A Practical Approach to Partition Applications in Pervasive Computing Environments

Nevin Vunka Junguma*, Nawaz Mohamudallya and Nimal Nissankeb

aSchool of Innovative Technologies and Engineering, University of Technology Mauritius (UTM), La Tour Koenig, Mauritius
bSchool of Computing, Information Systems and Mathematics, London South Bank University, London, UK

Abstract

Application partitioning to make optimum use of available computational devices in a pervasive computing environment is being researched heavily both in military domain as well for the civil society. This work is a sequel of our previous theoretical effort in developing a framework for seamless application partitioning and distribution at runtime in pervasive computing environments. In this work, we described four important requirements that any partitioning framework must adhere to: partitioning at runtime, network fault tolerant, retractable capability and preservation of application state. We proposed a framework that encompassed all the mentioned requirements and its prototype implementation. We then designed a very simple laboratory music player that run on top of the framework.

Keywords: software partitioning, framework, distributed systems, pervasive computing system, smart home

1. Introduction

In 1991, Mark Weiser1, the former Scientific Director of the Xerox Palo Alto Research Center, shaped the vision of Pervasive/Ubiquitous Computing Environment as an omnipresent infrastructure for information and communication technologies (ICTs). In such an environment, we may have one or more context-aware systems ready to assist the mobile users in their respective tasks. Following his work, in subsequent years we have witnessed a huge interest of

* Corresponding author
E-mail address: nevin.vunkajungum@gmail.com
the scientific community in the areas of pervasive computing, context-awareness, autonomous systems, ambient intelligence, context-aware middleware and so forth.

Applications in pervasive environments need to be context-aware ones. The words context-awareness was coined by Schilit et al.\textsuperscript{3} to illustrate a new class of computer software application that exploits the changing environment variables of a mobile computer user\textsuperscript{12}. These computer applications sense the environment in which they are running (their context) and adapt their behavior accordingly to best assist the user in his/her task\textsuperscript{12}, and any changes which may occur to that context. Typically, the contexts of interest might include the location of the user, surrounding environment and nearby computational resources\textsuperscript{9}, though bearing in mind that the amount of physical and logical environmental variables relevant for different applications are possibly infinite. Next generation context-aware applications must be able to make maximum and intelligent use of the digital nodes in the user’s surrounding in a pervasive environment. In order to make use of the digital nodes embedded in such environment, context-aware applications need to extend traditional applications by incorporating the partitioning and retraction capabilities. By this we mean that an application would be able to decompose itself into multiple codes fragment and be hosted on multiple nodes and recompose or retract back once the user’s task is done. Both operations should involve minimal user intervention or if possible no user intervention at all. Establishing a framework that would ensure that the development of context-aware applications would include the partitioning and retraction capabilities would yield novel and interesting applications for the emerging pervasive environments. An interesting application of such novel software technology is described by a group of researchers\textsuperscript{9} at the University of Brussel in the mid of 2008. They developed an application for communication in a pervasive environment called ContextCom. The latter has the capabilities of partitioning and retraction. For more detailed information about it, refer to the authors’ work\textsuperscript{9, 10}. The authors also mentioned that this specific field is a newly growing one with lots of research that needs to be conducted. Hence this basically confirmed the need for investigation in the area of partitioning and retraction for context-aware applications.

Existing attempts to software partitioning focused mainly on issues such as processor load and inter-process communication (which are important and we are not arguing against), but they are more concerned with software partitioning for hardware/processors; whereas in pervasive computing environments, software partitioning is more concerned for services.

Due to high mobility of users in a pervasive environment, there are very often network disconnections which existing approaches fail to address as they were designed for traditional wired distributed networks and assume very high networks connectivity which does not hold in pervasive environments.

Partitioned application when retracting back should have its application state preserve at any given time. The framework will have to manage all complexities in ensuring the state of application is available in the event of manual retraction or network failures.

