated task can reduce software development costs. Note that though both DISHES and FRAME use ambient computing resources to execute resource intensive tasks, their fundamental purposes and approaches are different. DISHES tries to reuse the software that has been well established on single machine systems without making any modifications. There is no software migration involved in DISHES and the shared resources are software oriented; that is, users look for ambient computing resources completely from a software perspective. On the other hand, when building an ad-hoc system, users select a device based

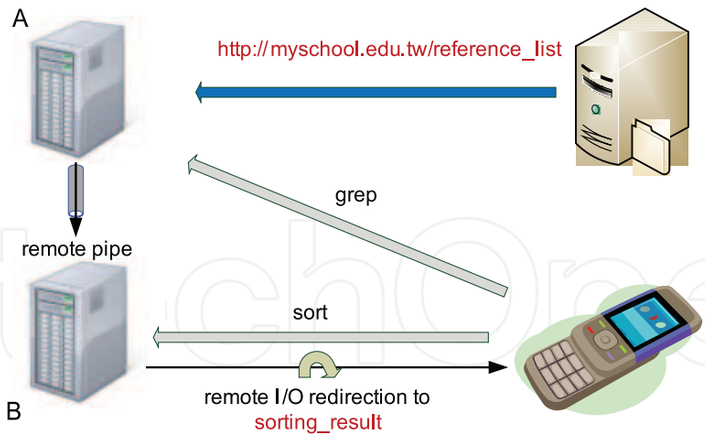


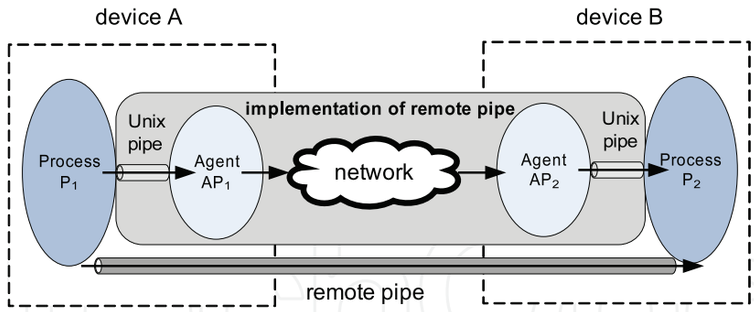
Fig. 5. The student issues the command with his smart phone to find research literature with the keyword "ubiquitous computing". DISHES finds computers providing the specified programs, and coordinates their execution via the remote pipe and I/O redirection mechanisms.

on the hardware features it provides. For example, in Fig. 2, the user may consider adding the TV into the ad-hoc system for the multimedia application because of its big LCD display.

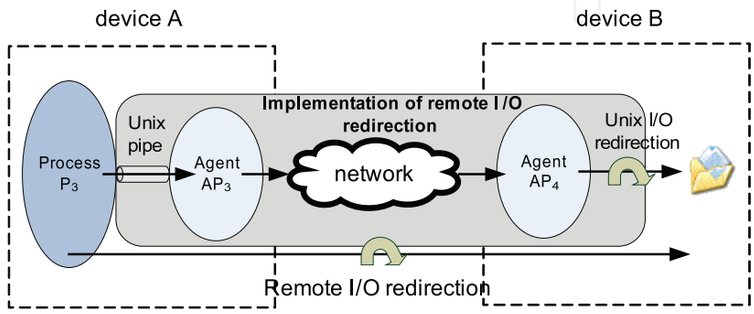
The main feature of DISHES is the implementation of the remote version of the pipe and I/O redirection without any modification to the original programs. The remote pipe allows the standard output of one process to be fed directly as the standard input to another over networks. It may be realized by the assistance of two agent processes as depicted in Fig. 6(a). For example, to construct a remote pipe from process P_1 of device A to process P_2 of device B, two agent processes, AP_1 on A and AP_2 on B, are created with two regular UNIX pipes, from P_1 to AP_1 and AP_2 to P_2. Moreover, a socket connection between AP_1 and AP_2 is constructed. Thus, AP_1 may receive the output of P_1 from the regular UNIX pipe and relay it to AP_2 via the socket connection, and then AP_2 may feed the output to P_2 via the other regular UNIX pipe. Consequently, we have a remote pipe from P_1 to P_2. Note that the tasks of the agent processes AP_1 and AP_2 are to relay the information, regardless of the output of P_1. Therefore, the remote pipe is realized without any modification to the original programs.

