Large Scale Testing of Pervasive Computing Systems Using Multi-Agent Simulation

Nanjangud C Narendra[1]

[1]IBM Software Labs India
Airport Road, Bangalore, India
narendra@in.ibm.com

Abstract — *The rapid growth of handheld computing devices such as mobile phones, PDAs or palmtops is paving the way for the emergence of pervasive computing systems. Just as in the case of traditional computing systems, pervasive computing systems need to be tested in the large before they can be deployed in the field. As opposed to traditional computing systems, however, large-scale testing of pervasive computing systems requires the presence of dozens (perhaps hundreds) of physical devices, arranged together in a network, executing a variety of complex scenarios. In order to reduce the cost of such testing, it would be better to simulate the operation of a pervasive computing system using well-known techniques from multi-agent simulation, by representing each (hardware or software) component of the system as a software agent. In this paper we describe our ongoing work, where we extend our earlier work on multi-agent simulation, for pervasive computing systems. Since adaptation in pervasive computing systems is expected to be common, we also show that our simulation technique can model adaptation.*

1 Introduction

Pervasive computing [9] is an era of computing with two special distinguishing characteristics: (a) users will no longer be tied to the desktop paradigm and (b) users will become increasingly mobile. This will result in usage patterns that are quite different from what we have known traditionally as workflow or office work. The new usage patterns will be increasingly defined by large numbers of low-power devices (mobile phones, PDAs, palmtops) co-existing with desktop computing systems; disconnected operation, and rapid and ad-hoc changes in usage patterns.

In order for pervasive computing to become a reality, large-scale testing of pervasive computing systems under varied and complex scenarios becomes necessary. Such testing would help to “scale up” the small-scale laboratory testing currently underway in most R&D organizations conducting pervasive computing research.

For example, if we were to consider a 200-bed hospital staffed by 40 nurses, 40 doctors and specialists and 10-15 administration staff, one could conceivably imagine an environment of about 100 pervasive computing devices such as mobile phones, PDAs, palmtops, etc., co-existing with the desktops comprising the hospital’s own computer system. This, of course, does not take into account the complexity of the tasks executed by the different users, such as medical tests, surgeries, emergency procedures, etc. It would be quite difficult to test and verify the performance and other characteristics of such a large-scale system without either involving a real hospital as a "guinea pig" for testing, or recreating a similar laboratory environment by purchasing several pervasive devices and enlisting the assistance of dozens of volunteers. Hence it is essential to investigate techniques such as multi-agent simulation [12], since the pervasive computing system can be modeled as a collection of cooperating agents. Moreover, even if a "guinea pig" could be found, simulation is needed in order to at least obtain an initial assessment of the efficacy of the system before deploying it on the "guinea pig".

Analytical models for such systems either cannot be developed with any degree of uncertainty, or would probably become unsolvable, given the complexity of the system. The objective of multi-agent simulation would be to develop "rules of thumb" that would assist system designers in developing large scale pervasive computing systems. These "rules of thumb" would evolve based on the results of simulation.

This paper is organized as follows. In the next Section, we discuss the requirements for simulating pervasive computing systems. In Section 3, we describe our model of the architecture of the system that we wish to simulate. Section 4 describes our multi-agent simulation architecture. In Section 5 we describe our simulation technique in detail. We present some related work relevant for our paper in Section 6, while the paper concludes in Section 7 with suggestions for future work.

In what follows, for the sake of simplicity, we also refer to the "pervasive computing system" as "the system".

2 Requirements for Simulating Pervasive Computing Systems

Simulation of a pervasive computing system raises several challenges. In particular, in addition to the behavior of software artifacts such as desktop systems, and software running on handheld devices, the actual characteristics (behavior/performance/capacities) of the devices themselves need to be modeled.

In this regard, the notion of *contexts* becomes important. Context is any information that characterizes the interaction between a component of the system (such as a user or a device) and its environment [9]. The behavior of an entity is therefore determined by the context that the entity operates in. For example, a doctor carrying a PDA can receive picture messages containing some images, whereas if he were to carry only a mobile phone, he may need to be only sent a text message stating where he could access the images. Similarly, if the doctor is in the operating room with a patient and cannot read detailed messages regarding his/her other patients, the system should only send him/her an informational message, specifying the location from where he/she can download the information later.

