VOL. 3, NO. 7, July 2012 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2009-2012 CIS Journal. All rights reserved.

http://www.cisjournal.org

Critical Analysis of Mobile Devices Based End-User Programming of Smart-Spaces

[1] Onukwugha C. G, [2] Osuagwu O. E
[1] Department of Computer Science, University of Port Harcourt, East-West Road, Choba, Rivers State, Nigeria.
[2] Department of Computer Science, Federal University of Technology, Owerri, Nigeria.
[1] onukwugha2000@yahoo.com

ABSTRACT

Smart-Spaces and their importance in the deployment of applications for various needs and the desire to provide specific end-user solution have been calling for an increased effort to provide end-users with the programming ability for handling their needs directly. This paper investigates a theoretical possibility of developing a task-driven model which can be deployed in providing end-user programmability of smart-spaces using mobile devices. It also analyses the present systems and models that are used to be able to evaluate their strength and pitfalls in realizing they said objective within a ubiquitous computing environment. The analysis takes into account the user, the presentation layer down to the realization layer of the system models.

Keywords: *Smart-spaces, Task-driven computing, Ubiquitous computing, Mobile devices, End-user programming.*

1. INTRODUCTION

Smart spaces are environments such as home, building, office, factory, etc. which are equipped with computers and computing devices that are capable of intercommunication. These computing devices are able to interact and exchange information within the space. They are also able to adapt to their environment in accordance to the result of the data they gather and process. The end-users monitor and control the environment, sometimes without knowing it, through the use of mobile devices especially cell phone.

Smart space programming presents us a very attractive offer of living in a digital world where computer becomes an integral part of everyday living. Top on the list of these attractions is the ease of use of the computer. The end-users do not need to learn conventional programming languages in order to accomplish a computational task. However, this trend has created some difficult challenges for information technology companies to place computers and sensors in virtually every device, appliance and piece of equipment in buildings, homes, workplaces and even clothing. Among the myriad of issues in smart spaces are the issues of privacy and interoperability.

Smart space programming supports context-aware computing. The old traditional desktop paradigm forces the user to search out the computer's interface, but, the new paradigm provides an interface that locates and serves the user. Information (physical, social, emotional) from the user is used as input to adapt the behavior of one or more computational components. Simply put, computers should know where they are.

Task-driven end-user programming of smart spaces using mobile devices seeks to accomplish tasks such as remote monitoring of electronic gadgets, reading e-mails, getting reminders, accessing information, etc. In order to perform a computing task such as these, the end-user must customize the generic programs developed by software vendors to suit his intent. To accomplish a task operation manually is difficult, but, a task-driven computing approach addresses the problem.

The technology that supports smart space programming is called ubiquitous or pervasive computing. Ubiquitous applications involve multiple resources and web services at the same time. Web services utilize these resources and provide the interfaces through which users can interact and control the smart environments. User-composed applications are realized from available resources and web services at run-time according to user need and other context information.

2. CHALLENGES OF PROGRAMMING OF SMART SPACES

The complexity emanating from the integration of a large number of technologies in smart spaces has given rise to many challenges some of which are examined below:

a. Lack of Standards

To foster integration, interoperability, and the development of emerging technologies, standardization and measurements are critical. The lack of common software infrastructure, tools to create, manage, measure, test, and debug pervasive services is also an issue. There is also lack of standards in key areas such as service discovery, MIS, Pico-cellular wireless networks, automatic configuration, and ad hoc transactional security.

b. Interoperability Problem

Interoperability problem borders on seamless and secure mobility for devices and services. The problem faced is how to interconnect a wide variety of heterogeneous and un-interoperable networks (wired and wireless) in order to provide users (end-users) with ubiquitous connectivity and the ability to roam seamlessly and securely across networks of different types. Key obstacles to seamless mobility include:

