Towards Service Orientation in Pervasive Computing Systems

Umesh Bellur
KReSIT, IIT Bombay
Mumbai, India
umesh@it.iitb.ac.in

N.C. Narendra
IBM Software Labs India,
Airport Road, Bangalore, India
narendra@in.ibm.com

Abstract

The emergence of the service-oriented computing paradigm has opened the possibility of using dynamic binding of application requirements to the resources needed to fulfill application tasks. Especially in pervasive computing that is characterized by disconnected operation and mobility, the process of using service specifications and dynamic binding becomes critical. In this paper, we summarize our ongoing work in the area of integrating service orientation into pervasive computing using the notion of specifying service requirements and using these specifications to bind to the available resources dynamically instead of hardwiring them statically. We term these specifications programmable requirements since they can be interpreted at run time to bind to a resource satisfying those specifications. Interestingly this approach also satisfies the basis for two key types of adaptation prevalent in pervasive computing systems – functional and architectural – as we show in this paper.

Keywords: Pervasive Computing, Service-Oriented Computing, Programmable Requirements, Adaptation, Ontologies

1. Introduction & Background

Pervasive computing [21] is an era of computing with the special distinguishing characteristic that users will no longer be tied to the desktop paradigm, and will therefore become increasingly mobile. Hence 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.

The newly emerging service-oriented computing [18] paradigm stresses the need for computing systems to develop the flexibility for dynamically binding applications to the resources needed to execute the applications, at run-time. This is an enhancement over traditional computing systems, where this binding is done statically at design time and any run-time changes can only be implemented with extensive re-programming (although CORBA does provide some level of dynamic binding, it suffers from limitations such as requiring exact naming matches for interfaces, which violate the loose coupling requirement necessary for dynamic binding as per service-oriented computing [18] principles). In the case of pervasive computing systems especially, where rapid changes in usage patterns are the norm, dynamic binding to resources is crucial. This is because resources in pervasive computing can exist anywhere, hence they may come online at any time in an environment where they will have to be discovered and bound to dynamically.

In addition, traditional computer system modeling has always assumed informal or semi-formal specifications of requirements, in natural languages which are not understandable by a computing device. The increasing complexity and distributed nature of today’s computing systems (and pervasive systems are no exception to this) is prompting the investigation of formal (or at least, machine-processable) requirements.

In this paper we integrate service orientation and machine-processable requirements into pervasive computing systems and summarize our ongoing work in what we call programmable requirements for pervasive computing systems. By adopting the activity-based computing approach [6, 20], users’ functional requirements are expressed in a process-like manner via usage scenarios. These scenarios are then automatically composed together into a workflow. For this, we extend our earlier work on workflow process creation from scenarios [13], since that work presents a simple and automatable technique for automatically deriving workflow specifications from usage scenarios. For programming non-functional requirements such as architectural, Quality of Service (QoS), performance, etc., we leverage from our earlier work on context ontology-based web service reconciliation [9] by presenting a 3-layer approach for representing non-functional requirements vis-à-vis resource capabilities using context ontologies. We emphasize, however, that our work is not the same as executable specifications using formal methods [22], since formal verification of the semantics of the generated workflows is outside the scope of this paper. Please also note that while UML allows formality in the expression of system design where one specifies interfaces (method signatures) and relationships between these interfaces, our focus is more on the specification of
the functional and non-functional requirements of the workflows. This is akin to the specification of the body of the UML method rather than its interface.

Since adaptation in pervasive computing systems usage is a common practice, we also show how our approach can be extended to model adaptation by leveraging our earlier work on 3-tier adaptive workflow modeling [12]. In particular, we model two key types of adaptation that are prevalent in pervasive computing systems [17]: functional adaptation arising from changes in user requirements, and architectural adaptation arising from changes in non-functional requirements. In particular, we show how architectural adaptation can be implemented via dynamic binding of workflow tasks to resources at run-time as per the service-oriented computing paradigm [18].

Our paper is organized as follows. In the next Section, we describe our programmable requirements approach, which consists of the following: converting users’ functional requirements into workflows; dynamic binding of workflow tasks to resources; and modeling functional and architectural adaptation. The current status of our work, and future work, are discussed in Section 3.

Before we begin to describe our approach, we need to qualify our meaning of “pervasive”. Since this is still ongoing work, we have so far modeled pervasive computing devices as primarily mobile artifacts through which the users interact with each other and with their environment. However, as we will see in Section 2, even such a highly limited interpretation of pervasive computing raises significant research issues. More detailed pervasive computing issues such as smart spaces, energy-aware systems, and network reliability (see [16] for an excellent taxonomy of pervasive computing issues and challenges) will be taken up for future work.

2. Our Approach

We describe our approach below. In what follows, we simply refer to the pervasive computing system as “the system”, in order to distinguish it from the “hospital computer system” which is introduced later below as a running scenario.

2.1 Converting Functional Requirements into Workflows

In order to model the task-oriented nature of the system, we capture the users’ functional requirements in the form of usage scenarios, in particular, Message Sequence Charts (also known as MSCs) [1]. Example scenarios are depicted in Fig. 1.

Imagine a hospital that runs a series of tests on a patient. Each test procedure can be specified as a usage scenario, as per Fig.1. The usage scenario in Fig. 1 describes three tests ordered by a doctor on a patient – blood, X-ray, MRI scan. Since the blood test (Scenario #1) is carried out by a skilled pathologist, the doctor

Fig. 1: Usage Scenarios – (i), (ii) and (iii) are labeled for illustration purposes; for explanation, see text
notifies the pathologist and the nurse separately, who will then coordinate with each other to carry out the test. The case for Scenario #3, i.e., MRI Scan, is similar. In the case of the X-Ray (Scenario #2), however, the doctor decides to delegate the task to the Nurse #2 directly, who will then ensure that the X-Ray test is carried out. In Scenario #4, the doctor confers with the Specialist in order to interpret the results of the MRI scan, after which he will prepare his diagnosis and enter it into the hospital’s computer system.

Pervasive computing devices such as mobile phones, PDAs, pagers, etc., are needed here, since the doctor, nurses, pathologist, and specialist could be constantly on the move when they need to be contacted.

For the sake of clarity, scenarios can be refined into sub-scenarios in a manner similar to that used to decompose MSCs [MSC]. The sub-scenario represents a decomposition of the parent scenario without affecting the parent scenario’s observable behavior.

One point to note in Fig. 1, is the absence of branching and iteration. However, MSCs can also represent branching and iteration. For example, since MRI scanning is a highly delicate procedure, even slight patient movement during the scanning can result in faulty results, thereby causing the entire procedure to be redone.

Composing these usage scenarios into a workflow is quite straightforward, and can be implemented using techniques such as the one presented in [13]. The workflow for the usage scenarios shown in Fig. 1, is depicted in Fig. 2. Workflows can be programmatically represented using workflow description languages such as BPEL (http://www-128.ibm.com/developerworks/library/ws-bpel/), with “handoffs” between tasks represented as interfaces of their respective services using a representation such as WSDL (http://www.w3.org/TR/wsdl).

Since each task is performed by a role, the task is then mapped into a service. A service is an abstract description of a task that the role is capable of performing, and it satisfies the functional semantics of the task (i.e., the goals that the task is supposed to meet). For example, one of the services that Nurse #2 can perform, is getting an X-Ray done, which is depicted as a task in Scenario #2.

Our overall conceptual model is as depicted in Fig. 3 (we describe Fig. 3 in more detail in Section 2.2, when dynamic binding of services to resources is introduced).

### 2.2 Dynamically Binding Services to Resources

Each service, as explained above, is merely an abstract description of a task, and should now be dynamically bound [18] to actual resources before it can be implemented. This binding is based on the non-functional requirements specified by the users [4]. For example, a doctor would specify that he would like to receive messages on his PDA in a point-to-point fashion, i.e., directly from particular senders rather than via a messaging middleware. On the contrary, while sending messages, the doctor may want to route them via the messaging middleware. One way of capturing and representing these requirements is in the form of policies as described in [11].

Resources in our pervasive computing context could mean human resources such as doctor, nurses, etc., or

---

**Fig. 2: Workflow Definition**
physical resources such as the computing devices used by the doctors and nurses, hospital computer system, etc. An example of binding human resources, would be to bind the role of Nurse #2 to a particular nurse in the hospital capable of providing the services of Nurse #2. An example of binding physical resources, would be to allocate a doctor’s device to a particular PDA that the doctor owns. Hence the binding of services to roles, and dynamic binding of services to resources (human and physical), is as depicted in our conceptual model shown in Fig. 3. That is, a service is provided by a role, and the service would also require the use of physical resources as well. This service is mapped to a task in the workflow. The linkage between roles and human resources has been explicitly provided in our model, to allow for role-based assignment (for example, only certain trained people can operate MRI scan machines, hence those human resources are assigned the MRI Technician role).

To reiterate, 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.

Binding of services to resources is primarily dependent on the context in which the binding occurs. Context is any information relevant to the interactions between a user and his environment [7]. For example, the message content sent to the doctor would need to be personalized depending on the computing device that the doctor would be carrying at that point in time, and also the doctor’s location. For example, if the doctor is carrying a mobile phone and is out of the hospital, he would like to only receive text messages reporting status, and telling him where in the hospital computer system he can obtain more information about the patient. On the other hand, if the doctor is inside the hospital, he should be able to use the Infrared feature on his mobile phone to download more information from a desktop computer located at the nurse’s station. In case the doctor is carrying a PDA and is out of the hospital, he would like to receive more detailed information, including digitized X-Ray and MRI scan images. The service binding is therefore dynamic, and would depend on the location and device of the doctor at that point in time; in fact, this information about the doctor usually cannot be provided in advance since it could change from day to day, and hence it is not possible to specify this at design time.

Hence a description of the properties of the context, and the capabilities of the resources operating within that context, is crucial. In other words, a context ontology is needed.

An ontology typically consists of the following: concepts depicting certain attributes, relations that describe the relationships among the attributes, and axioms depicting certain invariant properties obeyed by the concepts and relations. A context ontology is therefore a description of the context parameters, together with the relations among them. A description of using context ontologies in Web service environments in described in [9].

A context ontology consists of three sets of parameters. The first set, which we call environmental context ontology or simply E-context, is a description of the environmental parameters within which the overall system is to operate, and is to be modeled and maintained globally. The second set, which we call service context ontology or simply S-context, is a description of the non-functional requirements for each service. These two sets are typically specified by the users. The third set, which we call resource context ontology or simply R-context, is a description of the actual physical parameters of each resource.

Binding services to resources therefore requires first matching the non-functional requirements of a service against the available resources, while ensuring adherence to the overall environmental parameters. This can be done by checking the S-context for each service against the appropriate R-context, and then ensuring that the S- and R-contexts match to the E-context. This is similar to the

Examples of **E-context** would be: participant role (doctor, nurse, pathologist), names of devices used by the participants (mobile, PDA, pager, PocketPC), participant locations (nurse’s station, operating room, consulting room, outside hospital). Examples of **S-context** would be: architectural requirements (point-to-point messaging, publish-subscribe messaging), performance requirements (maximum time to be taken for each test, maximum time for reporting test results). Examples of **R-context** would be: screen size of a device, memory capacity, processing speed, facilities offered (paging, SMS, email).

As an example of dynamic service binding to resources, let us consider the following for the activity of X-Ray Technician conducting X-Ray & reporting to the doctor:

- **R-Context**: PDA carried by doctor
- **S-Context**: point-to-point message from X-Ray technician to doctor
- **E-Context**: doctor inside/outside operating room

Hence if the system determines that the doctor is in an operating room, it would merely send an informational message to the doctor about the X-Ray, with a reference to the location in the hospital computer system where the doctor can obtain the X-Ray image. Otherwise, depending on the type of PDA that the doctor is carrying, the system can directly send the digitized X-Ray image to the doctor.

### 2.3 Representing & Modeling Adaptation

Adaptation in pervasive computing systems is expected to happen constantly [17], due to two major factors:

- Changes in functional requirements: we call this *functional adaptation*, and this arises when users’ functional requirements – expressed as usage scenarios – change
- Changes in non-functional requirements: we call this *architectural adaptation*, and this arises when users’ non-functional requirements (QoS, performance-related, etc.) and/or resource availability, changes

In [12], we had defined a 3-tier model for representing adaptation in workflows. We are developing a similar but more lightweight 3-tier model for adaptation in pervasive computing systems, comprising **environment** layer, **service** layer and **resource** layer, mapping directly to the **E-**, **S-**, and **R-contexts**, respectively, and is depicted in Fig. 4.

Our adaptation model closely resembles the 3-tier adaptation model described in [8]. However, to the best of our knowledge, the model in [8], just like the rest of Project Aura that it is derived from, is suitable only for single-user workflows, whereas our approach in this paper makes no assumptions on the number of users jointly executing a workflow. It is also to be noted that our 3-tier adaptive workflow model [12] was already presented in 2000, and hence predates the model in [8].

Functional adaptation will be represented as modified MSCs, which would automatically translate to modified workflows, thereby directly affecting the **E-** and **S-contexts** (see Environment & Service layers in Fig. 4).

![System Architecture](Fig. 4: System Architecture)

This may affect resource binding indirectly, since resources available earlier at particular points in time
during the previous workflow, may not be available now. This would, in turn, trigger changes in non-functional requirements in order to find replacement resources (also referred to as “task approximation” in [20]).

Architectural adaptation would arise due to changes in users’ non-functional requirements and/or resource availability, and would directly affect the R- and S-contexts (see Resource and Service layers in Fig. 4). If alternate resources can be found without affecting the workflow sequence execution, then the S-context would not be affected. Otherwise, this could trigger a functional requirements change, since the workflow would need to be modified in order to ensure appropriate resource availability.

Even in our rather simple running example, several functional adaptation possibilities exist:

- The order of tests could be changed, e.g., X-Ray test would need to be conducted before blood test, due to the lack of availability of a pathologist
- Instead of reporting to the doctor after every test, the hospital’s computer system could first be directly updated and the doctor could be notified after all tests are completed
- The process could be further optimized by the appointment of a Head Nurse, who would report to the doctor on all the tests and coordinate with the nurses

Likewise, several architectural adaptation possibilities also exist in our running example:

- Some users’ computing devices could be unreachable (i.e., disconnected), leading to a switch from point-to-point communication to publish/subscribe communication via a messaging middleware. For example, in the service binding example of Section 2.2, the lack of availability of the doctor’s device at that point in time, could necessitate this type of a change.
- Alternatively, if publish/subscribe was the mechanism originally chosen by the users and if the message traffic on the middleware exceeds a performance threshold, it may be appropriate to dynamically switch to point-to-point communication for certain classes of messages (such as the doctor’s interaction with the specialist)

For functional adaptation, we are extending our earlier work on adaptive workflow [12], while incorporating context ontologies [9]. In particular, we are investigating how the adaptive workflow techniques in [12] can be “scaled down” for the more lightweight adaptation requirements here, so that mid-flight workflow modifications can be implemented with least disruption to ongoing tasks.

For architectural adaptation, we are investigating middleware reconfiguration techniques such as [3]. Of particular interest is enhancing the current work on architectural reflection via the use of “overlay networks” as component frameworks in the system, so that reconfigurability of the core messaging and networking functionality itself can be achieved. For implementing architectural adaptation, we are also investigating the integration of resource contracts idea from [14] with the policy matching idea from [11].

3. Current Status & Future Work

We have presented an approach that we are developing based on programmable requirements for integrating service orientation and machine-processable requirements specification into pervasive computing systems. We have also shown how our approach can also be used for modeling functional and architectural adaptation in usage patterns in such systems. So far, we have modeled the workflow generation approach from functional requirements, and a context ontology-based approach for associating non-functional requirements; we have also introduced mechanisms for implementing mid-stream functional and non-functional (architectural) adaptation.

Please note that this is still ongoing work, hence future work will focus on completing the development of our approach, and implementing and testing it. We will also be enhancing our work to incorporate the several pervasive computing challenges raised in [16], particularly smart spaces.

Recent work on protocol design [10] aims to develop an algebra for modeling and composing protocols, using the Process Handbook approach [2] as a basis. Interestingly, our earlier work on workflow creation [13] on which this paper is based, also uses the Process Handbook approach. Hence investigation of the extent to which the work in [10] can help us in modeling workflows using protocols, would be fruitful future work.

As part of adaptation modeling, we will also be investigating the use of so-called “proactive guidance” techniques [5] for anticipating changes in advance before they happen, and implementing functional and architectural adaptation before the users begin to feel the need for change. This would require leveraging techniques from Autonomic Computing [19].

Acknowledgements

The authors would like to thank Manu Kuchhal, Zakaria Maamar, Dipanjan Chakraborty and Soraya Kouadri Mostefaoui for their feedback. The second author would like to thank his manager, K. Muralidharan, for supporting his work.

Other company (i.e., non-IBM) product and service
names, may be trademarks or service marks of others.

References