Since pervasive computing systems are meant to be used by multiple users executing - jointly or otherwise - several activities, it is essential that their activities be suitably modeled and simulated – hence a need for workflow-based modeling of user activities as in [1,4]. Moreover, since usage patterns of users' activities are usually ad-hoc in pervasive computing systems, this leads to continuous workflow adaptation, which should also be modeled. In pervasive computing systems, workflow adaptation is of two types [1,5]:

- *Functional* – changes in users' requirements, leading to changes in their usage patterns
- *Architectural* – changes in availability of certain resources at certain locations and times, leading to changes in usage patterns themselves

To summarize, an effective simulation of a pervasive computing system should be: *workflow-based*, in order to capture and simulate users' activities, be able to model *workflow adaptation* due to constantly changing usage patterns, and *context-aware* in order to account for the context in which the users and resources are situated.

In addition, as already stressed in Section 1, the simulation environment should be able to model the system components as agents cooperating with each other to execute the defined workflows. In order to model workflow adaptation, the simulation environment would need to start, stop, suspend or resume the operations of one or more of these cooperating agents. For analysis purposes, it would need to possess a mechanism for collecting and manipulating execution data, so that appropriate conclusions about the performance and efficacy of the system can be drawn. This means that the simulation environment should able to model and control the *lifecycle* of each agent being simulated. In other words, the simulation environment must be *lifecycle-based*.

Therefore we see that two types of architectures need to be modeled:

- The *system architecture* to be simulated (which is actually the pervasive middleware), has to be defined so that the various components and their interconnections can be defined and modeled. This architecture should be workflow-based, adaptation-aware and context-aware. We will be using our 3-tier system architecture for pervasive computing systems, earlier introduced in [1], since it meets these requirements. This architecture will be described in Section 3.
- The *simulation architecture* itself, which is used to build the simulation environment that will simulate the components and behavior of the system architecture. This should be a multi-agent and lifecycle-based environment. For this reason, we extend from our lifecycle-based multi-agent simulation system described in [2], for simulating pervasive computing systems.

3 System Architecture

In order to simulate the system, we need a conceptual description of what the system will "look like". For this we first need to develop a conceptual model of the different entities in the system, and this is depicted in Figure 1 (also see [1]).

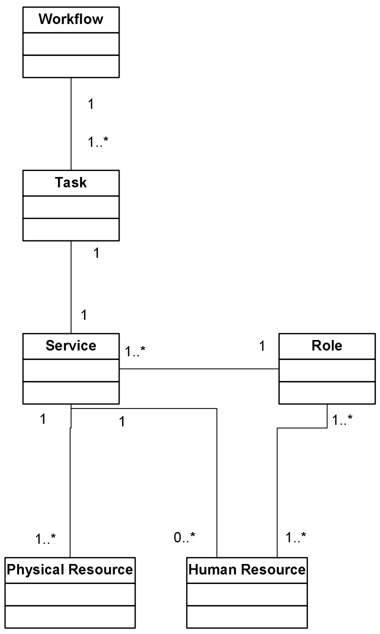

Figure 1: Conceptual Model – mapping of workflow tasks to services, and dynamic binding of services to resources

As per the conceptual model of Figure 1, a workflow task is mapped into a service. The service is provided by a role. The role is capable of being performed by one or more human resources. In addition, the service would also require certain physical resources for its implementation. Mapping of the task to the service, and of the service to the appropriate role, can be specified at design-time as per users' functional requirements. However, mapping of the service to the actual human and physical resources is done at run-time, in keeping with service orientation. This dynamic binding is therefore dependent on the context in which the binding occurs, as already explained in Section 2.

Our system architecture (derived from [1]) will therefore model contexts at three different levels, in keeping with the conceptual model of Figure 1. The topmost level is the *E-context* (standing for Environmental context) level, which will represent the contextual information of the overall environment, such as number of users, number of physical resources, users' functional and non-functional requirements, overall workflow model consisting of sequence of tasks. The next level is the *S-context* (Service context), which models the contextual information of each Service, such as task that the service is mapped onto, human and physical resources needed for service execution, succeeding and preceding workflow tasks of the task. The last level is the *R-context* (Resource context), which models the detailed resource information such as resource capacity (in the case of physical resources such as pervasive devices), roles performed by human resources.