- Lack of Scalability: Roaming is available between a limited set of networks and operators;
- Lack of standard handover interfaces: No interoperability between vendor equipment.
- Limited quality of service (Quos) guarantee during handover: During a handover there is an undesirable disruption to user traffic. These include significant latency, high signaling message overhead and processing time, significant resources and route setup delay, high handover failures and packet loss rate.
- No Security: It is hard to maintain the same (if any) level of security when roaming across different access networks. In any given smart environment no one part of the systems has full knowledge or has full control over what is stored, sensed or communicated.

c. Human-Computer Interaction Issues

No contemporary human-computer interaction models, whether command-line, menu-driven, GUI-based, is appropriate for ubiquitous computing. The “natural” interaction paradigm suitable for a fully robust ubiquitous computing is yet to emerge.

d. Fixed Configuration of Hardware and Software

Today’s operating system, like Dos and UNIX, assume a relatively fixed configuration of hardware and software at their core: This makes sense for both mainframes and personal computers, because hardware or operating system software cannot reasonably be added without shutting down the machine. But in an ubicomp world, local devices come and go, and depend upon the room and the people in it. New software for new devices may be needed at any time, and it will not be possible to shut off everything in the room at once. This suggests that new communication protocols that explicitly recognize the concept of machines that move in physical space must be created.

e. Privacy

This is one of the social issues that ubicomp will engender. Hundreds of computers in every room, all capable of sensing people near them and linked by high-speed networks, have the potential to pick and divulge people have valued secrets. Just as a workstation on a local-area network can be programmed to intercept messages meant for others, a single rogue tab in a room could potentially record everything that happened there [1].

f. Security Issues

Security in its broadest definition is widely accepted to cover the three main properties of confidentiality, integrity and availability. Confidentiality is concerned with protecting the information/service from unauthorized access; integrity is concerned with protecting the information / service from unauthorized changes; and availability is concerned with ensuring that the information/service remains accessible. Confidentiality and integrity are essentially about encryption and decryption. Encryption, decryption and hence authentication in a smart space are complicated by the characteristics of the environment. Key to these environments is the largely decentralized and dynamic nature of principals and their interactions, due to the transient nature of relationships among principals roaming between administrative domains (principal is the term used in security to signify the entities, people, agents, devices, etc of interest). For authentication, public key infrastructures (PKI) rely on a certification authority (a trusted third party) hierarchy that can verify the principal carrying the key is the owner of the key. The rigid structure of this fixed hierarchy of authorities has limitations for a decentralized environment where two entity’s certification hierarchies may not intersect [2].

Availability also suffers from new problems in the smart environment domain. There could be a denial of service attacks which might affect the communication channel. For instance, a battery with limited power could be made to keep awake till all its power is completely used up. Attack of this nature often leads to intermittent service failure in the system.

It should be clear from the discussion so far that there is a need for new security mechanisms to provide a more dynamic view of security for the dynamic environments under consideration here.

3. APPROACHES TO COMPUTING

Software vendors develop generic programs without having any specific person or task in mind. An end-user, in order to perform a computing task must have to adjust this generic program to suit his intent. To customize generic software manually by non-expert users is a difficult task. Frameworks and tools that allow the composition of semantically annotated web services to achieve user-specified tasks are the automatic composition and semi-automatic composition approaches.

- Automatic Services Web Composition: This approach in general proposes user interaction methods and tools that capture the user's intent and translate it into some abstract semantic service description (user task) ([3], [4], [5], [6]). It works by forming a composite service after discovering the available services. The composite service so formed will semantically match as much as possible with the user- task. The composite service is then executed to perform the user-task. This approach demands less user interaction but does not perfectly capture the user's intended task as a result of heterogeneity and complexity of various computing environments.
- Semi-automatic or Interactive Web Service Composition: This approach proposes user interaction methods and tools that capture the intent of the user interactively and continuously during the composition process of the composite task [7].

