A Generic Mobile Agent Framework for Ambient Intelligence

Yung-Chuan Lee, Elham S.Khorasani, Shahram Rahimi, Bidyut Gupta
Southern Illinois University
Department of Computer Science
Carbondale, IL 62901, USA
{ylee, elhams, rahimi, bidyut}@cs.siu.edu

ABSTRACT

The purpose of this paper is to introduce an innovative framework for implementation of ambient intelligence (AmI) environments. Compared to the existing state-of-the-art approaches, this framework creates a more decentralized and distributed AmI environment. In addition, the proposed approach is not limited to one specific domain, unlike many others. The openness of the presented architecture allows it to support a variety of devices ranged from small-embedded sensors to complex computing facilities. Finally, given that this approach is formulated based on multi-agent standard concepts, it can be easily implemented as add-on for existing software agent platforms to achieve rapid deployment. Implications for the development of this framework and future directions are discussed.

Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Intelligent agents and Multiagent systems

General Terms

Management, Design.

1. INTRODUCTION

Ambient Intelligence (AmI) implies a ubiquitous environment of computing, networking, and interfacing that is aware of and reactive to the presence of people. It provides personalized knowledge and services to each individual by intelligently interacting with the environment and the individuals [5] [6]. With the advance of mobile technology, the technology disappears into our surroundings and only the user interface remains perceivable. In an ambient intelligence environment, people are supported in carrying out their activities in an easy and natural way, using intelligence that is embedded in the environment. By utilizing sensors and other small devices, AmI enabled systems can retrieve profile of a user and provide relevant information and services to her, and even further, intelligently learn from the interactions with the user to refine her profile.

AmI environment presents a mixed combination of autonomy, learning, parallel and distributed computing domains [12]. Both traditional client-server approach and mobile agent technology are capable of being applied for implementation of AmI environment. However, multi-agent systems (MAS) in nature is superior fit and shares many of its characteristics with AmI environments [2]. MAS facilitates design and development of AmI environment by providing features such as autonomous reasoning, learning, mobility, and collaboration among others. Agents can migrate into different hosts to perform computation, while communicate with other agents, and carry the results back to its origin. Many believe that MAS is the Holy Grail to completely reveal the promising future of AmI environment [4][11][12][13][15].

While a few MAS-base AmI architectures have been proposed, none has appropriately evaluated the integrity and correctness of its model. Because of the dynamicity posed by AmI environment and the complexity of the criteria of migrations and communications among agents and hosts, it is necessary for the models to be verifiable. Software verification employs formal methods such as π-Calculus [9][10] that are mathematical provable formulations to perform program analysis and model checking. It provides a mechanism to warrant the correctness of a program and enhance the reliability at each stage of software life cycle [14][7]. With the help of formal methods, a dynamic and complex system can be mathematically formulated which can then be analyzed to verify its correctness. Thus, it is critical to incorporate a verification procedure as part of the AmI framework to offer the above benefits to both system developers and regular users.

This work presents a framework that utilizes mobile agents for ambient intelligence in a distributed ubiquitous environment to provide users with personalized knowledge and intelligent interactions as well as to sustain expeditious performance under dynamic resource demands. In this paper, general formulation using π-Calculus is included to model the proposed system and to construct the framework for future system verification implementation. Some important features such as integrity verification on a given system, communication verification among agents and performance verification of giving scenario will be included in the initial implementation; however, detailed formulations of those features are out of the scope of this paper and will be elaborated in forthcoming works.

1.1 The State of the Art Approaches

There are several studies related to utilization of mobile agents in ambient intelligent environment. Satoh employed RFID and software agents, creating location-aware service to provide personalized information to users [13]. Each user is assumed to carry a RFID tag and can be uniquely identified by RFID sensors in each equipped location server. According to the user profile, the location server then assigns the user to a host within its coverage area. Then a mobile agent is spawned to assist the user and moves from one host to another to "follow" the user.

As another example, a similar approach was developed by AMILAB research group to apply software agents in attempting to create an AmI manufacturing environment [11]. In this approach, AMICO architecture is formulated to interact intelligently with users to provide user-specified context information and machine functionalities. Furthermore, Yong and et al. proposed a context-aware AmI application system based on a multi-agent architecture [15].

