A Multi-Agent Framework for Ambient Systems Development

Xiping Hu, Weichang Du, Bruce Spencer *

*Faculty of Computer Science, University of New Brunswick, Fredericton, E3B5A3, Canada

Institute for Information Technology, National Research Council Canada, Fredericton, E3B9W4, Canada

Abstract

This paper proposes a multi-agent framework to support applications development of ambient systems. In the programming model of the framework, ambient systems can be developed by collaboration of mobile agents and service (or resident) agents, where resident agents provide application services on devices and mobile agents provide communication services on behalf of owner applications. On top of device infrastructure, the architecture of the framework consists of three layers: framework service layer, software agents layer and application layer, to fully support dynamic and collaborative tasks of ambient systems.

Keywords: ambient systems; software agent; service; ambient application; application development

1. Introduction

An ambient system is a ubiquitous environment that establishes a mechanism to provide the users with all the functionality of the devices, components and local and distributed software applications in a flexible, integrated and almost transparent way for the end-user [1]. We often find that when we try to execute some tasks in our mobile devices they are not capable of doing the tasks in isolation. If these devices are not capable of offering the expected or desired result, it might be possible that this result might be obtained by the collaboration among different devices [2, 3]. Meanwhile, ubiquitous computing tries to get away from the paradigm of the user in front of a PC as the unique interaction mechanism with software components and promotes the use of mobile and embedded devices [4]. Thus, many of the current applications meet with challenges arising from the dynamic and mobile nature of ambient systems, such as the increasing number of mobile devices, limitation of sources, unstable network connectivity, etc. Therefore, compared to conventional systems which just offer specific and concrete functions, the main features of ambient system include the following: it can provide sufficient and automatic services to end-users, it has real collaboration capability among the devices, and ambient applications can self-adapt to a dynamically changing environment.

Software agent can be seen as a composition of software and data in mobile devices which is able to work autonomously. It can also transport its state from one environment to another with its data intact, and be capable of...
performing appropriately in the new environment. When an agent decides to move, it can save its current state, then transfer this saved state to the new host, and recover from the saved state [5]. Mobile agents are software entities that act on behalf of their creators and move independently between hosts [6]. Meanwhile, after being dispatched, mobile agents become independent of the process that created them and can operate asynchronously, react dynamically and autonomously to environmental changes, and require little interaction with users. At the same time, software agents have the capability to coordinate among themselves, resulting in collaborating services and resources among the devices in ambient systems. Consequently, software agents are very suitable to execute applications for ambient systems.

The development steps of an ambient system involve obtaining products (product networks) according to their rules, such as developing a wireless network for the electrical devices and designing the service specifications of these devices etc. Based on these, firstly we should develop hardware devices and obtain computational artifacts with physical characteristics. The second step is to develop the software platform (middleware, framework) on which to develop and install different applications that make up an ambient system [7]. In this paper, we only consider the step of software development for ambient systems. We consider the following three main goals for developing software platforms for ambient systems: to support high-level application programming, to provide a systematic approach and model for ambient applications, and to support easy and effective application development for ambient systems.

We developed Aframe, a software agent based framework for ambient systems. It is based on a multi-layers architecture built on existing operating systems, and implemented by using AmbientTalk [8]. Aframe supports easy and effective application development for ambient systems. It supports applications self-adaptive in a dynamically changing environment, dynamic application services and resources deployment, and services collaboration among agents. Moreover, application programmers can easily develop their application services by extending Aframe’s services.

The outline of the rest of the paper is as follows. Section II introduces the programming model and architecture of Aframe. Section III analyzes the development challenges and implementation of ambient system with Aframe. Section IV shows an application example of ambient system based on Aframe. The paper ends with conclusions and prospects in Section V.