Two other striking approaches are:

- Recombinant Computing Approach: The recombinant approach was developed in Xerox PARC and uses a model in which each computational entity on the network is treated as a separate component [8]. The presence of new components is detected through the use of dynamic discovery protocols, and it relies upon a mobile code framework to deliver to implementations of components needed at run-time.
- Task-driven computing approach: Task-driven computing approach aims at empowering non-expert users with the ability to perform complex tasks in information-rich, device-rich, and service-rich environments using mobile devices [9]. In this approach the functionality of available resources are exposed as semantic web services. The user discovers these services and composes his application.

a. Task-Driven Programming

The end-user encounters a dynamic range of computing resources in the smart space which he will want to use to perform his computing tasks such as writing documents, reading mails, controlling home appliances, accessing information, etc.. The end-user needs to customize the generic programs written by software vendors to suit his specific tasks. It is a difficult task to do because today's computing infrastructure does not support this model of computing very well. This is because computers interact with users in terms of low level abstractions: application and individual devices. Today, a mobile user that wants to use the computing resources of a different environment has to manually figure out how to perform the computing task. It becomes more difficult if he has to continue a task that was started in another environment. In both cases he has to:

a. keep track of his computing context (usually mentally),
b. find out what computing resources he has access to in the new environment,
c. find out what kinds of software services/applications are installed and which should be used to realize a computing task,
d. configure the chosen services to re-establish his computing text, often involving setting application options, moving and transforming necessary flies, initializing software services and co-ordinating multiple computing devices.

b. Shortfalls of Manual Task Operations

The current trend involving manual operations when a user is performing a task is unacceptable for the following reasons:

1. It does not scale with increasing numbers of service and computing devices: The knowledge and time required of users today will increase dramatically as more software services and computing resources become available.

- It does not scale with increasing user mobility: Manual configuration costs time. If a user remains in a computing environment for only 15 minutes, he would not want to spend the first 10 minutes restoring computing contexts manually.
- It does not tolerate change or failure of computing resources: The services available in an environment change in the presence of mobile devices, such as laptops and I/O devices. Software services can be installed or removed without user's knowledge. Proximity-based networking (such as IrDA and Bluetooth) leads to dynamically changing networks. Failures of services and networks can change the availability of computing resources.

Manual configuration is simply too costly in this setting of transient services, networks and devices. As noted, in today's world much of a mobile user's attention is wasted on low-level activities such as managing groups of computing devices, mentally keeping track of computing context and manually re-instantiating it, and figuring out how to accomplish a task using available services. This is, for sure, frustrating and time wasting.

c. What Task-Driven Computing Approach Offers

The task-driven programming approach frees the user from the afore-mentioned low-level activities/operations. The key idea behind this approach is that the system should take over many low-level management activities so that users can interact with a computing system directly in terms of high-level tasks that they wish to accomplish. This new approach centers on the following elements:

- system-maintained, globally-accessible abstract computing tasks and task states;
- new types of services that support automatic configuration;
- automatic selection and configuration of service co-alitions to accomplish computing tasks and re-establish computing contexts;
- proactive guidance to help a user accomplish a task and compensate for resource limitations in an environment;
- automatic management of dynamic service and resource conditions.

Making computing contexts globally accessible frees users from having to remember them mentally. Automatic collection of environment information frees users from having to locate accessible services in an environment by hand. Automatic task-based service discovery and configuration makes rapid configuration possible, even in the presence of large dynamic collections of devices and resources.

The new approach will significantly reduce the attention and knowledge requirements of mobile users, and make possible automatic adaptation to change and failure. If fully realized, a user would be able to walk into any environment and have all service configured automatically so he can resume his computing tasks. He need not care anymore about issues such as data location, data types, computing contexts, applications and configurations [10].

4. ARCHITECTURE OF A TASK COMPUTING ENVIRONMENT (TCE)

The general architecture of a task computing environment is shown in fig. 1.1. It is made up of four layers namely: realization layer, service layer, middleware layer and presentation layer [11].