Similar to the remote pipe, two agent processes, AP_3 on A and AP_4 on B, are needed for remote I/O redirection, as illustrated in Fig. 6(b). The output of P_3 is fed to AP_3 via a regular UNIX pipe, and then to AP_4 via a socket connection. Finally, the output is saved to file via a regular UNIX I/O redirection mechanism.

Traditional shells on single machine systems use an explicit specified list of directories (e.g., via the environment variable PATH under UNIX) for searching programs by name. However, due to the dynamic characteristics of ubiquitous computing environments, program locations are usually unknown in advance. Therefore, similar to the assembly module in FRAME, service discovery mechanisms may be incorporated to search programs in an unfamiliar environment. In addition, the intermediate results of one computer may be transmitted to another computer for subsequent program execution via the network, so performance optimization may need to be considered during the service discovery (e.g., to find a sequence of computers so that the total communication time for transmitting the intermediate results is minimized). The problem can be formulated as a minimum sequential workflow problem in static environments and solved by a polynomial algorithm (Lai & Ko, 2010). However, it



(a) remote pipe



(b) remote I/O redirection

Fig. 6. Implementation of remote pipe and I/O redirection. Agents are needed to relay information between processes on different devices.

becomes an on-line problem under ubiquitous computing environments and requires further study.

4. Mutual Assistant Networks

In this section, we introduce the concept of integrating social networks which connect people and ubiquitous computing which connects computers, allowing people to use not only ambient computing resources, but also human resources. For example, a user may want to know if a festival is worth going to before setting out. Such information is timely; it becomes useless after the festival ends and thus must be answered by someone who is currently near the festival. Given the ubiquitous existence of people carrying network-connected devices, finding people close to the festival who are willing to help is quite feasible. Thus, the objective of mutual assistant networks (MANs) is to provide necessary mechanisms to bridge the user and these people. Referring to Fig. 7, a MAN will forward the user's question about the festival to people near the festival to collect their opinions. Similar to WSNs, a MAN may analyze and aggregate the data, and then return the results to the user.

The features of MAN applications and WSN applications are similar in some aspects. WSNs can collect information about physical environments from sensors while MANs collect knowledge or opinions from people. Both can use geographical parameters to specify whether data is collected from sensors or people. Therefore, many proposed WSN infrastructures may also be adopted for MANs, such as geographic routing, network configuration and coordination, data dissemination, and data aggregation, to name just a few.

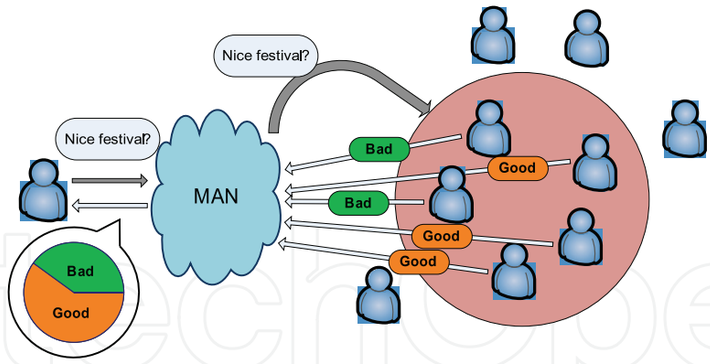


Fig. 7. A MAN can forward a user's question about the festival to the people close to the event and collect their responses. The MAN then analyzes and aggregates the data, and then returns the results to the user.