The system architecture is pictorially depicted in Figure 2, and consists of the following layers:

- The topmost layer is the *Environment* layer, which models the overall environment of the system, including users' functional and non-functional requirements, and overall workflows meeting those requirements, which are then translated into the services (workflow tasks) whose execution will meet the requirements. It consists of the following modules:

- ➢ The *Requirements Management* module stores and manages the users' functional and non-functional requirements
- ➢ The *E-Context Management* module is responsible for setting and maintaining the E-context information; it also interacts with the S-Context Management module at the Service layer for obtaining and aggregating contextual information from the Service layer
- ➢ The *Functional Adaptation* module implements functional adaptation (see Section 2) as per changes in users' functional requirements

- The middle layer is the Service layer, which models the tasks executed by the users in the system in the form of the services derived from the users' requirements; these services which are bound to the physical and human resources needed for service execution. It consists of the following modules:
  - ➢ The *S-Context Management* module manages and maintains the S-context information; it also interacts with the R-Context Management module at the Resource layer, for obtaining and aggregating contextual information from the Resource layer. The S-Context Management module also sends its own contextual information up to the E-Context Management module at the Environment layer.
  - ➢ The *Service* Module represents the invocation and execution of the services represented as activities in the workflow model
  - ➢ The Role Management module is responsible for managing the assignment of roles to the activities that represent the services
- The bottom layer is the *Resource* layer, which models the behavior of the physical and human resources in the system to which the services are bound. It consists of the following modules:
  - ➢ The *R-Context Management* module manages the R-context information, and relays the information up to the S-Context Management module
  - ➢ The *Service Binding* module is responsible for the dynamic binding of services to physical and human resources, as per the conceptual model depicted in Figure 1.
  - ➢ The *Architectural Adaptation* module implements adaptation caused by changes in users' non-functional requirements, such as resource availability changes.

Our system would operate as follows. Based on users' functional requirements, the behavior of the system would be modeled as a set of workflows. Each task in a workflow is mapped onto a service provided by a role in the system. In addition, the service would also require certain physical resources (such as pervasive devices) for its implementation. Mapping of the task to the service, and of the service to the appropriate role, can be specified at design-time as per users' functional requirements. However, mapping of the service to the actual human and physical resources is done at run-time, as per users' non-functional requirements.

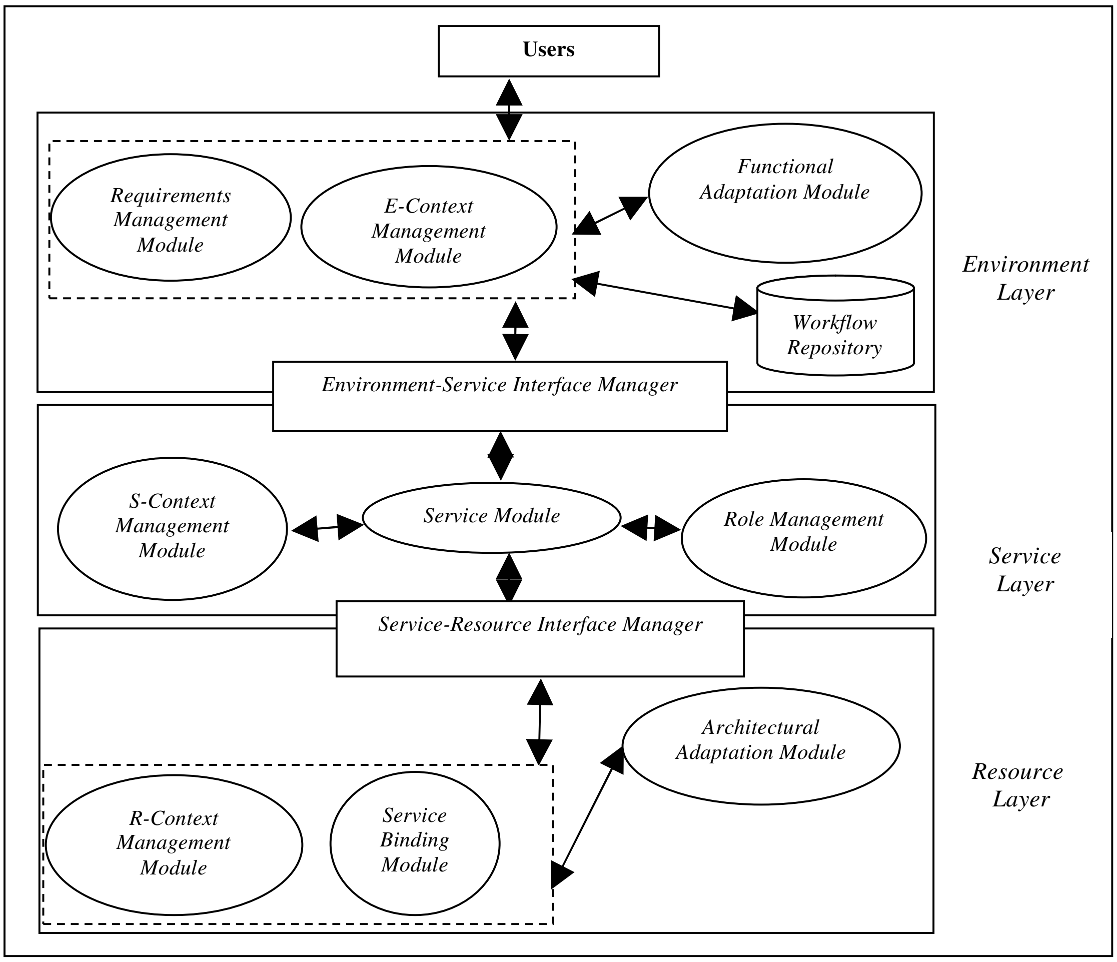
Users
Requirements Management Module
E-Context Management Module
Functional Adaptation Module
Workflow Repository
Environment Layer
Environment-Service Interface Manager
S-Context Management Module
Service Module
Role Management Module
Service Layer
Service-Resource Interface Manager
Architectural Adaptation Module
R-Context Management Module
Service Binding Module
Resource Layer