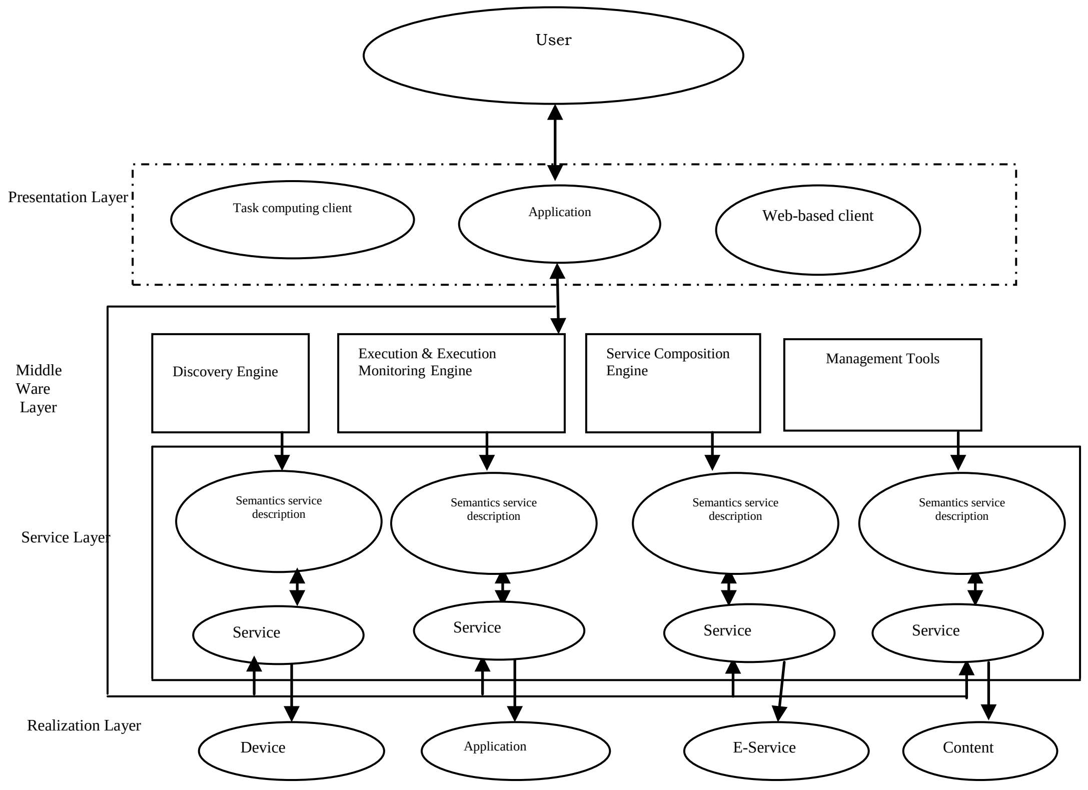

Fig 1.1: The general architecture of a task computing environment. Courtesy: Kalofonos & Reynolds (2006)

- **Realization Layer:** This is the bottom-most layer that is made up of the universe of devices, applications, e-services and content, where all functionality available to the user originate.

- **Service Layer:** The various sources of functionality are made computationally available as services. The semantic descriptions of these services are meant to shield the user from the complexity of the underling sources of functionality.

- **Middleware Layer:** The middleware layer components are in charge of discovering services, deciding how services can be composed, executing services and monitoring services execution, and also enabling and facilitating a variety of management tasks, including the creation and publishing of semantically described services.

- **Presentation Layer:** The most important aspect of task computing takes place in the presentation layer. This layer uses the capabilities of the layer below in order to provide the user with a "task" abstraction of the complexity of whatever lays underneath. The presentation layer presents to the user an environment where functionality that can be transient and dynamically created can be assembled (real time) to perform user's tasks. Since the middleware components expose well defined service API's, it is possible to create custom applications in the presentation layer in any development environments that can invoke web services.