The current prototype has been implemented on the Android platform, with the architecture illustrated in Fig. 8. It provides APIs for MAN applications implemented into four modules. The user profile module manages user information under MAN, including identity. It also maintains a credit system to encourage users to share knowledge and a reputation system to track the validity of information that users have provided depending on feedback ratings provided by questioners. The networking module can deliver messages to destinations via routing techniques using geographic parameters such as GF (Finn, 1987) and GPSR (Karp & Kung, 2000). The data processing module provides various functions for manipulating information collected from people, including statistics, indexing, aggregation, and dissemination. The sensor module is basically an abstraction layer allowing applications to access the various hardware sensors present in devices.

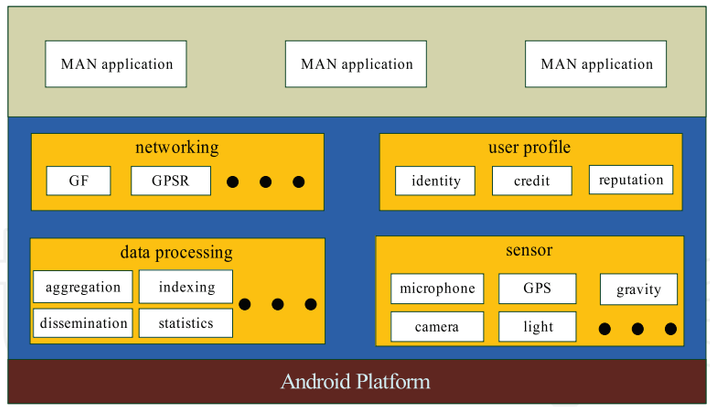


Fig. 8. The architecture of the current MAN prototype. MAN applications are developed through four modules, namely networking, user profile, data processing, and sensor.

Several social networking projects have been prosed to connect people, including WhozThat (Beach et al., 2008), SocialFusion (Beach et al., 2010), MoSoSo (Tsai et al., 2009), and Micro-Blog (Gaonkar et al., 2008). SocialFusion is a system capable of systematically integrating diverse data streams including sensors for mobile social networks that enable new context-aware applications such as context-aware video screens and mobile health applications. Note that one conceptual difference between the MAN and SocialFusion is that MAN uses people as sensors. As many of the infrastructure design considerations for

MANs originate from WSNs, the infrastructure can handle thousands or millions of nodes, i.e., people, spread over a large area. Many of constraints and challenges facing WSNs are also found in MANs. For example, a MAN is more expansive and dynamic than the current TCP/IP network and may create new types of traffic patterns that are quite different from the conventional Internet and raise demand for new approaches to minimize the amount and range of communication through local collaboration, such as aggregation or duplicate data suppression. How, where, and what information is generated and consumed by users will affect the way information is compressed, routed, and aggregated. Furthermore, MANs connect people who may not have had any prior social interaction, so it is more appropriate to address people by physical properties, such as location or proximity, than by names or IP addresses. There is also a need for advanced query interfaces and resource discovery engines to effectively support user-level functions.

MAN is still a developing project that requires further improvements. The goal of the architecture design is to provide service abstractions as a base for the development of new applications. The modular architecture design allows new protocols or algorithms, along with the integration of third party services; e.g., Google Maps for specifying geographic parameters.

5. Conclusion

This article illustrates three possible approaches for people to access the computing and information resources from ambient devices and other people anytime and anywhere. FRAME can realize the concept of ad-hoc systems in which a user may utilize hardware features from the computers nearby for better performance and interfaces. DISHES allows a user to coordinate the execution of a sequence of programs located on different devices without modifications. MANs provides an infrastructure for new location-aware social network applications, in which people may share the local and timely information that cannot be obtained in time from the Internet. With the ubiquitous existence of computers, there is no need to carry excess software and hardware, and thus people's mobility will be boosted. In addition, the ubiquitous existence of people carrying mobile devices may promote sharing of knowledge and thus extend the senses of human beings to a normally inaccessible locations via knowledge sharing.

6. References

Albrecht, J., Oppenheimer, D., Vahdat, A. & Patterson, D. A. (2008). Design and Implementation Tradeoffs for Wide-Area Resource Discovery, *ACM Transactions on Internet Technology* 8(4): 1–44.