Figure 2: System Architecture

4 System Architecture

In order to model the simulation architecture, we are leveraging from our earlier work on multi-agent simulation [2], as already explained in Section 2. Basically, each component in the system is modeled as a software agent possessing some qualities essential to agents: *autonomous* (can function without active user intervention), *proactive* (can adjust its behavior according to external stimuli), *reactive* (capable of reacting to external stimuli), and *social* (can interact with other agents). Hence each Agent can model one of the components of our system – a user, a desktop system, a pervasive device, a component of the middleware as depicted in Figure 2 [1], etc. The local behavior of each agent under particular inputs is modeled by Event-Condition-Action (ECA) rules for each input [6], where each input to an agent is modeled as an event triggering a response. The overall system behavior is therefore evolved as a combination of these local behaviors. The ECA rules are flexible enough to even model the workflow tasks executed by users as described in Section 2. The simulation architecture is depicted in Figure 3.

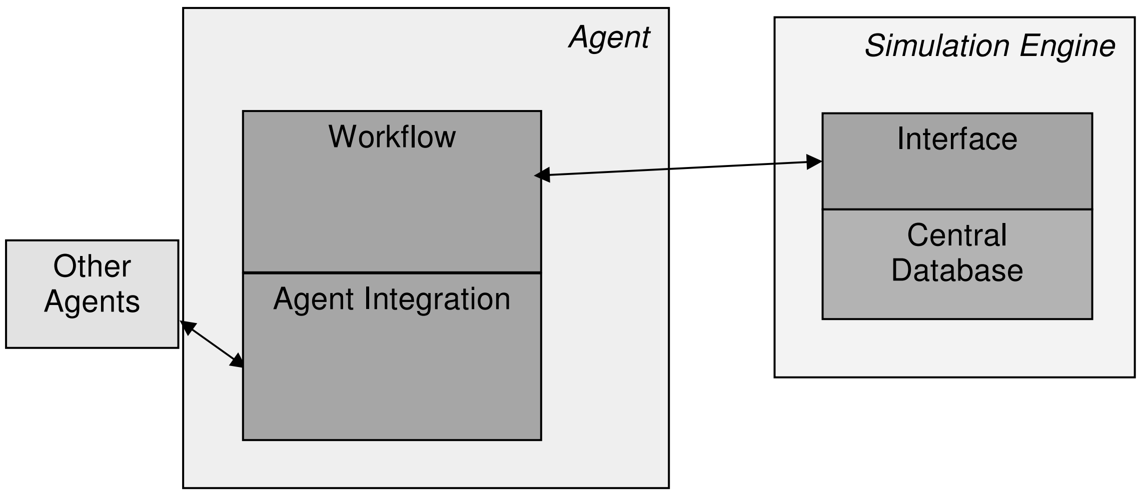

Figure 3: Simulation Architecture – Simulation Engine Interacting With An Agent

In Figure 3, the *Simulation Engine* is a software program that maintains and executes the workflow models consisting of sequences of task executions modeled by the ECA rules. This is done by triggering the execution of each *Agent* (which represents a component in our system) as per the workflow model; the Agent would respond to the trigger as per its ECA rules. The Simulation Engine consists of two components: the Interface module is responsible for triggering each Agent for task execution; and the Central Database is responsible for maintaining the workflow models, and the simulation results, which can be used later for visualization or analysis (as described in more detail in Table 1 in Section 5.1 below).

Each Agent in the system is modeled as shown in Figure 3, with the following modules: the Workflow modules is responsible for interacting with the Interface module of the Simulation Engine, from where it receives triggers and to which it responds to the Simulation Engine as per its predefined ECA rules; the Agent Integration module helps the Agent in interacting with the other Agents in the system if needed, as per its ECA rules (this is needed for hierarchical workflow modeling, described in more detail in Section 5.1 below). Hence in keeping with the agent's autonomous status, the Simulation Engine is responsible for the overall workflow execution, while each agent is responsible for its respective task execution.

The simulation proceeds as follows. The overall workflow execution model is defined by the Simulation Engine and consists of a sequence of workflow tasks, with each task to be executed by an agent. As per the workflow model, the Simulation Engine invokes the agent by sending it a trigger message, which is the Event part of the agent's ECA rule. The result of the task execution, which is the Action part of the ECA rule, is sent back to the Simulation Engine. In case of hierarchical workflow modeling, the agent may in turn trigger other agents via the Agent Integration module, and obtain results from their respective Action parts. Based on the result of the Agent's task execution and the workflow model, the Simulation Engine will invoke the next agent, and so on, until the workflow execution completes. At every point, the simulation results are collected and maintained in the Central Database for future analysis.