Several approaches are centered at facilitating people in daily tasks. Kidd and et al. presented "the aware home project" to study how ubiquitous computing could assist people in daily life [8]. Because of the different characteristics of office and home environments, where activities in office environment are more goal oriented while in home environment are more flexible, the project concentrated on developing a methodology that discovers useful applications from the latest advances for each application. Hagras and et al. implemented an ambient-intelligence environment, iDorm, using embedded sensors, actuators, and software agents [4]. Users can interact with the agents in the embedded controller, robots or mobile devices to control the environment. The system evolves from those interactions to provide more precise user-friendly living environment.

The current state of the art approaches are domain-specific and cannot be flexibly applied to other domains. Moreover, one of the mentioned approaches have addressed the performance of the system or load balancing among agencies in an AmI environment. Because of the autonomy of the agents and the dynamicity of the environment, any approach should take the performance criteria into consideration to provide users with responsive services. Furthermore, the complexity of such architectures highlights the importance of rigorous verification of the integrity of the system. Hence, we propose a verifiable generic agent-based AmI system with concentrations on optimized communication costs and load balancing among agents and agencies.

1.2 π-Calculus Overview

Since the proposed system is modeled in π-Calculus, this section provides a brief overview to the main entities of π-Calculus. The interested reader is referred to [10][11] for more information on pi-calculus syntax and reduction rules.

In π-Calculus, two main entities are specified, "names" and "processes" (or "agents"). "Name" is defined as a channels or a value that can be transferred by a channel. We follow the naming rule and syntax in [10] in which u, v, w, x, y, z range over names and A, B, C, ... range over process (agent) identifiers.

The syntax of process is defined as follows:

$$P ::= 0 \mid \Sigma_{i\in I}\lambda_i.P_i \mid \overline{y}x.P \mid \overline{y}(x).P \mid \overline{x}(K) \mid x(U) \mid y(x).P \mid$$
$$\tau.P \mid P_1 \mid P_2 \mid P_1+P_2 \mid (x)P \mid [x=y]P \mid A(y_1,...,y_n) \mid !P$$

- 0 means agent P does not do anything.
- $\Sigma_{i\in I}\lambda_i.P_i$ is called summation. P will behave as either one of $\lambda_i.P_i$ where $i \in I$, but not more than one, and then behaves like P_i. If $I = \Phi$, P actually behaves like 0. Here λ_i denotes any actions that could take place in P (such as τ, $\overline{y}(x)$, and so on).
- $\overline{y}x.P$: Agent P sends free name x out along channel y and then behaves like P.

Name x is said to be free if it is not bound to agent P. In the same way, if x is bound to P (also say private to P), that means x can only used inside of P. The following example could best explain the relationship between the free name and the bound name. Suppose a system consists of agent P and Q. The name x in P is free while name y in P is not free. Q contains free name x and bound name y. Here the free name x in P and the free name x in Q are the same one, while the bound name y in P and the bound name y in Q are different although they have the same name.

- $\overline{y}(x).P$: Agent P sends bound name x out along channel y and then behaves like P.
- $y(x).P$: Agent P receives name x along channel y and then behaves like P.
- $\tau.P$: (τ is a silent prefix.) $\tau.P$ performs the silent action τ and then behaves like P.
- $P_1 \mid P_2$ is a composition. The agent P has P_1 and P_2 executing in parallel. P_1 and P_2 may behave independently or they may interact with each other. E.g. if $P_1 = \tau_1.P_1$ and $P_2 = \tau_2.P_2$, then P_1 and P_2 will behave independently. Otherwise, if $P_1 = \overline{y}(x).P_1$ and $P_2 = y(x).P_2$, then P_1 will send x to P_2 through channel y.
- The sum $P_1 + P_2$ means either P_1 or P_2 will be exercised, but can not be both.
- $(x)P$ is a restriction action. Agent P does not change except for that x in P becomes private to P. That means any outside communication through channel x will be prohibited.
- A match $[x=y]P$

  If $x = y$, this agent behaves like P, otherwise like 0.
- $A(y_1,..., y_n)$ is an agent identifier where $y_1,..., y_n$ are free names occurring in P.
- $!P$ is called replication and can be thought of as an infinite composition $P \mid P \mid P \mid ...$, i.e. $P = P \mid !P$.