Balazinska, M., Balakrishnan, H. & Karger, D. (2002). INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery, *Proceedings of the First International Conference on Pervasive Computing*, Springer-Verlag, Zurich, Switzerland, pp. 195–210.

Beach, A., Gartrell, M., Akkala, S., Elston, J., Kelley, J., Nishimoto, K., Ray, B., Razgulin, S., Sundaresan, K., Surendar, B., Terada, M. & Han, R. (2008). WhozThat? Evolving an Ecosystem for Context-Aware Mobile Social Networks, *IEEE Network* 22(4): 50–55.

Beach, A., Gartrell, M., Xing, X., Han, R., Lv, Q., Mishra, S. & Seada, K. (2010). Fusing Mobile, Sensor, and Social Data To Fully Enable Context-Aware Computing, *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, ACM, Annapolis, Maryland, pp. 60–65.

Berbner, R., Spahn, M., Repp, N., Heckmann, O. & Steinmetz, R. (2006). Heuristics for QoS-aware Web Service Composition, *Proceedings of the IEEE International Conference on Web Services*, IEEE Computer Society, Washington, DC, USA, pp. 72–82.

Blair, G. S., Coulson, G., Robin, P. & Papathomas, M. (1998). An Architecture for Next Generation Middleware, *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer-Verlag, London.

Candale, T. & Sen, S. (2005). Effect of referrals on convergence to satisficing distributions, *Proceeding of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, Utrecht, The Netherlands, pp. 347–354.

Canfora, G., Di Penta, M., Esposito, R. & Villani, M. L. (2005). An Approach for QoS-aware Service Composition based on Genetic Algorithms, *Proceedings of the 2005 conference on Genetic and Evolutionary Computation*, ACM, Washington DC, USA, pp. 1069–1075.

Cheng, S.-W., Garlan, D. & Schmerl, B. (2006). Architecture-based Self-Adaptation in the Presence of Multiple Objectives, *Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems*, ACM, New York, NY, USA, pp. 2–8.

da Costa, C. A., Yamin, A. C. & Geyer, C. F. R. (2008). Toward a General Software Infrastructure for Ubiquitous Computing, *IEEE Pervasive Computing* 7(1): 64–73.

de Lara, E., Wallach, D. S. & Zwaenepoel, W. (2001). Puppeteer: Component-based Adaptation for Mobile Computing, *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, California.

Endres, C., Butz, A. & MacWilliams, A. (2005). A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing, *Mobile Information Systems* 1(1): 41–80.

Finn, G. G. (1987). Routing and Addressing Problems in Large Metropolitan-Scale Internetworks, *Research ISI/RR-87-180*, Information Sciences Institute.

Flissi, A., Gransart, C. & Merle, P. (2005). A Component-based Software Infrastructure for Ubiquitous Computing, *Proceedings of the 4th International Symposium on Parallel and Distributed Computing*, IEEE Computer Society, Washington, DC, USA, pp. 183–190.

Gaonkar, S., Li, J., Choudhury, R. R., Cox, L. & Schmidt, A. (2008). Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation, *Proceeding of the 6th International Conference on Mobile Systems, Applications and Services*, ACM, Breckenridge, CO, USA, pp. 174–186.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B. & Steenkiste, P. (2004). Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure, *Computer* 37(10): 46–54.

Garlan, D., Siewiorek, D., Smailagic, A. & Steenkiste, P. (2002). Project Aura: Toward Distraction-Free Pervasive Computing, *IEEE Pervasive Computing* 1(2): 22–31.

Gu, X., Messer, A., Greenberg, I., Milojicic, D. & Nahrstedt, K. (2004). Adaptive Offloading for Pervasive Computing, *IEEE Pervasive Computing* 3(3): 66–73.

Hill, R., Al-Muhtadi, J., Campbell, R., Kapadia, A., Naldurg, P. & Ranganathan, A. (2004). A Middleware Architecture for Securing Ubiquitous Computing Cyber Infrastructures, *IEEE Distributed Systems Online* 5(9).

Huang, A.-C. & Steenkiste, P. (2003). Network-Sensitive Service Discovery, *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, USENIX Association, Seattle, WA.