2. Overview of Aframe framework

The system model of Aframe is shown in Figure 1. In Aframe, an application programmer programs resident agents to provide all the application services on devices of ambient system. Such application services provide local resources for the application to visiting mobile agent. To program resident agents, the programmer can invoke, configure and extend application services provided by Aframe and underlying hardware devices. A mobile agent can automatically migrate from node to node in the ambient system carrying its state and accumulated results. It can dynamically use different application services provided by resident agents on local devices for accomplishing the
mobile agent’s specific application tasks. Mobile agents are created by applications. An application as the owner of a mobile agent can send the mobile agent to travel in the ambient system and receive the mobile agent back to the application’s device.

Aframe is a multi-layer agent framework for Ambient Systems. It contains four layers: Framework Services Layer, Resident Agent Layer, Mobile Agent Layer and Application Layer. The architecture is shown in Figure 2.

A. Framework service layer

The framework service is at the bottom of the framework. It provides the interfaces to invoke the existing services which are provided by the underlying hardware device, it also provides the generic functions and services to the upper layer, so as to help ambient applications self-adapt to dynamically changing environment. The framework services are accessed only by the resident agent layer but not by other layers. The resident agent layer can invoke any services from this layer, and then provide them to mobile agents. The following are more detailed descriptions of the framework services.

-ID name: When there are multiple agents and multiple applications workings at the same time in an ambient system, naming collision may happen. In this situation, a unique ID name of each device is very important. Using this service, once the resident agent is initialized in a device, it will generate a universally unique identifier (UUID) as ID name to each device. Upper layers can read this information but not configure it.

-Data fetch and storage: In order to reduce network overhead and improve application efficiency in ambient systems, Aframe divides data into two types: shared and non-shared. The shared data only contains the local ID name and available services of local device; resident agents will share this information to the ambient system. On the other hand, non-shared data contains code of agents, specific application services, existing data in local devices and executing result of mobile agents. Meanwhile, when programmers develop mobile agents, they can decide whether to let mobile agents store the executing result in local devices, and what data mobile agents need to carry away to the next migrating devices.

-Ambient system status: This service makes use of the network information provided by AmbientTalk. In AmbientTalk, every node can listen to the whole local network to determine how many nodes are currently available in the ambient system. Also, resident agents can share all the IDs of currently connected nodes, as well as available services of local devices. By using this service, the framework service layer can provide the latest status of the ambient system to upper layer, such as the ID names list and available services list. Then agents can self-adapt to the dynamically changing environment. Meanwhile, users of ambient systems can access the current available services from user interfaces based on this service.

-Migration service: This service provides the basic migrating environment to the upper layer. Resident agents directly invoke this service and provide it to mobile agents. Using this service, mobile agents can migrate around the mobile devices in ambient systems.

-Time service: In some situations, such as in a bed and breakfast (B&B) described further in Section 4, there may be ambient services provided to guests. When a visitor decides to schedule some services, temporal information is very important. Therefore, using the time service, the ambient system can automatically release mobile agents to execute any ambient applications that the B&B may provide, in the scheduled time. Moreover, when a mobile agent is migrating to another device, it can get the current local time by invoking this service through the resident agent. On the other hand, we can test the time efficiency of each ambient application by invoking this service.

-Deployment service: Due to the high dynamism of ambient systems, every mobile device can join and leave the ambient system anytime and anywhere. Using this service, Aframe can dynamically and automatically deploy the framework services and the resident agent to a new device when it joins the ambient system. At the same time, if a device does not have the necessary resources or services, mobile agents can also automatically transfer the necessary resources or services between devices by using this service through the local resident agents.

B. Resident agent layer

Resident agents provide all the local application services to visiting mobile agents. A resident agent can be automatically deployed to a new device when it is joining the Ambient System. Once a resident agent is deployed
onto a device, it will stay there to provide application services. Mobile agents can directly use all the services which the resident agents provide. Application services provided by the resident agent layer can be built on the services provided by the framework services layer. Framework services only provide basic and generic functions and services for applications. Application programmers can develop new application services and deploy them to this layer. Thus the services provided by this layer contain three parts: services directly provided by the underlying hardware devices, services provided by the framework services layer and specific application services developed by application programmers.