Adaptation, in the form of *exceptions*, can also be simulated via the ECA rules, which will model the local behavior of each component when exceptions occur. However, the interesting aspect of modeling adaptation, is to investigate the global behavior of the system as a combination of these local behaviors under exceptions. In this context, we are also investigating the so-called "butterfly effect" [3], viz., whether small perturbations in local behaviors would result in large deviations in the overall global behavior of the system. We briefly describe simulation of adaptation in Section 5.2.

In the next Section, we will describe our simulation technique in detail.

5 Simulation Technique

5.1 Simulating the System Architecture

As per our simulation approach the Simulation Engine of Figure 3 would engage in the creation, running, controlling and destruction of the various workflow execution scenarios that would characterize the behavior of the pervasive computing system. This means that the simulation model should be lifecycle-based. To that end, we leverage the approach that we proposed earlier [2], and which was based on a lifecycle approach towards multi-agent simulation based on software process modeling [11].

Our lifecycle-based simulation approach is as follows [2]:

Hierarchical modeling: users' workflow processes can be modeled hierarchically, via techniques such as specialization and decomposition as in [10]. In this approach, all processes in an organization are represented hierarchically, with processes lower down in the hierarchy being specializations of the root process. Each process is, in turn, decomposed into sub-processes recursively until the lowest level atomic task is represented.

ECA-based simulation scripts: as already explained in Section 4, each atomic task execution by an agent can be expressed as a simulation script in ECA form. An example from the hospital domain for a nurse's PDA, would be:

When (EVENT = "arrange for X-Ray test of patient P")
If (CONDITION = "X-Ray technician available")
Then (ACTION = "schedule X-Ray for time = 4 PM; date = 22nd Dec 2004")

The above ECA rule states that the nurse will schedule an X-Ray test for a patient P, under the condition that the X-Ray technician is available at a convenient time. The availability information of the X-Ray technician would form part of the R-Context (physical and human resources are discussed in more detail later), while the task of X-Ray scheduling forms part of the S-Context. The CONDITION and ACTION parts of the ECA rule would also make use of the environmental information stored in the E-Context in order to determine the availability of an X-Ray technician.

The workflow processes are then modeled as combinations of the atomic tasks represented via these ECA rules.

- *Lifecycle-based modeling:* as depicted in Table 1 below, the Simulation Engine would need to monitor the agents throughout their entire lifecycle. This ranges from metamodeling for developing the metadata necessary for modeling the events, conditions and actions; right up to running the simulation, recording the results for playback and analysis, and finally archival.

Process Step in Lifecycle	Description
Metamodeling	As in [1], defining ontologies (meta-data) for agent interactions and workflow models
Modeling	Capturing and representing the choreographing workflow models/instances, in computer-understandable form
Analysis	Defining and modeling static and dynamic (including semantic) properties of the workflow models
Simulation	Symbolically executing the workflow models in order to observe their behavior
Redesign	Redesigning workflow models based on results of simulation
Visualization	Graphical visualizations of workflow execution and their performance characteristics
Prototyping, walkthrough and performance support	Incremental enactment of workflow executions for evaluation purposes
Administration	In a manner similar to that described in [12], user monitoring of the simulation system
Integration	Simulation of external tools/services/devices into the system, and how they would integrate with each other
Monitoring, recording and auditing	Collecting and measuring execution data
History capture and replay	Recording the enactment history of the workflow executions, in order to analyze them either wholly or in part
Articulation	Diagnosing, repairing and rescheduling workflow executions that have failed
Evolution	Using performance data to incrementally and iteratively enhance and restructure workflow model definitions
Process Asset Archival & Management	Organizing and managing the collection of workflow model definitions and instances of workflow executions.

Table 1: Multi-Agent Simulation Lifecycle

- *Resource Modeling:* resources in our system are of two kinds: human and physical. Appropriate representation of resource properties is essential for accurate ECA-based modeling of the system. In the case of physical resources (e.g., devices such as PDAs), some important properties are resource memory size, processor power, screen size, etc. Some key properties of human resources are capabilities (whether the resource is trained to be a nurse, technician, doctor, specialist, etc.), availability (at what time of day or week the resource is available), etc.

Resource properties in our system architecture are to be maintained by the R-Context Management module, which is expected to develop ontologies (metadata) for representing this information. This is expected to be done during the Metamodeling phase of the simulation lifecycle depicted in Table 1.

5.2 Simulating Adaptation

Changes in usage patterns (workflows) are expected to happen constantly in pervasive computing systems, first, due to changes in users' functional requirements, and second, due to changes in resource properties (such as availability or capacity). As explained in Section 2, we call the former *functional* adaptation, whereas the latter is called *architectural* adaptation. Since adaptation also involves taking actions in response to events, it can also be simulated using the ECA rules described above. The main difference here is that adaptation-related ECA rules would operate on the "normal" execution-related ECA rules. That is, adaptation rules would effect changes in the execution of already running ECA rules.

At the most elementary level, workflow adaptation consists of a combination of the following atomic actions [13]: adding a task, and deleting a task. Example ECA rules for task addition and deletion from the hospital domain, are (the impact of adaptation on the existing workflow is discussed later below):

When (EVENT = "adaptation needed for workflow")
If (CONDITION = "additional biopsy of patient needed")
Then (ACTION = "add task T - Biopsy")

When (EVENT = "adaptation needed for workflow")
If (CONDITION = "patient condition has become stable")
Then (ACTION = "delete task T – MRI Scan")

It is easy to implement the above rules for a workflow that has not yet started executing. However, for mid-flight changes, i.e., changing an already running workflow, existing tasks need to be either aborted or rolled back (as per rules defined in [13]), before adaptation can be implemented. Typically, tasks that are currently executing can either be aborted or rolled back, whereas tasks that have completed execution will need to be rolled back. Tasks that have not yet started execution can only be aborted. Task abort and rollback actions can also be encoded into ECA rules in a manner similar to that for task addition and deletion.

6 Related Work

Simulation modelling for pervasive computing systems is a new and growing research area, motivated by the need for testing pervasive computing systems before they are deployed in the field. In this section, we describe some related work that is most relevant for our paper.

A simulation model for self-adaptive applications in pervasive computing systems was proposed in [14]. This paper presents a design of a simulation model of contexts so as to test the context logic of a context-aware application, by allowing sensor data to be produced from a variety of locations.

One of the most well-known simulators for pervasive computing systems, is UBIWISE [15]. This simulator focuses on creating interactive ubiquitous scenarios for mobile devices, in order to test how they would behave in the real world before the devices themselves are developed. However, this approach is not as comprehensive as our multi-agent simulation approach, in that it does not perform simulations at the level of workflow-based user interactions.

In the related area of sensor networks, several testbeds, such as EmStar (http://cvs.cens.ucla.edu/emstar/), MoteLab (http://motelab.eecs.harvard.edu/), and Gnomes (http://www-old.ece.rice.edu/~cavallar/cmclab/) have been developed. These are complementary to our approach, since they can more accurately capture data at the resource layer of Figure 2, which can enrich our multi-agent simulation approach.

7 Conclusions and Future Work

In this paper, we have discussed the challenges involved in large scale testing of pervasive computing systems, and we have proposed multi-agent simulation as one technique for addressing these challenges. We have also described our ongoing work on a workflow-oriented multi-agent simulation technique for testing the varied usage scenarios that arise in pervasive computing systems.

Future work will focus on implementing our simulation system and testing it using several real-life examples. For this, we will be focusing on the medical and transportation domains. We have used the medical domain in our paper to illustrate our ideas. In the transportation domain also, the shipping of goods from suppliers to customers consists of complex workflows involving the usage of several pervasive computing devices such as RFID tags, mobile phones and PDAs. These workflows are also subject to constant change, given the dynamic nature of transportation, especially across state and national boundaries. For example, changes in customs and taxation regulations, which would impact transportation workflows, can be represented using the adaptation ECA rules as described in Section 5.2.

Acknowledgments

The author wishes to thank Umesh Bellur, Zakaria Maamar, Wathiq Mansoor and Manu Kuchhal for their feedback. The author also wishes to thank his manager, K. Muralidharan for supporting his work. Thanks are also due to the anonymous reviewer, whose feedback significantly improved the quality of the paper. Other company (i.e., non-IBM), product and service names may be trademarks or service marks of others.

References

[1] U. Bellur and N.C. Narendra, Towards Service Orientation in Pervasive Computing Systems, *Proceedings of ITCC 2005, Pervasive Computing Track,* 2005

[2] N.C. Narendra. eAgents: An Approach for Modeling and Simulating Multi-Agent Systems. *Proceedings of AMCIS 2003*

[3] Butterfly Effect. See http://www.cmp.caltech.edu/~mcc/chaos_new/Lorenz.html

[4] J.E. Bardram and H.E. Christensen, "Open Issues in Activity-based and Task-Level Computing", available from http://www.pervasive.dk/publications/files/open_issues_bardram.pdf

[5] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications,* 8(4), August 2001.

[6] G. Kappel, S. Rausch-Schott and W. Retschitzegger, *Coordination in Workflow Management Systems - A Rule-based Approach,* Springer LNCS 1364, 1998

[7] W. Scacchi, and P. Mi, "Process Life Cycle Engineering: A Knowledge-Based Approach and Environment," Intelligent Systems in Accounting, Finance and Management, Vol 6: 83-107 (1997)

[8] D. Chakraborty, A. Joshi, T. Finin and Y. Yesha. "Towards Distributed Service Discovery in Pervasive Computing Environments," IEEE Transactions on Mobile Computing, July 2004

[9] A.K. Dey, G. D. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," Human-Computer Interaction Journal, Special Issue on Context-Aware Computing, 16, 1, 2001.

[10] T.W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C.S. Osborn, A. Bernstein, "Tools for inventing organizations: Toward a handbook of organizational processes," Management Science, Vol. 45, No. 3, March 1999

[11] W. Scacchi, and P. Mi. Process Life Cycle Engineering: A Knowledge-Based Approach and Environment. *Intelligent Systems in Accounting, Finance and Management,* Vol 6: 83-107 (1997)

[12] M.L. Griss and R. Letsinger. Games at Work: Agent-Mediated E-Commerce Simulation. HP Labs Technical Report HPL-2000-52, available from http://www.hpl.hp.com/techreports/2000/HPL-2000-52.pdf

[13] N.C. Narendra. Design Considerations for Incorporating Flexible Workflow and Multi-Agent Interactions in Agent Societies. *Journal for Association of Information Systems,* 1, 2003

[14] M.C. Huebscher and J. A. McCann. Simulation Model for Self-Adaptive Applications in Pervasive Computing, *Proceedings of 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS'04),* Aug 30th – Sep 4th, 2004

[15] J.J. Bartin and V. Vijayaraghavan. UBIWISE, A Ubiquituous Infrastructure Simulation Environment, *HP Labs Technical Report HPL-HP-2002-303,* 2002