There is the issue of partitioning at design time and partitioning at runtime that we believe is important to address in a pervasive environment. For example, most of the existing partitioning approaches for traditional distributed computing environments were already aware which part of the partitioned application is going to be moved/hosted where. This decision is taken by the application developer at design time and actually neglecting about how the user intend to use the application. Since in a real-life pervasive environment it is difficult to predict where the users would like to move the partitioned software as new devices/resources are discovered and others loss at different time intervals, partitioning at runtime is more relevant to partitioning at design time so that the applications can easily adapt to any configuration change to better assist the users in the services they are using.

The rest of the paper is organized as follows. In Section 2, we carry out a brief study of existing approaches to software partitioning and retraction. Next in Section 3, we identify a set of important requirements that should be satisfied for proper partitioning and retraction of applications in pervasive computing environment: (a) partitioning at runtime, (b) network fault tolerant, (c) retractable capability and (d) preservation of application state. The requirements are extracted from a typical real-life motivating scenario of a music video player running in a smart home\textsuperscript{2} and while studying existing works in Section 2. Then we present the design and prototype implementation of a framework. In Section 4, we design a video player application running on top of the framework. And we finally conclude in Section 5 and briefly describe immediate future work that need to be addressed.
2. Related Works

In JavaParty\(^5\) and Doorastha\(^13\) the authors proposed language abstractions on top of Java RMI (Remote Method Invocation) to ease the development of multi-threaded Java software in a distributed environment. In this approach, it is important to note that partitioning is decided at design time. A class named "remote" is made available and has the job of recognizing classes that should be distributed while concealing all communication details behind. One very important issue with JavaParty is that once the specification of the partitioning scheme is set, it cannot be dynamically modified at runtime. This characteristic of JavaParty makes it hard to be considered for partitioning context-aware applications where changes in context variables are often and these could possibly influence the decision about which partitioning method to use. Application developers have to add annotations which specify different passing modes for java objects. Three different annotations to tag a class declaration are provided in Doorastha. They are globalizable (local objects are transformed to remote objects), copyable (objects are transferred by-copy) or migratable (objects are migrated to a node).

In Addistant\(^8\), the system takes Java byte code as input to transform it such that it can be executed in multiple JVM. A strong limitation of Addistant is that application developers must specify where to allocate the instances of each class among multiple hosts in a policy file. Using this approach, users moving in new environments where new nodes/devices are discovered will seriously cause confusion about which class should be hosted in which node. Furthermore, the partitioning operation is not dynamically modifiable at runtime; the translated code is moved to a remote JVM after the Addistant class loader started at each host.

Using AdJava\(^4\), an application developer has to specify in the application source code the distributable objects (the classes by using the specific keyword Distribute) and provide a list of remote locations (the nodes/devices) on which the agents are running. Such agents will then decide at runtime the destination nodes based on migration and load balancing techniques. Load balancing is achieved based on the knowledge of existing nodes/machines and their continuous availability is assumed which is different in a mobile pervasive environment setting whereby nodes are unknown at design time and availability is not guaranteed. Application developers need to explicitly specify at design time on which nodes distributed objects can be migrated. In case of network or device failure, any distributed object fails to retract back to the original source device/node. However, if we assume complete and absolute network connections and no device failure, retraction is 100% guaranteed. Developers building distributed applications for wired networks can positively consider AdJava as a first choice.

Hydra\(^6\) is a very good step and contribution towards partitioning software for pervasive environments. The framework enables partitioning of an application at runtime which is relevant for applications to be deployed in pervasive environments. The framework has no mechanism to handle network failures though. So in the event an application has been partitioned, migrated to a node and there is a network disconnection, the application simply crashes.

The authors in papers\(^7, 11\) designed a system called J-Orchestra that has objective to transform a centralized Java program into a distributed one. Java byte codes are taken as input and method invocations and (direct) object references are automatically replaced with remote method calls and indirect remote object references (i.e. proxy objects to remote objects) respectively for the classes that the application developer has specified to be mobile. Programmers do not have to modify by hand the application source code while using J-Orchestra since the latter was designed to provide automatic partitioning. Application developers have to specify which classes to partition and the specific nodes/devices where the application will be hosted. Also this approach does not provide any means for retracting back the partitioned codes.

3. Proposed Framework Design and Implementation

As mentioned in Section 1, we identified some important requirements based on the scenario description of a music video player running in a smart home we covered in a previous paper\(^2\) and while studying existing works. The five known main properties for pervasive/ubiquitous computing are autonomous, distributed, iHCl, context-aware, and intelligent. However for a partitioning and retraction application system as described in this paper the following milestones are mandatory.
(a) Partitioning at runtime
In pervasive environments mobility is a core ingredient. Some devices in the environment will keep on moving thus becoming available and unavailable, which is hard for the application developers to forecast. Therefore, any framework must ensure that applications built on top of them are partitioned at runtime based on the different configurations/network topology and settings instead of restricting application developers to specific devices and/or configurations which will definitely violate end-users’ freedom to physically move in the environment and make use of different devices around them. This will involve understanding how users intend to use relevant devices discovered within the environment while naturally balancing processor load and inter-process communication.

(b) Network fault tolerant
Due to mobility of users, network disconnection is no longer an assumption but a core fundamental principle that needs to be addressed as we pointed out earlier. The framework needs to be able to handle appropriately a high frequency of network failures at different time intervals and for any duration period. In the event of a network failure the application developer should be able to enable the application to resume work from the lost device back to the original device as if nothing happened.

(c) Retractable capability
Partitioning an application and move it over to a device/node is the first step. The framework must also provide application developers the ability to retract the partition codes back to the source device/node. This will ensure that the application gets back to its original form and will work as usual. Retraction can be triggered in several ways, for example, end of task, network disconnection, user decision and change of environment.

(d) Preserving application state
While a partitioned application is running on multiple devices, if at any point in time the user decides to retract back all partitioned parts, the state of the application must be preserved. For example, a movie resume on the current device from where it stopped on the previous device.

In general, when an application C is partitioned based on some constraints, say in n parts, we have:
\[ \text{Partition (app C)} = \{ C_1, C_2, \ldots, C_n \} \]
where \( n \) is the number of parts app C can be partitioned and \( n \geq 2 \).
For example, app C is to be partitioned in 5 parts, we then have:
\[ \text{Partition (app C)} = \{ C_1, C_2, C_3, C_4, C_5 \} \]
Henceforth, we will refer to a partition block of codes as \( C_1, C_2, \ldots, C_n \), if for example the application name is C.

The framework has been implemented as a 2-tier architecture. On both the client and server sides, the BubbleCodes framework need to be running on top of the Java Virtual Machine (JVM) prior to launching an application and stay on listening mode. In a smart home, there will be different versions and types of Java virtual machines embedded in for example, washing machine, toaster, refrigerator, laptops, desktops, smartphones, and so on. Figure 1 below shows an overview of the BubbleCodes framework. An application is partitioned and a copy of each partitioned block of codes referred as a Bubble is moved to available nodes in the environment. Moving a copy of the Bubbles to the nodes ensure network resilience and full retraction. Whenever a node hosting a Bubble is out of network or simply fail due to any hardware problem (e.g., low power), we always have a copy of the Bubble on the source device.

As an overview, when an application is launched, in case of available devices, the application has been designed in such a way that the user is given the option to partition the application thus moving parts of it to available devices within the environment. As illustrated in Figure 1 below, the partitioned codes, that is, the Bubbles (red, blue, yellow and green circles) consists only of the application logic. The list of available nodes/devices are described as Participating Nodes. An instance of the BubbleCode server-side already started in all nodes participating in the environment. Each Bubble (application logic), based on certain preferences of the user, are moved to the nodes and will be running on top of the BubbleCodes Framework. Thus the complexity of partitioning, networking, exchanging messages between Bubbles and so on, are abstracted to application developers leaving them to concentrate more on the application.
Figure 2 below shows the architectural components of the BubbleCodes Framework. Before starting an application, the framework need to start on all nodes. The BubbleCodes instance/framework on all nodes will first attempt to detect available devices/nodes in their surrounding environment; then check for permission and get the communication time from one node to another, local memory available and local processor load. Thus a list of potential participating nodes is created. The previous processes are performed by the Detect participating nodes module. Once the nodes to host the Bubble(s) have been selected the BubbleMap module creates a list, called a BubbleMap, that list all Bubbles and their corresponding nodes and all available nodes and the BubbleMap is sent to all nodes running the BubbleCodes framework. This has two important advantages: (1) it allows an application initial launch device to know exactly where a Bubble is located at any point in time and (2) in case of node unavailability due to network disconnection, the initial device can choose another node to send a copy of the Bubble. Whenever there is an update in the BubbleMap, a copy of the latter is sent to all Bubbles concerned. The module will also have to continuously monitor the availability of the nodes hosting the Bubbles and take the required actions (for example, re-assign a Bubble to another node) in case of lost connectivity or end of task.
If the user has already set certain preferences concerning available nodes, for example, the user might have already set a HiFi System to output the sound. In this context, the user will not be proposed with a choice of potential nodes. The application will seamlessly partitioned itself and move the concerned Bubble(s) to the HiFi System. In case of no preferences set, the user can choose among which device available to partition the application.

The Conn. To Server Status module of both the initial node/device and the remote nodes will communicate to check for connection, Task completion module to notify end of a task and to destroy a Bubble to free space. Application State module will keep on syncing any relevant data to the initial node that would help the application to resume normally in case of network failure. Figure 3 below shows how the BubbleCodes framework on the client side send a Bubble to a remote participating node using the sendBubble(InetAddress add, int port, Bubble b, FileProgressListener li) method we implemented in the Partitioning Decision module. We also implemented the executeBubble(File dir, String name) method in the Partitioning Decision module that has the task of executing the Bubble(s) once they are moved the remote nodes.

4. A Laboratory Music Player for Pervasive Environment

We set up our environment that has approximately an area of 51m² and consist of three physical rooms and a corridor that connects them. A laptop is placed in each one of them representing a HiFi system (living room), a TV set (dining room) and a laptop itself (bedroom) as shown in Figure 4 below. The laptops we used are two HP Intel Core 2 Duo, 2.2 GHz, 4 GB Memory, 64-bit Operating System running Windows 7 and Java 7 Update 45 and one Samsung Intel Core i5, 2.5 GHz, 6 GB Memory, 64-bit Operating System, x64-based processor running Windows 8 and Java 7 Update 40.
We build a very simple music player to play MP3 files as shown in Figure 5 below. We first launch BubbleCodes on the server-side and then launch the application on the client-side. Under the *Files* area, all classes that build up the music player application is listed. In *Active Network* participating nodes/devices IP addresses are displayed. All songs are stored locally in a folder and are listed in *PlayList*. Whenever we attempt to transfer classes from the initial device to the targeted one, their status are displayed under *File Transfer Status*. For this work, the prototype was designed in such a way to give us the flexibility to manually transfer the classes. However, we intend to automate this process. The buttons *Send File* and *Retract* are used to transfer and retract back the classes respectively.

As we walk into the bedroom, we were able to move the concerned classes (*Player Setting*, *Playlist*, *Music Player*, and so on) to our bedroom's laptop. We then move to the living room, retract the classes in the bedroom's laptop using the retraction option and move again the concern classes to output the sound to our living room's laptop.
5. Conclusion and Future Work

As mentioned earlier, this contribution is a sequel of our previous theoretical effort in developing a framework for seamless application partitioning and distribution at runtime in pervasive computing environments. In this work, we carried out a brief study of existing approaches to software partitioning and retraction. We identified a set of important requirements that should be satisfied for proper partitioning and retraction of applications in pervasive computing environment: partitioning at runtime, network fault tolerant, retractable capability and preservation of application state. The requirements are extracted from a typical real-life motivating scenario of a music video player running in a smart home and while studying the existing works. We then proposed the design of a framework and implemented a prototype along with the implementation of a music player application running on top of the former. As future work, we intend to perform the evaluation of the proposed framework with respect to multiple applications running in the environment. As such, bandwidth cost, CPU usage and memory usage will be monitored and analysed.

Acknowledgement

This work has been sponsored by the Tertiary Education Commission (TEC) Mauritius, Website: http://www.tec.mu

References