In addition to the basic π-Calculus, higher-order π-Calculus has the ability to send and receive process (agent). So in higher-order π-Calculus, $\overline{x}(K)$ means send a name or process K through channel x and $x(U)$ means receive name or process through channel x.

Based on the above syntax, different components of the proposed framework are defined.

2. PROPOSED FRAMEWORK

Our framework for ambient environment allows users to obtain their personalized information such as their profile, preferences, likings and habits, while having the minimum interaction with the environment. As the user moves, this information will be available to her at the new location. The framework presented here aims to be flexible, high performance and easy to implement.

Figure 1 illustrates the overview of the framework. The system is formed from multiple geographically distributed environments. Each environment may provide different services to users such as banking, shopping, etc. Each environment consists of three main components: (1) mobile devices (2) hosts or agencies and (3) directory service center (DSC). Different environments are connected through their directory service centers. In the following sections, we describe each of the elements in figure 1 in more details. As mentioned before we model our ambient framework using higher order π-Calculus in order to provide mathematical infrastructure for evaluation and verification of the system. More specifically, an environment is a cluster consists of one DSC, multiple mobile devices and multiple hosts and can be formulated as following.

$$ENVIRONMENT = DSC \mid MD_1 \mid ... \mid MD_n \mid HOST_1 \mid ... \mid HOST_k$$

Where *DSC* is the Directory Service Center, $MD_1 ... MD_n$ represents mobile devices and $HOST_1 ... HOST_k$ indicates different host/agencies in the environment.

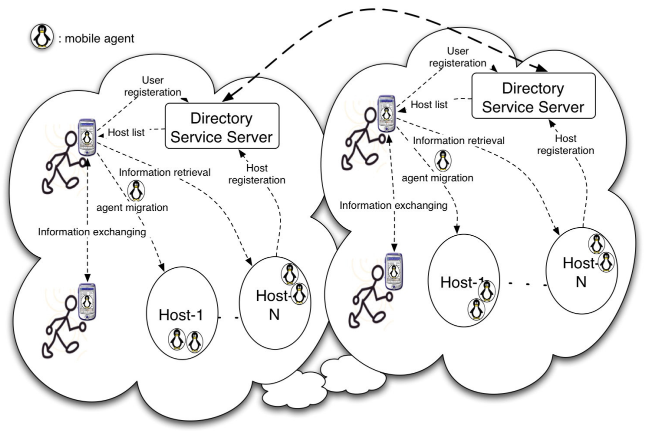

Figure 1. Overview of proposed framework

2.1 Mobile Device

For simplicity we assumed that each user carries at least one computing device such as tablet PCs, PDAs, cell phones, notebooks or even wearable. User carries these portable devices to store her personalized information as well as to communicate with the environment over the wireless media. We refer to these portable devices as mobile devices. These devices are dedicated to a single person; therefore, the user is responsible for the security of her personalized information which is stored in her mobile device. Due to the limited capabilities of the portable devices in terms of CPU power, amount of memory, and input/output facilities, mobile agent technology is employed. The mobile agent can merge into a suitable host (also called agency) in an environment to perform various services and computation for its owner. Additionally, it can use the environmental information which is provided by the agency.

Figure 2 depicts the internal structure of a mobile device. A mobile device (MDi) consists of registration module (MD_REGi), administration module (ADMINi), load-balancing module (LBi), learning module (LEARNi) and agent database (MD_DBi) and and be formulated as following:

$$MD_i = MD\_REG_i \mid ADMIN_i \mid LEARN_i \mid LB_i \mid MD\_DB_i$$

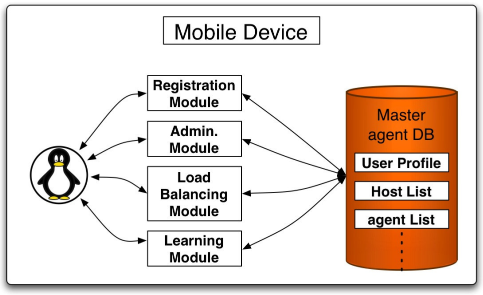

Figure 2. Mobile device structure

Registration module (MD_REGi) is responsible for registering the mobile device with new environments as user goes from one environment to another one. Whenever a user enters to a new environment, her mobile device submits its ID, the physical network adapter address, to the DSC of the environment. DSC then decides whether to accept or reject the mobile device registration request due to the security considerations.

$$MD\_REG_i = \upsilon(x, id)(\overline{md}(x).\overline{x}(id)).0$$

Where *md* is a communication channel between MD_REG_i and *DSC*.

Mobile device acts as a master agent. It can create numerous mobile agents and dispatches them to different hosts. The administrative module ($ADMIN_i$) is responsible for creating and keeping track of the mobile agents that belong to the mobile device and reside in different agencies. This module consists of the following processes:

$$ADMIN_i = MERGE_i \mid TERMINATE_i \mid INQUIRY_i \mid RETREIVE_i \mid UPDATE_i$$

- *MERGE*: This process creates a mobile agent and sends its requirements to DSC through channel *a*. These requirements specify agency capabilities that are needed by mobile agent to perform its services. If the request was approved, DSC sends back a list of candidate hosts $(h_1, h_2, ..., h_m)$ that fulfill the requirements. This list contains the agency's network address and other necessary information. *MERGE* process then sends the list of candidate hosts to the load balancing module (LB_i) through channel *ad* and receives the address of the most appropriate host (h_{opt}) to which the mobile agent immigrates.

$$MERGE_i = \upsilon(AGENT, \overrightarrow{req}, c)(\tau_{create}.\overline{a}(c).\overline{c}(\overrightarrow{req}).$$
$$c(h_1, ..., h_m).\overline{ad}(c).\overline{c}(h_1, ..., h_m).c(h_{opt}).$$
$$\overline{h}_{opt}(AGENT)).MERGE_i$$

where τ_{creat} is the internal action for creating *AGENT*; $\overrightarrow{req}$ is the host capabilities needed by *AGENT*, which is sent to DSC along the private channel c. $h_1,\dots,h_m$ is the list of the hosts received from DSC and sent to LB_i module. *h* is the preferred host sent back by LB_i for migrating agent.

- *RETREIVE*: This process retrieves the mobile agent from the environment to the mobile device. This includes retrieving the current address of the mobile agent from the agent database and sending a *ret* signal to the agent and receiving the agent along a private channel (*c*).

$$RETREIVE_i = \nu(c)(\overline{h}_i(c).\overline{c}ret.c(AGENT).RETRIEVE_i$$

  Where h_i is the current address of the Agent.

- *INQUIRY*: This process enquiries the mobile agent about the completeness of its current task, by sending *inq* signal.
- *TERMINATE*: This process sends termination signal, *term*, to the mobile agent to cease it. After sending the termination signal, the record of this mobile agent should be deleted from the mobile device database.

$$TERMINATE_i = \overline{ag}_i^j term.TERMINATE_i$$

$$INQUIRY_i = \overline{ag}_i^j inq.ag_i(\overrightarrow{res}).INQUIRY_i$$

$$AGENT_i^j = ag_i^j(m).([m = term]\tau_{term}.0 \mid$$
$$[m = inq]\overline{ag}_i^j(\overrightarrow{res}).AGENT_i^j)$$

  Where ag_i^j is the channel of communicate between the mobile device (MD_i) and $AGENT_i^j$, τ_{term} is the internal action for terminating the agent and $\overrightarrow{res}$ is the result of inquiry.

- *UPDATE*: This process updates agents' list in the database of the mobile device. For each mobile agent that belongs to the mobile device, there exists a record in the database that contains the mobile agent's ID and its current address. If the mobile agent migrates from one host to another, it will send its new address to the mobile device and *UPDATE* process updates the record of this mobile agent in the database.

The load balancing module (LB_i) is responsible for spreading the communication and computation loads among the hosts in the environment to get close-optimal utilization and minimum computation and communication delay. Whenever the admin module creates a mobile agent, it forwards the list of the candidate hosts ($h_1,h_2,\dots,h_m$) for that mobile agent to the load-balancing module through the channel (*ad*). The load balancing module then sends a message (*m*) to each host ($HOST_i$) through channel (h_i) in the list to obtain the hosts' resource utilization and to determine the response time ($\overrightarrow{resp_i}$) of each one of them. It estimates the cost of migrating the agent to each of these hosts and provides the most underutilized one to the administrator module.

$$MD\_LB_i = (ad(c).c(h_1,h_2,\dots,h_m).(\overline{h}_1 m.h_1(\overrightarrow{resp_1}) \mid$$
$$\overline{h}_2 m.h_2(\overrightarrow{resp_2}) \mid \dots \mid \overline{h}_m m.h_m(\overrightarrow{resp_m})).\tau_{opt}.\overline{c}(h_{opt})).MD\_LB_i$$

Where h_{opt} is the selected host for the agent's migration and τ_{opt} is the internal action for computing the underutilized host based on the response time and process load.

The user profile as well as hosts and agents' information are stored in the database of the mobile device. Users are responsible for updating their sensitive information, such as username/password, credit card information and etc. explicitly and are responsible for the security of this information. The other information of the user profile, such as preferences and habits, are automatically updated by learning module, based on user's interactions with the environment. The learning module concentrates on the behavior of the user and employs computing with words and gesture recognition methodologies to achieve its objectives. Because the complexity of learning cannot be fully comprehended within few paragraphs and the main purpose of this paper is to sketch the blueprint of the framework, details on learning module will be included in an upcoming paper.

2.2 Host/Agency

The agencies provide the facilities for the mobile agents to be executed and to perform various services for their owners. Java Application Development Framework (JADE) [1] is the platform of the choice to provide the runtime environment for mobile agents and the agencies to execute. Figure 3 illustrates the host's structure. It consists of user-history database ($HOST\_DB_i$), registration module ($HOST\_REG_i$), information module ($INFORMATION_i$) and JADE system ($AGENT\_EXEC_i$) and is modeled as follows:

$$HOST_i = HOST\_REG_i \mid AGENT\_EXEC_i \mid INFORMATION_i \mid HOST\_DB_i$$

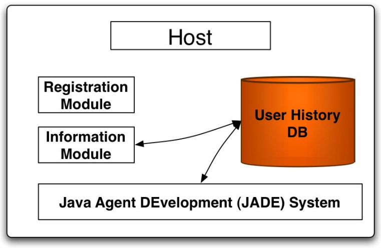

Figure 3. Host/agency structure

The registration module ($HOST\_REG_i$) registers the host with the directory service center. It sends a registry request to DSC that contains the agency's network address and its device profile.

$$HOST\_REG_i = \nu(b)\overline{h}(b)\overline{b}(\overrightarrow{prf})$$

Where h_i is the channel between $HOST_i$ and DSC and $\overrightarrow{prf}$ is the device profile.

$AGENT\_EXEC_i$ process receives and executes agents (τ_{exec}). It can also return back the agent to its mobile device upon receiving the *ret* signal.

$$AGENT\_EXEC_i = !(h_i(AGENT).\tau_{exec} \mid h_i(c).c(m).[m = ret]\overline{c}AGENT)$$

When a mobile device communicates with an agency, either by dispatching an agent to the agency or by remote method invocation (RMI), the agency records the user history in the user history DB. The information update module stores and updates the user's history according to the user activities. This history could include the user interaction with the agency, or in the case of agent migration, it could be the result of the execution of the mobile agent. After migration, the agency can request the mobile device for the summary of the agent execution results and update the user's history based on this information. The user can decide whether to provide this information to the agencies or not, by configuring its mobile device.

By employing the above approach, the environment could learn from the user activities, and in consequence, the next time the user comes to the environment, it can access its past information stored in the agencies. The user should first authenticate itself to the host to be able to access its history.

Due to the excessive amount of details and lack of space, we skip modeling user history database and information modules in π-calculus.

2.3 Directory Service Center

The Directory Service Center (*DSC*) is responsible for managing mobile devices, host/agencies and their intercommunication as well as communicating with other DSCs in other environments. Figure 4 depicts the DSC structure. It consists of registration module (*DSC_REG*), communication module (*COMM*) and registration Database (*DSC_DB*) and is formulated as follows.

$$DSC = COMM \mid DSC\_REG \mid DSC\_DB$$

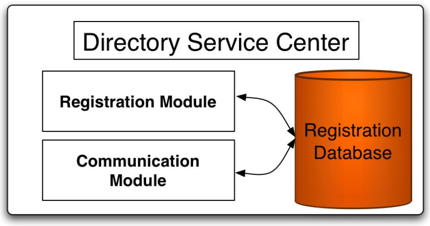

Figure 4. Directory Service Center Structure

Registration module *(DSC_REG)* is responsible for storing and updating registry information of mobile devices, agents and host/agencies in the registration database. For host and mobile device registration, DSC receives the device profile and mobile device id respectively and stores them in its database. For agent registration, DSC receives the agent device requirements and sends back the list of hosts that satisfy the requirements. Other than host, agent and mobile device registration, the registration module also periodically verifies the network connectivity to the hosts and the mobile devices to validate existing records in the registration database. The π-calculus formulation is:

$$DSC\_REG = HOST\_REG \mid AGENT\_REG \mid MD\_REG$$

$$HOST\_REG = !(h(b).b(\overrightarrow{prf}))$$

$$AGENT\_REG = !(a(b).b(\overrightarrow{req}).\bar{b}(h_1,..,h_m)$$

$$MD\_REG = !(md(b).b(id))$$

The communication module manages the communication among the mobile agents as well as the communication between the mobile device and the hosts. If the mobile device asks for a service that is not provided by a host in the same environment then DSC broadcasts a message to DSCs in other environments to obtain the network address of a host that provides such a service. It then forwards the network address of that agency to the mobile device so that the mobile device can communicate directly with the agency.

The mobile agents' communication is more complicated since their network addresses change upon each migration. Our proposed approach employs hierarchical methodology for mobile agent communication in our framework. The proposed framework benefits from a two-level hierarchy for tracking agent locations. Figure 5 demonstrates this hierarchy. Each mobile agent belongs to a mobile device. Upon migration, the agent notifies its mobile device regarding its new location. The directory service center stores the network address of all mobile devices that are registered. Each mobile agent is associated with a unique ID. This ID consists of two parts: (1) its master-agent-ID that shows to which mobile device this agent belongs, and (2) the agent-ID. As described earlier the mobile device acts as a master agent and keeps track of the location of its agents. And the directory service center records the network addresses of all mobile devices in the environment. Figure 5 depicts our communication scheme. The numbers shows the sequence of communications. When agent1 needs to communicate with agent2, it sends agent2's ID to DSC and asks for the location of agnet2. The directory server extracts the master-agent-ID part and sends a request to the mobile device associated with the master-agent-ID and asks for the location of agent2. As mentioned, the mobile device has a record of the actual location of the host in which agent2 resides. This location is then sent back to the DSC by the master agent. Consequently DSC forwards this address to agent1. Now agent1 can communicate directly with agent2.

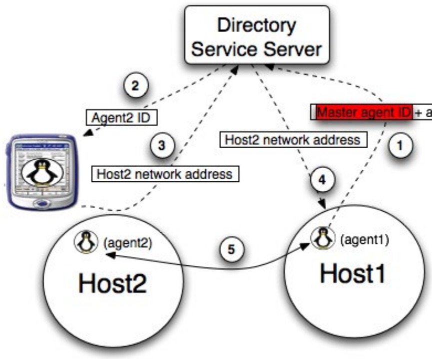

Figure 5. Communication Scheme

This communication scheme can be represented as following.

$$COMM = DSC\_COMM \mid MD\_COMM_i \mid TAGENT\_COMM \mid SAGENT\_COMM$$

$$DSC\_COMM = \upsilon(ma_1, ma_2,..,ma_m)!(c(ma,sub).([ma = ma_1]\bar{d}_1 sub.d_1(addr)\mid$$

$$[ma = ma_2]\bar{d}_2 sub.d_2(addr)\mid ... \mid [ma = ma_m]\bar{d}_m sub.d_m(addr)).\bar{c}addr).DSC\_COMM$$

$$MD\_COMM_i = !(d_i(sub.).\bar{d}_i(addr))$$

$$SAGENT\_COMM = \upsilon(m)\bar{c}(ma,sub).c(addr).!\overline{addr}(m)$$

$$TAGENT\_COMM = !addr(m)$$

Where *DSC_COMM, MD_COMM$_i$, SAGENT_COMM and TAGENT_COMM* are the communication processes of DSC, master agent, source mobile agent and target mobile agent respectively. *ma* is the master-agent-ID and *sub* represents the mobile agent-ID, and *addr* is the address of target agent.

3. CONCLUSION AND FUTURE WORK

By utilizing the proposed framework, we believe that a more intelligent and responsive interface between users and computing infrastructure can be implemented in a seamless environment. The framework employs mobile agents to eliminate hard-coded and fixed features of computing. This realizes intelligence capability where an intelligent infrastructure is more dependable, manageable, adaptable and affordable [3]. Mobile agents are deployed by demand and "live" in computing infrastructures to interact with users intelligently and intuitively, to provide personalized information, and to assist users on both daily and specific tasks. The proposed framework is designed to have performance advantages over the current state of the art when the number of the users increases.

In the near future, a simulated system with mentioned characteristics will be implemented for the framework to demonstrate the practicality and advantages of the proposed approach. π-Calculus will be employed to evaluate the system integrity, validation, and performance.

The authentication and other security issues need to be studied further to concrete the reliability and stability of our approach. Currently, we assume that authentication method uses public key cryptography and the public key of each user and hosts are repopulated in the directory service centers when a regular user or host administrator physically registers to join the system. Finally, to find the optimal network throughput in our framework, communication protocols among agents and migrations of agents among agencies will be further examined.

4. REFERENCES

[1] Bellifemine, F., Caire, G., Trucco, T., Rimassa, G., "Jade Programmer's Guide", JADE 2.5 http://sharon.cselt.it/projects/jade/, 2002.

[2] Braun, P., Rossak, W. R. (2004). Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit. San Fransisco: Morgan Kaufmann.

[3] Ferguson, R., Charrington, S., "Building an Intelligent IT Infrastructure", Intelligent Enterprise, Vol. 7, No. 18, P. 18, Dec. 2004.

[4] Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., Duman, H., "Creating an Ambient-Intelligence Environment Using Embedded Agents", IEEE Intelligent Systems Vol. 19, pp. 12-20, Nov. 2004.

[5] ISTAG, "*Scenarios for Ambient Intelligence in 2010*", http://www.cordis.lu/istag.htm.

[6] ISTAG, "*Ambient Intelligence: from vision to reality*", http://www.cordis.lu/istag.htm.

[7] Jackson, M., "What Can We Expect from Program Verification?", Computer, Vol. 39, No. 10, pp. 65-71, Oct. 2006.

[8] Kidd, C., Abowd, G., Atkeson, C., Essa, I., MacIntyre, B., Mynatt, E., Starner, T., "The Aware Home: A Living Laboratory for Ubiquitous Computing Research". In the Proceedings of the Second International Workshop on Cooperative Buildings, pp. 191-198, Oct. 1999.

[9] Milner, R., Parrow, J., Walker, D., "A Calculus of Mobile Processes - Part I" – LFCS Report 89-85. University of Edinburgh June 1989.

[10] Milner, R., Parrow, J., Walker, D., "A Calculus of Mobile Processes - Part II" – LFCS Report 89-86. University of Edinburgh June 1989.

[11] Perez, M. A., Susperregi, L., Maurtua, I., Ibarguren, A., Tekniker, F., Sierra, B., "Software Agents for Ambient Intelligence based Manufacturing", IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Application, DIS '06, 2006.

[12] Remagnino, P., Foresti, G. L., "Ambient Intelligence: A New Multidisciplinary Paradigm", IEEE Transactions on Systems, Man and Cybermetrics, Part A, Vol. 35, pp. 1-6, Jan. 2005.

[13] Satoh, I., *"Software Agents for Ambient Intelligence"*, IEEE International Conference on Systems man and Cybernetics, 2004.

[14] Woodcock, J., "First Steps in the Verified Software Grand Challenge", Computer, Vol. 39, No. 10, pp. 57-64, Oct. 2006.

[15] Zhang, Y., Hou, Y., Huang, Z., Li, H., Chen, R., "A context-aware AmI system based on MAS model", IEEE Proceedings of the 2006 International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP '06, pp. 703-706, Dec.2006.