5. END-USER PROGRAMMING

And end user is a person who makes use of an application program. That person may not necessarily be a computer programmer; rather, he uses the computer as part of his daily life. End user programming is all about non-professionals writing programs. They use extension languages to do so e.g. AutoCAD has an embedded LISP language for extensions; Microsoft word for windows has an embedded basic language. Some extension languages use programming by demonstration (PBD) techniques so that users do not need to learn conventional programming languages.

a. End User Programming Architecture

The architecture for an EUP framework is shown in fig. 1.2 below and can be distinguished in five layers: (i) the user layer, (ii) the scripting layer, (iii) the semantic adaptation layer, (iv) the core distributed middle layer, and (v) the local system layer [12].

User Layer: The user layer is responsible for the interaction with the end-user. Its role is to capture the user's intent and communicate the results of the user-actions in a manner that is intuitive and easily understood. Users interact with the End-User Programming Tools (EUP- tools) to create new customized applications and scripts that are used to perform their intended tasks. The applications are native applications that provide an appropriate front-end that invokes the scripts composed by the users.

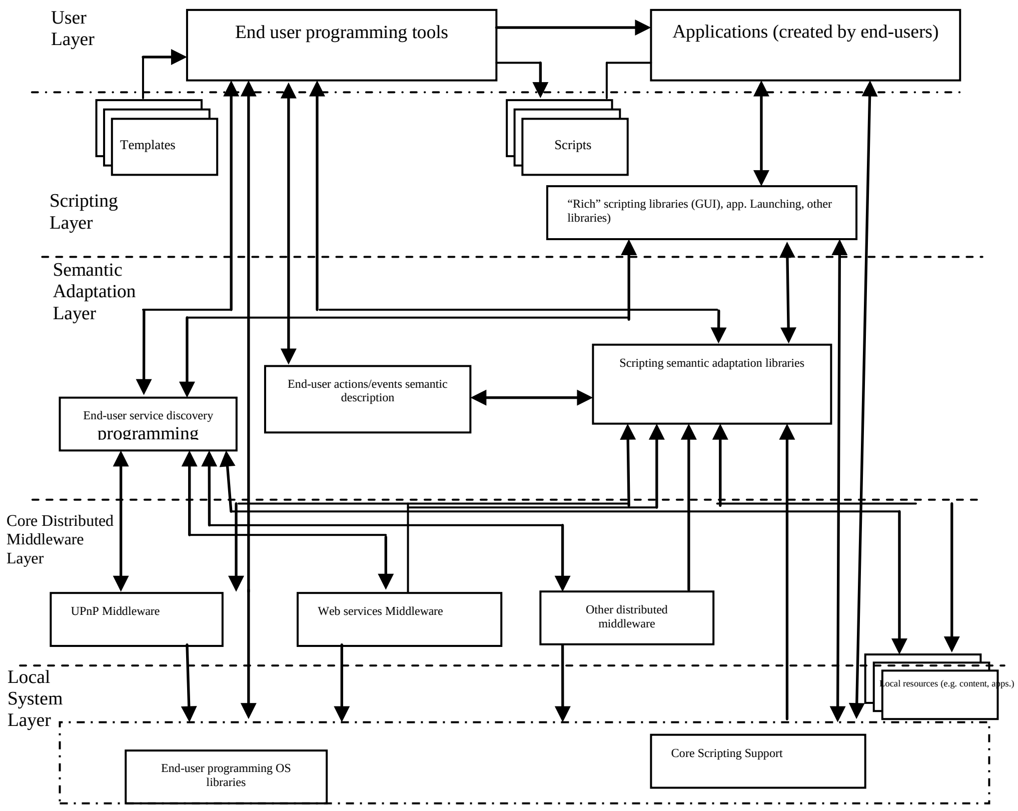
User Layer
End user programming tools
Applications (created by end-users)
Templates
Scripts
Scripting Layer
"Rich" scripting libraries (GUI), app. Launching, other libraries)
Semantic Adaptation Layer
Scripting semantic adaptation libraries
End-user actions/events semantic description
End-user service discovery programming
Core Distributed Middleware Layer
UPnP Middleware
Web services Middleware
Other distributed middleware
Local System Layer
Local resources (e.g. content, apps.)
Core Scripting Support
End-user programming OS libraries

Fig 1.2: A high-level architecture of the end-user programming framework. (Courtesy: Kalofonos & Reynolds, 2006) modified (2012)

Scripting layer: The end-user programs are essentially scripts. The scripting layer may contain scripting templates which prescribe certain compositions of available services similar to task-oriented templates [13].

These templates may be created by the provider of the EUP framework (e.g. Nokia through "Forum Nokia") and can be downloaded by the end-users. For example, a user may search with Google for a "home automation" template available for the Nokia EUP-framework and download it to her phone. The EUP-tools will load this template and use the results of the framework's service discovery to customize it and create the final "home automation" program for her home.

Semantic adaptation Layer: The semantic adaptation layer is responsible for making semantic translations between the concepts perceived by the end users and the functionality provided by the underlying system. This layer consists of the end user semantic descriptions of actions and events which describe how low-level actions and events are mapped to the user-level concepts that are exposed to end-users. They also describe how user-level actions and events can interact with each other to compose higher-level tasks. This functionality is used by EUP-tools to guide the user in the composition of tasks by limiting his available choices or by suggesting alternatives.

Core Distributed Middleware Layer: This layer includes all the low-level distributed computing middleware necessary to access devices and services which constitutes the user's smart space. The EUP-framework is built on top of this lower-layer middleware. Examples include UPnP, web services, Jinni, Salutation, JXTA, Bluetooth services, etc. The interface to the various components of this layer is known and usually standardized. Even though various components of this layer may use different methods to expose similar services, they are translated by the semantic adaptation layer to the same user-level concepts. For example, whether a networked printer is using UPnP, salutation, or Bluetooth printing profiles, the user-level concepts of "print", "select double sided", "color", "high-resolution", "out of-paper", "low-ink" are the same.

Local System Layer: The local system layer includes the core platform and OS facilities, as well as the local user resources (e.g. local applications and content). The system layer provides the native interface and core scripting support and perhaps some native OS RUP libraries. The local user resources may also be exposed by the semantic adaptation layer as services.

b. Application Development by the User

Application development refers to the process of building applications by assembling services offered by computing devices embedded in the environment.

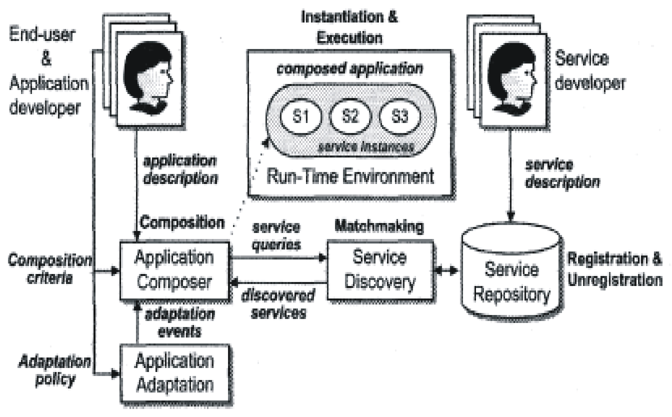

Fig: 1.3 Generic Application Composition Process.

Fig.1. 3: Application Composition Process (Courtesy: Davidyuk et al (2009))

As shown in fig. 1.3, the application composition process involves two separate actors, the end-user and the service provider. The role of the service provider is to develop and publish services which provide some functionality through a well-defined interface. The end-user's role is to create applications and utilize (i.e. interact with) applications [14].

The application composition process relies on descriptions of applications and services. An application description specifies the abstract services the application is composed of and the relationships between them (i.e. control and data flows). The application composer is responsible for decision-making in the application composition process. The application composer uses application descriptions and possible composition criteria (preferences, fidelity constraints, cost, etc.) provided by the end-user as inputs and choose available services which satisfy the given criteria. The selection of services is supported by a service discovery protocol, which is responsible for the matchmaking functionality (i.e. for matching service discovery request against the service descriptions stored in the service registry). Since the discovered set of services may potentially contain redundant instances, the application composer optimizes (i.e. further reduces) the service set and produces an application configuration (i.e. application composition plan) which satisfies the criteria given earlier. After this, the application is instantiated and executed by the run-time environment. During execution, the application can be adapted (i.e. recomposed) according to user defined adaptation policies. These policies are formed rules which trigger actions when the application or user context changes.

The vision of smart space programming has left us with some questions demanding answers. These are:

a. The very features that allow smart environments to be personalized and dynamic are the features that contribute to privacy problem. How could this issue be addressed considering the complexity of integrating numerous technologies, networks and devices?
b. In the application composition by the user, could there be alternative ways of enhancing the robustness of the system?

6. ANALYTICAL FINDINGS

In the end-user programming model examined in this paper, it is clear that they have one main deficiency or the other when evaluated in terms of usability and reliability in handling special end-user needs in a smart environment. They use extension languages to do the programming which assumes that the users are conversant with the languages, this leads users into a situation where they need to learn the new programming languages before they will be able to manipulate the system to solve their needs. Embedded LISP language for instance may seem to be a simple programming language but to the end-user may turn out to be a cumbersome language in developing systems for his use. If the user has not learned any programming language it may be very difficult for him to just adopt the language and start using it without many efforts. Microsoft Visual Basic Application (VBA) for instance may equally seem simple to deploy but its applicability in smart spaces are limited to only few devices. Its usage however also requires some measure of learning to be able to develop reasonable level of sophistication required in solving certain problems within the smart space environment.

In certain cases, some extension languages use programming by demonstration (PBD) techniques so that users do not need to learn conventional programming languages. The challenges remain the ability of the end-user to be able to learn and match one demonstration to the need that is desired so that using the steps of the demonstration solutions can be developed. But a major challenge with this type of end-user programming is that we assume that each set of problem will have a corresponding set of solution. This assumption is not substainable since in real life there are many instances of problems which are never envisaged during the development of the demonstration codes. Some of this user needs only arise due to use or change in desire and even peculiarity of user environment. Hence, there must be new ways of capturing this variation in both needs and user environment.

7. RECOMMENDATION

Having examined the challenges of the present systems and its shortcomings in providing holistic solution to end-user programming of smart spaces we make our recommendations which include.

a. Programming by trial and error (PTE)

It is a known fact that most things that the minors learn using the computer is by PTE. It is also clear that when new features are introduced in most social networking sites such as Face book, MySpace etc user manuals are not usually provided, yet, users quickly get adapted to them. The developers of such systems provide a simple navigational system directing users on possible expectations. When users are in need of any process they can easily reach at them by the combination of different steps to reach at them. This approach is therefore recommended for use in the process of developing end-user programmability which may simply require the following of some simple steps in arriving at something which may not exactly provide solution to the user but provides insight which will further guide the user in further steps and process combinations that will actually lead to what the user may actually need.

b. Action Programming

Action programming on the other hand is the programming of system by selection of actions or combination of actions which provides certain results. In this new programming proposed in this paper, each action is developed to provide specific results which can be testable in isolation. A combination of actions can also be taken which will produce result that is also testable. We believe that the user will use the actions to gather the required experience needed in the provision of end-user application that need to be programmed by the end-users themselves.

8. CONCLUSION

In the light of the inherent challenges of present end-user program development strategies identified in this paper, we believe that the end-user programming recommendations made would be adopted in the process of developing end-user programming systems for smart spaces. Issues such as usability, flexibility matters are of more concern to end-users than even complexity and efficiency. End-users seem to be more satisfied with the system they develop to their taste which may not be as efficient as the one developed elsewhere with higher efficiency but not directly targeted to their needs. In all, users must be made to be major stakeholders in the development of systems that are mobile devices-based. End-User programming of Smart-Spaces must also make usability its watchword in adopting methods of its developments.

In all, this paper has investigated a theoretical possibility of developing a task-driven model which can be deployed in providing end-user programmability of smart-spaces using mobile devices. It also has made analysis of the present systems and models that are used to be able to evaluate their strength and pitfalls in realizing they said objective within a ubiquitous computing environment. Recommendations have also been made which if developed we believe will proffer better solution to the realization of end-user programmability of smart-spaces using mobile devices.

REFERENCES

[1] Weiser, Mark (1991). “The computer for the 21st Century” http://www.ubiq.com /hypertext/weiser/sciAmDraftz.html

[2] Nixon Paddy, Gerard Lacey and Simon Dobson (eds), Managing Interactions in Smart Environment, Springer Verlag Press, pp.243, December 1999.

[3] Wu Dan, Sinin Evren, Hendler James, Nau Dana, and Parsia Bijan (2003), Automatic web services composition using SHOP2. In 13th international conference on automated planning and scheduling, workshop on planning for web services, Trento, Italy, June 2003.

[4] Sycara Katia, Paolucci Masimo, Ankolekar Anupriya and Srinivassan Naveen (2003): Automated Discovery, Interaction and Composition of Semantic Web Services: Web Semantics, science, services and Agents on the World Wide Web, pp. 27-46, and 2003.

[5] Majithia Shalil, Walker W. David, Gray W.A., (2004) A Framework for Automated Service Composition in Service-Oriented Architecture: In 1st European Semantic Web Symposium, 2004.

[6] Mokhtar Sonia Ben, Georgantas Nikolaos, issarny Valerie (2005): Ad-hoc composition of user tasks in pervasive computing.

[7] Sirin Evren, Hendler James, Parisia Bijan (2003): Semi-automatic composition of web services using semantic description; In web services, modeling, architecture and infrastructure workshop in ICEIS 2003, Angers, France, April 2003.

[8] Edwards W. Keith, Newman W. Mark, Sedivy Z. Jana (2001): The case for recombinant computing, Xerox Palo Alto research centre technical report CSL-01-1. April 20, 2001.

[9] Masuoka Ryusuke, Parsia Bijan, Labron Yannis and Sirin Evren (2003), Ontology-Enabled pervasive computing applications, IEEE intelligent systems, vol. 18, pp: 68-72, Sep./Oct. 2003.

[10] Garlan Zhenyu Wang David (2000), CMU-CS-00-154, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15217.

[11] Song Zhexuan, Labrou Yannis and Masuoka Rynsuka (2004), Dynamic services recovery and management in task computing, pp. 310-318, mobiQuitous 2004, August 22-26, Boston, MA, 2003.

[12] Kalofonos N. Dimitris and Reynolds D. Franklin (2006), Task-Driven End-User Programming of Smart Spaces Using Mobile Devices, Nokia Research Center Cambridge, January 10, 2006.

[13] Newman W. Mark, Ssedivy Z. Jana, Neuwirth M. Christine, Edwards W. Keith, Hong I. Jason, Izadi Shahram, Marcelo Karen, Smith F. Trevor (2002), Designing for Serendipity: Supporting End-User configuration of Ubiquitous Computing Environment, in Proceedings of DIS'02, June 2002.

[14] Davidyuk, O., Georgantas, N., Issarny, V. Riekk, J. (2009), MEDUSA: middleware for End-User Composition of ubiquitous applications, and ARLES research team, INRIA Paris-rocquencourt, Domain de Voluceau, Le Chesnary, 781.5