C. Mobile agent layer

A Mobile agent is mainly used to execute different application services provided by resident agents. As discussed above, a mobile agent does not contain different application services in its code; it just contains some basic data, such as its migration strategy, processing strategy of executing result, as well as computation and communication results. Meanwhile, a mobile agent is also used to transfer necessary resources or application services to some devices when they do not contain such resources or services.

D. Application layer

Applications are owners of mobile agents. An application provides the user interface to its user who uses a mobile device. Using the user interface, a user can select the currently available services which are provided by the ambient system. Then the Aframe will automatically release a mobile agent to execute the relevant application task, or release multiple mobile agents to accomplish different tasks simultaneously. Aframe supports multiple agents with multiple application owners working at the same time.

3. Ambient system development with Aframe

Aframe is designed to meet the unique challenges of ambient systems, such as supporting applications self-adaption in dynamically changing environment and collaboration of devices; so as to facilitate the ambient environment. Meanwhile, Aframe works as a software platform of ambient systems on which to deploy diverse applications. Furthermore, Aframe objects to provide a systematic development approach and programming model to programmers, so as to enable them to effectively develop applications for ambient systems. In this section we will first present and analyse the design challenges of ambient systems, and then describe how to implement Aframe so as to address these challenges.

3.1. Dynamic changing environment

Ambient Intelligence environments are characterized by their high dynamism. The set of devices present in the environment is constantly changing, and consequently, the set of services offered by these devices is varying as well [9]. Therefore, it is a challenge to let ambient applications self-adapt to a dynamically changing environment. In Aframe, we mainly consider two situations when mobile agents are executing the ambient applications while the ambient environment changed:

- Devices get disconnected when mobile agents are migrating around ambient systems.
- New devices join the ambient systems when mobile agents are executing applications.

In Aframe, the framework service layer can provide the latest status of the ambient system to the resident layer. At the same time, Aframe provides three migration strategies for application programmers to develop mobile agents: migrating among all the devices, migrating among some of the devices, or only migrating to one device. Thus, based on the latest ID name list and latest available services list from Aframe, programmers can flexibly select a migration strategy, define a migration list and get a primary identification of service requirements when they are developing mobile agents to accomplish ambient applications. Moreover, programmers can design an algorithm for the migration sequences of mobile agents according to their specific situations.

Therefore, for the first situation, there are two main issues: (a) devices which mobile agents are migrating to get disconnected and (b) devices which mobile agents are currently visiting get disconnected. In Aframe, a mobile agent
can obtain the latest ID names list of the currently connected devices from the resident agent when it migrates to a new device. We also use the strategy that a mobile agent can store the list of the devices which it had visited before. Thus, the mobile agent can compare the mobile agent’s migration list, latest ID names list and history list every time when it is migrating to a new device, to decide the next migrating target automatically. Meanwhile, because the migration time of a mobile agent from one device to another is very short, the probability that the next visiting target device will get disconnected during a mobile agent’s migration is low. Moreover, if a device which a mobile agent is currently visiting gets disconnected, we use the strategy that mobile agent’s owner application will wait for some preassigned time, which is decided by the programmer. When the time runs out, the owner application can automatically release a new mobile agent. Consequently, in Aframe, mobile agents can self-adapt when devices get disconnected as they are executing applications in an ambient system.

On the other hand, connectivity situations will change and new services requirements will arise when new devices join the ambient system. As discussed above, mobile agents can get the latest connectivity status when they migrate to a new device and automatically decide the destinations. Therefore it will not impact mobile agents’ migration when new devices join the ambient systems and connectivity situations change. The main issue for the second situation is varying services requirement. We first assume that all the devices have the initial executing environment of Aframe when they join the ambient system. Since Aframe provides the deployment service, we use the strategy that when a new device joins the ambient system it will automatically send a request to other devices by using AmbientTalk [8]. Once a former device of the ambient system gets the message, it can automatically deploy the framework service and the resident agent to the new device. After that, as previously mentioned, based on the framework service, the new device can get the latest status of ambient system, such as the latest ID name list and latest available services list. Therefore, based on these services, when mobile agents are migrating to a new device which just joined the ambient system, and that device does not contain the necessary services or resource for mobile agents’ task, mobile agents can automatically transfer the services and resources to it from other devices through the local resident agent. Meanwhile, a user of new device can release a mobile agent to collect the necessary services and resources.

3.2. Collaboration of devices

Although we have the necessary infrastructure to achieve some specific goals usually, there is still a large gap between the raw functionalities offered by a surrounding system and the higher level user needs. This is mainly because the major devices in such systems are usually isolated. They just offer a specific and concrete function but lack a real capacity to collaborate among themselves [10]. Consequently, we use both resident agents and mobile agents simultaneously in Aframe so as to improve collaboration capability of devices in ambient systems.

There are two collaboration mechanisms in Aframe: collaboration among resident agents and collaboration between resident agents and mobile agents. By the collaboration among resident agents, every device in ambient system can get the latest status of ambient system, such as how many devices are currently available and which services the devices contain. Meanwhile, as mentioned above, based on the latest status of the ambient system, mobile agents can self-adapt to dynamic changing environment when they are executing ambient application. Also, through the collaboration between resident agents and mobile agents, mobile agents can become more flexible since resident agents provide sufficient application services from local devices to support mobile agents’ application tasks. Mobile agents can transfer new services and resources to a device through resident agents, to help the device extend application services, so as to provide sufficient services to end-user. Moreover, in Aframe, mobile agents can also share their messages through resident agents when they are cooperating to execute a task, so as to finish the task faster.

4. Application example

Nowadays, bed and breakfast (B&B) establishments are very popular, and in order to keep competitive with the rest of the lodging industry, their quality and service must be regularly improved. Diverse customers may expect it can offer business conference services, spa services, nightly wine and cheese hour’s service and many other services. Thus, for a B&B, it is very important to provide sufficient and appropriate services to customers, execute their selected services according to a schedule time, and update the status of available services to customers as soon as
possible. Meanwhile, there have been many successful household applications by using ambient systems [11]. Therefore, we designed a demonstration application of ambient system with Aframe about services provision in B&B.

The ambient system we designed for B&B is shown in Figure 3. We deployed the devices to all the places which provide services to customers, such as the kitchen, fridge, conference room etc. After a customer finishes the registration at the front desk of the B&B, the service computer will automatically deploy the agents and generic services to the customer’s cellphone, such as the current available services list. Also, the service computer will update the services list to customers and other service devices periodically. Then from the user interface of this cellphone, the customer can select the services which he requires. For example, if he wants to book a conference room at a scheduled time, he just needs to select the business conference service in the user interface, and then his cellphone will automatically release a mobile agent to the device of conference room. Once the device in the conference room receives this service request from the mobile agent, it will automatically verify the identity (or name) of the customer with the service computer in front desk. If it passes the identification, the device in conference room will check whether the schedule time is available or not. Finally, the mobile agent can get the result and automatically bring it back to the customer’s cellphone. Meanwhile, if the customer wants to make an appointment for the breakfast, the services devices for menu, kitchen and dining room will collaborate together; the customer can first get the information about the breakfast, such as the current menu, available time or some extra services from the kitchen, and then define the breakfast and schedule the time for it based on these information. Moreover, as Figure 3 shows, customers in the B&B can get many other intelligent services by using the ambient system with Aframe.

The working procedure of the ambient system in the B&B is shown in Figure 4. For this application, we firstly developed resident agents so as to provide the necessary application services at local devices to visiting mobile agents. Beyond involving the existing services from the underlying hardware devices as mentioned above, in order to help the visiting mobile agents dynamically and self-adaptively migrate around the ambient system, the resident agents provide the migration service and status of ambient system service. These two services can directly be invoked from Aframe’s framework services layer. Also, in the B&B application scenario, the names (or identity) of customers and timestamps of the schedule services are very important. Thus, the resident agents also invoke two other framework services: ID names service and time service. By using these two services, the resident agents in services devices can get the identities (or names) of the customers who used their mobile devices to request these services when mobile agents are visiting there, as well as the schedule time for these services. Secondly, we developed the mobile agents and its owner applications for this application. The mobile agents’ tasks are to automatically execute the application services in the service devices of ambient system, and bring the executing results back to their owners. The applications create and send the mobile agents to execute the tasks in the ambient
system and display the results of the tasks in the user interfaces of customers’ mobile devices.