Iamnitchi, A. & Foster, I. (2004). A Peer-to-Peer Approach to Resource Location in Grid Environments , *Grid Resource Management: State of the Art and Future Trends*, Kluwer Academic Publishers, Norwell, MA, USA, pp. 413–429.

Issa, H., Assi, C. & Debbabi, M. (2006). QoS-Aware Middleware for Web Services Composition - A Qualitative Approach, *Proceedings of the 11th IEEE Symposium on Computers and Communications*, IEEE Computer Society, Washington, DC, USA, pp. 359–364.

Java Platform Standard Edition API Specification (n.d.).
URL: *http://download.oracle.com/javase/6/docs/api/*

Jha, S., Chalasani, P., Shehory, O. & Sycara, K. (1998). A Formal Treatment of Distributed Matchmaking, *Proceedings of the second International Conference on Autonomous Agents*, ACM, Minneapolis, Minnesota, United States, pp. 457–458.

Jr, J. M. A. & Kofuji, S. T. (1996). Bootstrapping the Object Oriented Operating System Merlin: Just Add Reflection, *in* C. Zimmerman (ed.), *Advances in Object-Oriented Metalevel Architectures and Reflection*, CRC Press, chapter 5.

Karp, B. & Kung, H. T. (2000). GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ACM, Boston, MA, US, pp. 243–254.

Ko, R.-S., Lai, C.-C., Yen, C.-K. & Mutka, M. W. (2008). Component-Based Ad Hoc Systems for Ubiquitous Computing, *International Journal of Pervasive Computing and Communications Special Issue on Towards merging Grid and Pervasive Computing* 4(4): 333–353.

Ko, R.-S. & Mutka, M. W. (2002). A Component-Based Approach for Adaptive Soft Real-Time Java within Heterogeneous Environments, *A special issue of Parallel and Distributed Real-Time Systems* 5(1): 89–104.

Kon, F., Román, M., Liu, P., Mao, J., Yamane, T., MagalhÃčes, L. C. & Campbell, R. H. (2000). Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB, *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, number 1795 in *LNCS*, Springer-Verlag, New York, pp. 121–143.

Lai, C.-C. & Ko, R.-S. (2010). DISHES: A Distributed Shell System Designed for Ubiquitous Computing Environment, *International Journal of Computer Networks & Communications* 2(1): 66–83.

Li, B., Kalter, W. & Nahrstedt, K. (2001). A Hierarchical Quality of Service Control Architecture for Configurable Multimedia Applications, *Journal of High Speed Networks, Special Issue on Management of Multimedia Networking*, IOS Press.

Maes, P. (1987). *Computational Reflection*, PhD thesis, Laboratory for Artificial Intelligence, Vrije Universiteit Brussel, Brussels, Belgium.

Martin, D. L., Paolucci, M., McIlraith, S. A., Burstein, M. H., McDermott, D. V., McGuinness, D. L., Parsia, B., Payne, T. R., Sabou, M., Solanki, M., Srinivasan, N. & Sycara, K. P. (2004). Bringing Semantics to Web Services: The OWL-S Approach, *Proceedings of the first International Workshop on Semantic Web Services and Web Process Composition*, San Diego, CA, USA, pp. 26–42.

Mckinley, P. K., Sadjadi, S. M., Kasten, E. P. & Cheng, B. H. (2004). Composing Adaptive Software, *IEEE Computer* 37(7).

Mokhtar, S. B., Georgantas, N. & Issarny, V. (2007). COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support, *Journal of Systems and Software* 80(12): 1941–1955.

Ogston, E. & Vassiliadis, S. (2001). Matchmaking Among Minimal Agents Without a Facilitator, *Proceedings of the 5th International Conference on Autonomous Agents*, ACM, Montreal, Quebec, Canada, pp. 608–615.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S. & Wolf, A. L. (1999). An Architecture-Based Approach to Self-Adaptive Software, *IEEE Intelligent Systems* 14(3): 54–62.