![Figure 4. The working procedure of the ambient system with Aframe](image)

5. Comparison and evaluation

<table>
<thead>
<tr>
<th>Project</th>
<th>Programming model</th>
<th>Systematic approach</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aframe</td>
<td>Multi-layer</td>
<td>Support</td>
<td>High</td>
</tr>
<tr>
<td>AmbientTalk</td>
<td>Actor</td>
<td>Partial support</td>
<td>Low</td>
</tr>
<tr>
<td>3APL</td>
<td>Multi-agent</td>
<td>Partial support</td>
<td>Middle</td>
</tr>
</tbody>
</table>

3APL [12] is a programming language for implementing agent-based ambient systems. It provides programming constructs to deal with agents’ tasks and a set of practical reasoning rules to update or revise the agents’ goals. The execution of each 3APL program can be monitored by exchanging messages among the agents. Also, 3APL is integrated with Java programming languages. Compared to 3APL, the programming model of Aframe is application specific services oriented, thus Aframe supports diverse applications. In Aframe, programmers can develop different application services according to the specific applications. Meanwhile, all the application services in Aframe are provided by the resident agents, every application in Aframe has the same structure. Therefore, Aframe fully supports systematic approach for application development of ambient systems.

The basic idea of the AmbientTalk programming paradigm is that it can incorporate network failures in its programming model [8]. Meanwhile, AmbientTalk employs a purely event-driven concurrency framework based on actors. In AmbientTalk, actor executions can be concurrent, with asynchronous actor method invocations. Also, AmbientTalk combines the JVM as a platform, so it is easy for AmbientTalk programs to use Java libraries. However, AmbientTalk is a completely new language, programmers have to spend a long time to become familiar with it before using it to develop applications for ambient systems. Furthermore, AmbientTalk does not provide implemented application services, so the developing efficiency by AmbientTalk is low. Aframe makes use of the mechanism of AmbientTalk symbiotic programming with Java, and integrates generic services in its framework, thus Aframe supports programmers easily and effectively develop applications for ambient systems by using Java.
6. Concluding remarks

In this paper we present Aframe, a multi-agent framework for ambient systems. By integrating AmbientTalk and software agent technique, as well as multi-layers framework, Aframe can simplify and hide the complexity of handling different connectivity status, varying services requirements and diverse devices. It provides a high level software platform to developers of ambient systems. Aframe improves efficiency and effectiveness of application development, as well as easy programming, for developing ambient systems, also with extensibility support. Moreover, Aframe supports collaboration among multiple agents working in multiple devices simultaneously; thus it can provide sufficient services and functions to end-users when they are using the ambient system, and the ambient applications can be more autonomous and self-adaptive in dynamically changing environments.

Further challenges of developing Aframe are in two aspects: universality and security. Since the framework service layer of Aframe is implemented by AmbientTalk, and due to some technique constrains of AmbientTalk, the current version of Aframe can just run on top of AmbientTalk, and AmbientTalk mainly runs on Android platform and on J2ME under a CDC/Personal profile configuration, as well as conventional operating systems etc. However, since the devices in an ambient system are usually heterogeneous, it requires the framework (or middleware) of such system to have excellent universality and not only to supports some specific platforms. Consequently, we plan to encapsulate the Aframe to a Java virtual machine, so as to make it also work in other platforms (or operating systems) as long as they can support Java, such as TinyOS. This would improve Aframe’s universality. For the security issues, we plan to use two strategies to approach it. One is the identification of the ID names when a new device wants to access the ambient systems. The other strategy is the dynamic services security identification. With some techniques, such as semantic and data mining, Aframe can only authorize specific services to each device according to its identity. When a device releases a mobile agent, it can use only the authorized application services. Aframe can also analyse the activity of each mobile agent, then based on the mobile agent’s activity history and its owner’s identity, to decide whether the mobile agent is secure or not.

References