Porter, J. & Sen, S. (2007). Searching for Collaborators in Agent Networks, *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, IEEE Computer Society, Washington, DC, USA, pp. 508–511.

Sen, S. & Sajja, N. (2002). Robustness of Reputation-based Trust: Boolean Case, *Proceeding of the first International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM, Bologna, Italy, pp. 288–293.

Singh, M. P., Yu, B. & Venkatraman, M. (2001). Community-based service location, *Communications of the ACM* 44(4): 49–54.

Stevenson, G., Nixon, P. & Ferguson, R. I. (2003). A General Purpose Programming Framework for Ubiquitous Computing Environments, *System Support for Ubiquitous Computing Workshop*, Seattle, USA.

Tandler, P. (2001). Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices, *Proceedings of the 3rd international conference on Ubiquitous Computing*, Springer-Verlag, London, UK, pp. 96–115.

Truong, M. T. & Harwood, A. (2003). Distributed Shell over Peer-to-Peer Networks, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, USA, pp. 269–278.

Tsai, F. S., Han, W., Xu, J. & Chua, H. C. (2009). Design and development of a mobile peer-to-peer social networking application, *Expert Systems with Applications* 36(8): 11077–11087.

Wada, H., Champrasert, P., Suzuki, J. & Oba, K. (2008). Multiobjective Optimization of SLA-Aware Service Composition, *Proceedings of the 2008 IEEE Congress on Services - Part I*, IEEE Computer Society, Washington, DC, USA, pp. 368–375.

Wang, Y.-M. & Lee, W.-J. (1998). COMERA: COM Extensible Remoting Architecture, *Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, USENIX, pp. 79–88.

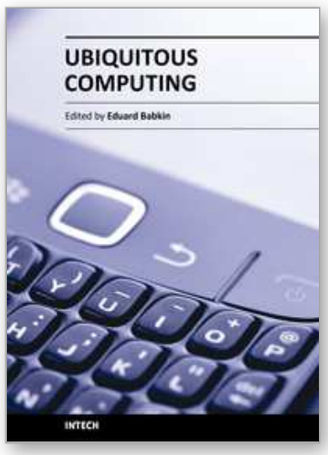
Weiser, M. (1991). The Computer for the 21st Century, *Scientific American* 265(3): 66–75. Reprinted in IEEE Pervasive Computing, Jan-Mar 2002, pp. 19-25.

Yokote, Y. (1992). The Apertos Reflective Operating System: The Concept and Its Implementation, *in* A. Paepcke (ed.), *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, Vol. 27, ACM Press, New York, NY, pp. 414–434.

Yolum, P. & Singh, M. P. (2005). Engineering Self-Organizing Referral Networks for Trustworthy Service Selection, *IEEE Transactions on Systems, Man and Cybernetics, Part A* 35(3): 396–407.

Yu, T. & Lin, K.-J. (2005). A Broker-Based Framework for QoS-Aware Web Service Composition, *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, IEEE Computer Society, Washington, DC, USA, pp. 22–29.

Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J. & Chang, H. (2004). QoS-Aware Middleware for Web Services Composition, *IEEE Transactions on Software Engineering* 30(5): 311–327.



Ubiquitous Computing

Edited by Prof. Eduard Babkin

ISBN 978-953-307-409-2
Hard cover, 248 pages
Publisher InTech
Published online 10, February, 2011
Published in print edition February, 2011

The aim of this book is to give a treatment of the actively developed domain of Ubiquitous computing. Originally proposed by Mark D. Weiser, the concept of Ubiquitous computing enables a real-time global sensing, context-aware informational retrieval, multi-modal interaction with the user and enhanced visualization capabilities. In effect, Ubiquitous computing environments give extremely new and futuristic abilities to look at and interact with our habitat at any time and from anywhere. In that domain, researchers are confronted with many foundational, technological and engineering issues which were not known before. Detailed cross-disciplinary coverage of these issues is really needed today for further progress and widening of application range. This book collects twelve original works of researchers from eleven countries, which are clustered into four sections: Foundations, Security and Privacy, Integration and Middleware, Practical Applications.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen