Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 292 (2013) 39–56

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

MultiS: A Context-Server for Pervasive Computing

Felipe Weber Fehlberg[1] Carlos O. Rolim[2]
Valderi R. Q. Leithardt[3] Claudio F. R. Geyer[4]
Luciano C. Silva[5]

Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil

Anubis G. M. Rossetto[6]

Federal Institute of Education, Science and Technology Sul-rio-grandense – Campus Passo Fundo
Passo Fundo, Brazil

Abstract

Context-aware applications are capable of recognizing environmental changes and adapting their behavior to the new context. This process can be divided into three stages: monitoring, context recognition and adaptation. On the monitoring layer, raw information about the environment is collected from sensors. The context recognition layer processes the data acquired from the context and transforms it into information which can be useful for the adaptation process. With this information, the adaptation system can determine what behavior is correct for the application in each different context. This paper proposes a context server called MultiS, which has the goal of solving the problems arising from the context recognition layer, and which includes the following advantages: a) the production of new context data based on the information of several sensors and an ability to react to changes in the environment; b) definition of a composed language for the context data called CD-XML; c) support for mobility.

Keywords: Pervasive Computing, Context-aware Computing, Adaptation, Middleware.

[1] Email: ffwf@inf.ufrgs.br
[2] Email: carlos.oberdan@inf.ufrgs.br
[3] Email: valderi.quietinho@inf.ufrgs.br
[4] Email: geyer@inf.ufrgs.br
[5] Email: luccg@inf.ufrgs.br
[6] Email: anubis.rossetto@passofundo.ifsul.edu.br

doi:10.1016/j.entcs.2013.02.004

1 Introduction

Ubiquitous computing is a vision about the future that was initially raised by Mark Weiser [17]. It is based on the belief that computers will spread throughout the environment. Their presence will be so common that it will not be noticed by the users. Some years later, IBM adopted the concept of Pervasive Computing [14], which is closer to the existing technology. Both wired and wireless devices can be used to provide a service network. The user data, programs and processing power will be available through the network. The service will be provided to the user by means of the resources available in the environment. This paper treats Ubiquitous Computing and Pervasive Computing as synonyms.

The current technology, however, has not yet reached the required level to become ubiquitous. In the view of the ISAM group, this future can be achieved by integrating three different concepts: Grid Computing, Mobile Computing and Context-Aware Computing [16]. Context-Aware Computing will be responsible for detecting the resources that are available in the environment, and enabling it to have access to the applications. One goal of context-aware systems is to acquire and utilize information in the context of a device so that it can provide services that are suited to the needs of particular people, places, times, events, etc. [3][5]. The ISAM (Infrastructure Support for Mobile Applications) project seeks to build a required support infrastructure for the execution of pervasive context-aware applications.

The ISAM authors propose an infrastructure where the computational environment is divided into cells. Each cell is composed of wired and wireless nodes. The cells are able to communicate with each other by using Peer-to-Peer protocol [15]. The applications are programmed with ISAM adapt abstractions [2], a programming language built for pervasive applications, together with EXEHDA (Environment Execution for High Distributed Applications) [16], which is the ISAM execution environment. This work forms a part of the ISAM initiative and seeks to propose a context-recognition engine for pervasive applications.

At the beginning of 2005, scientists from The British Computer Society published a list of what they believe will be the seven greatest challenges to computer science in the next twenty years. One of those challenges is how to build scalable ubiquitous systems. According to this list, Context-Aware Computing is one of the most important properties of Pervasive Computing.

Several works have been published with the aim of fostering Pervasive Computing. However, most of them focus on specific aspects of the environment, such as personalization or platform independence [12]. The integration of contextual information from different sources and the generation of contextual data using these different sources, is a problem yet to be solved. Other works which study context as a general area, such as context toolkit, fail to take account of important properties like mobility [9], which are very important for context- aware applications. Mobility is an essential feature of context-awareness, as it can enable the dynamic environment to set down special requirements for context-aware systems [10]. In contrast, the ISAM project regards context awareness as a premise. It is not only

present in applications, but also in the services that compose middleware.

In Context-Aware Computing, the application behavior can be adapted, depending on the user and context of the system. The system acquires this information through sensors, and usually provides data about a given aspect of the environment. This data must be interrelated so that it can produce a high level of information, and the system is easy to use by choosing the kind of behavior that is suitable for each situation.

Since several applications will be executing tasks in a given environment concurrently, there is a good chance that some of them will use the same high level of data (e.g. information about the location of a particular user). The support of middleware could save some system resources by sharing the common data between applications. However, although different applications may do this, each application may be sensitive about a given aspect of the environment and have its own particular needs. In view of this, the supporting middleware must enable each application to specify how it wants to receive its information. A context server approach can extend the middleware- based architecture by introducing an access managing remote component. This, aggregated sensor data is moved to the context server to facilitate simultaneous multiple access. As well as allowing the re-use of sensors, using a context server has the advantage of freeing clients from resource- intensive operations [3]. These are the factors that were the driving-force behind this work: the building of a supporting middleware that enables pervasive, context-aware applications to be executed.

This paper provides an outline of MultiS, a Multi-Sensor Context Server for Pervasive Computing. The text is structures as follows: Section 2 outlines the MultiS. Section 3 examines a case study, where a MultiS prototype was used to support the execution of an application, while Section 4 analyzes the test results acquired through the current implementation of MultiS and the execution of the developed application. Section 5 discusses related work in the field and compares MultiS to some of the existing projects in the same area. Section 6 discusses the conclusions and makes recommendations for future work.

2 Multis: Multi-Sensor Context Server

This section will give a detailed account of the MultiS architecture. MultiS is a context-server made for Pervasive Computing. To begin with, the context recognition process will be analyzed, followed by an examination of the MultiS project. The architectural elements of MultiS are described in Section B. In addition to the Context-Server, a programming language was created to compose sensors; this language is detailed in Section C. Section D shows how to deal with disconnected events. The sections from E to F studies the sub system responsible for processing and the contextual information, called Context-Compositor. Finally, Section G shows how the contextual information is shared across applications.

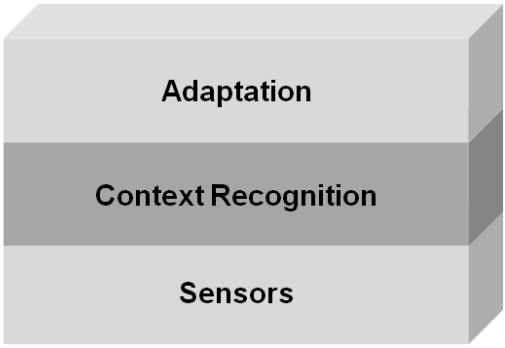

Fig. 1. Stages of the Context Adaptation Process

2.1 Context-Awareness Morphology

The context-awareness process can be understood as comprising three separate stages, as shown the Figure 1. Initially, it is necessary to acquire information from the environment, and this data is acquired through sensors. Each sensor gathers information about a given aspect of the context, though providing the system with a partial view of reality.

At the bottom of the Figure is the adaptation layer. Its responsibility is to transform the applications behavior when the environment is changed. The middle layer, (context-recognition), is the focal point of this work. It is there that the raw data from the sensors are processed and new contextual information is created. This information is used to carry out the adaptation process.

2.2 The MultiS Architecture Components

The MultiS architecture is composed of several different layers, each with a different role. Figure 2 shows the architecture components. For the sake of clarity, Figure 2 also displays the Monitoring and Adaptation systems, which are not part of MultiS but are required for its operation.

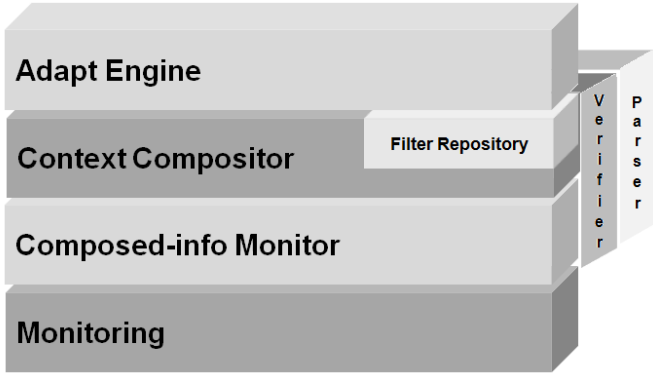

Fig. 2. MultiS Architecture

The layers of the proposed architecture are described below:

Parser: The Parser module is responsible for receiving the context definition from applications and forwarding them to the language interpreter. After the request, the information is then transmitted to the verifier module, which is responsible for allocating the required resources.

Verifier: The verifier module displays an API used by the Parser module. Its functions allow a dynamic loading of an execution code, allocation and free resources on several MultiS modules. While the resource allocation is taking place, the verifier attempts to find existing allocated resources that can be used to perform the required task. If necessary, the Verifiers also ask the Monitoring system for the sensors allocation. If the system is unable to allocate the required resources, an error is returned to the application.

Monitoring: Sensors are very important for context-aware computing. It is through them that systems know about the surrounding environment. There are different techniques to acquire information and there have been several studies about this. The differences in each approach make it hard to come up with a single generic solution. For this reason, most of the studies on context-recognition use an external monitoring system. The same strategy is employed by the MultiS project, which uses an external system for monitoring information. It is suggested that a monitoring system should be used that is capable of providing information about different sources and making them available through a common interface.

Context-Compositor: Context-Compositor is the core of the context-recognition system. It is responsible for the integration and processing of sensor data. This processing is carried out by using filters that are arranged as a context inference tree, the details of which are shown in Section F. If any situation is found that is programmed in the application request, the adaptations system is activated to change the application behavior.

Filter Repository: The filters are APIs used for processing contextual information. They are stored in a database called a filter repository. The filters are dynamically loaded by the Verifier module, and are used in Context-Compositor to produce contextual information. The repository also contains detailed information about each filter that allows programmers to query the database when a filter is needed.

Composed-Info Cache: The Composed-Info cache is an abstraction layer for accessing contextual data. Its role is to interact with the monitoring layer and with the Context-Compositor component to retrieve contextual information. When information is required, the filters query the composed-info cache, which then queries its internal cache or the monitoring system.

Adapt-Engine: The adapt engine is an external service and is responsible for changing the application behavior when the context-server detects a change in the environment.

2.3 A Language to Compose Contextual Information

According to the model used in this work, the application programmer must describe how its application relates to each aspect of the environment and in what

situations changes in the environment make it necessary to change the behavior of the application. This description is used by the context-server to process the information of the sensor and communicates the adaptation system whenever an adaptation is needed.

However, some technique is needed to enable the applications to describe their requirements to the context-server. Several other projects propose the use of languages that are especially designed for this task. The Aura/CIS [11] proposes a language close to SQL. It works very well for synchronous queries, but lacks expressivity for dealing with asynchronous queries, which are very important to react to environmental changes. The Solar [6] proposes the use of a XML-based language, but avoided discussing this in detail. No solution to this problem has been found in the bibliography. CD-XML is a proposal of a language of this kind that aims to solve the problem of communication applications requirements for context-recognition servers.

There are several challenges in developing such a proposal. New sensors can be added and new filters can be made can be made available during the execution of tasks by the context-server. The language used to describe the connection between the context-server and the application must be easily adaptable, and involve the addition of these new features. Support is still needed for the characteristics of pervasive computing, where mobile nodes may be unavailable at any time, and there is a risk that this may result in an undefined state of the sensors located in them. Moreover, the language should be simple and user-friendly for application programmers. The following is the CD-XML language, which has been created with the objective of meeting these requirements.

The CD-XML language: The CD-XML (Context-Definition XML) was created to enable the applications programmers to describe how the environmental information must be related and processed to produce useful information for the adaptation process. Each section of the language conveys information from a given number of sources through filters. The data used by the filters may be from sensors or information produced previously by a filter. The produced information may be used either by another filter or in the adaptation process. When external context information is changed, the adaptation system is activated. It receives data about the new state of the environment and may change the behavior of the application. Each language feature is described above:

<sensor> : Represents a sensor that is available in the monitoring system. The value of the tag <sensor> is that it is used to identify the sensor in the monitoring system.

<context> : Represents an item of information produced by the context-recognition system. The information held by this tag describes the data used to produce its value. Each tag of this type contains one filter and at least one information source. The filter is identified by the <filter> tag, while the information source may be a <sensor> or another <context> tag. In the case of the latter , the information is produced by the context-server.

<filter> : Represents a filter used to process contextual information. The prop-

erty name identifies the filter. Additionally, a filter may receive an additional parameter using the <parameters> tag. Filters may also receive the default' parameter, used to specify the filter behavior in disconnection situations.

<parameters> and <parameter> : Holds information about the filter parameters. The <parameters> tag may hold several <parameter> tags.

2.4 Treating Disconnection and Undetermined Values

One of the main characteristics of Mobile Computing is that mobile nodes are highly susceptible to disconnection. Since some sensors may be installed on mobile nodes, it may be necessary to specify the filter behavior when it cannot access the data of the sensor. When treating a disconnection situation, the behavior of each filter can be specified by the filter programmer. Nevertheless, there are some filters that are so general that they can be used in several different situations. Each circumstance may require a different approach. Moreover, it may be hard for the filter programmer to forecast each of the situations. The filter behavior can be changed by informing it how to behave when facing a disconnection case. To do this, the CD-XML language makes available the default property of the filter tag. When this option is specified, the filter returns its value whenever a disconnection situation occurs.

2.5 Detecting Sensor Changes

The changes in the environment are detected by the sensors. When a sensor detects the change, it triggers the context recognition process. This may result in an adaptation to the application behavior. Two different approaches can be used by MultiS to fire the context-recognition process.

It is possible to delegates this responsibility to the monitoring system, some of which are able to detect and warn when its sensors are changing. If the monitoring system is not able to do this, MultiS may periodically obtain the values of the sensors and compare them with the previous value. When a change is detected, a new calculation is started. This mechanism may also be used to overcome faults in the monitoring system. The use of both techniques enables the system to detect changes in the sensors even if the monitoring systems do not warn MultiS when a sensor is changed.

It is possible to configure how often this kind of verification will occur. If the time between the verifications is too short, it may waste system resource generating overhead. If the time is too long, the time to react to changes in the environment may be longer than what is acceptable to the user. It is suggested that this time should be set as near as possible to the maximum time that the user can wait for an environmental change to the systems reaction.

2.6 Context Inference Tree

Whenever a sensor change is detected, the context-recognition system is activated. We propose that the contextual information should be related by using a tree structure (already available on the EXEHDA context-server). These trees are allocated

when the application submits the CD-XML request to MultiS. Figure 4 displays a tree allocated for the CD -XML code shown in Figure 3.

```
<?xml version="1.0" enconding="UTF-8"?>
<context label="WhenInRoom"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    xsi:noNamespaceSchemaLocation="cd-xml.xsd">
    <filter name="alert">
        <parameters>
            <parameter>present</parameter>
        </parameters>
    </filter>
    <sensor>currentTime</sensor>
    <context label="isPersonInRoom">
        <filter name="FilterAtLeast" default="false">
            <parameters>
                <parameter>3</parameter>
            </parameters>
        </filter>
        <sensor>isComputerInUse</sensor>
        <sensor>isChairInUse</sensor>
        <sensor>isMovActive</sensor>
        <sensor>isLightOn</sensor>
        <sensor>isDoorLocked</sensor>
    </context>
</context>
```

Fig. 3. CD-XML example

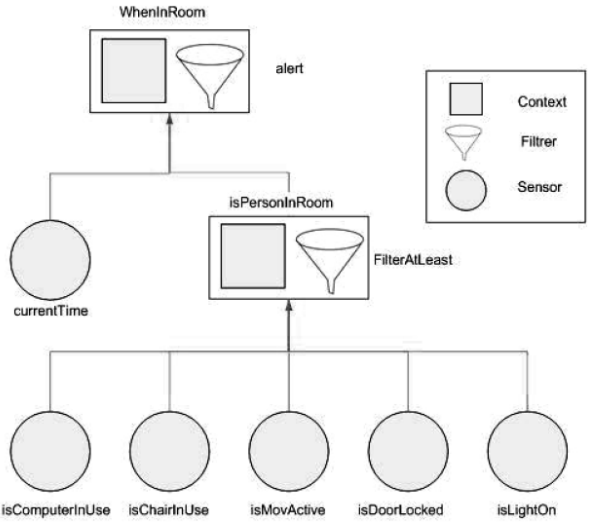

Fig. 4. Context Inference Tree

2.7 Processing Context Inference Trees

The <context> element represents the root node of a given tree or sub-tree. It always holds the filter and the information sources that may be a <sensor> node

or another <context> node. The value of the <context> node is given by the execution of its filter for the given sources of information.

For instance: to produce the value for the <context> isPersonInRoom, the system invokes the filter FilterAtLeast that receives the information from the following sensors: isComputerInUse, isChairInUse, isMovActive, isDoorLocked, isLightOn. Any change in the value of these sensors activates the context recognition system. It worth pointing out that any leaf node is a sensor, and thus the tree calculation algorithm is always bottom up.

If the change on the sensors value cascade to a change on a sub-tree root node, all the nodes that use this information will have to be re-evaluated. If the root node value changes, the EXEHDA AdaptEngine Service is activated. It should be stressed that not all sensor changes generate adaptation. It is possible that the change in the sensor value was not enough to change the root node; in this case, no action is taken.

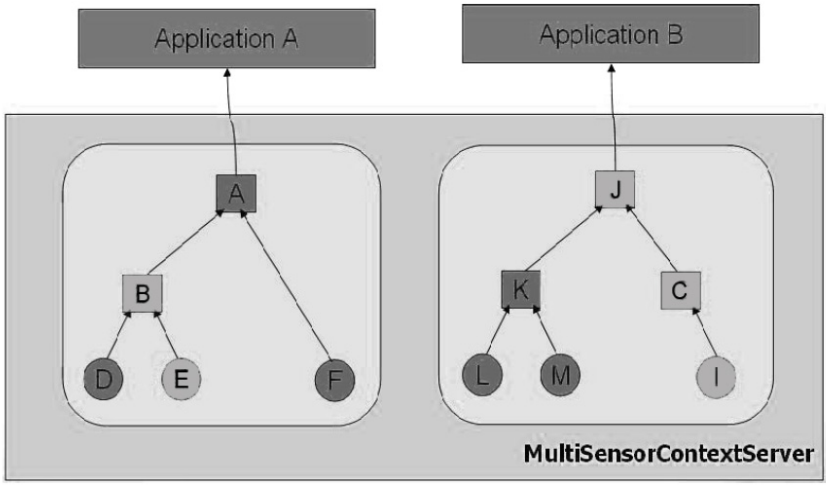

Fig. 5. Node Update Propagation After Environmental Change

Let us see how the algorithm works in a real scenario when the monitoring system has detected changes on two values of the sensor. The situation is shown in Figure 5. The first sensor is indicated by node E. When its value changes, the context-server re-evaluates all the sub-trees that use its value (i.e., node B). Despite this, the change at node E is not enough to change the value of the B node and thus, no action is taken. The second sensor, called I, causes the node C to be re-evaluated. The change at I makes the C value change too. The same happens to node J, which finally generates an event that transforms the application behavior (through an adaptation system). Note that no action was required in node K, because none of the sensors it used was changed.

2.8 Sharing Processed Information

The context server is able to support several applications subscribed concurrently. These applications are usually subscribed to a given server (i.e., cell) because it

contains information about the surrounding environment. For this reason, applications subscribed by a given server tend to use some common information and produce the same contextual data (such as a location of a given user, or a detected environmental activity).

Figure 6 represents the execution of three different context inference trees for different applications. It can be observed that, even if the applications X and Y have different requirements for the context-server, both share some common nodes, which are identified by the sub-tree with root node C.

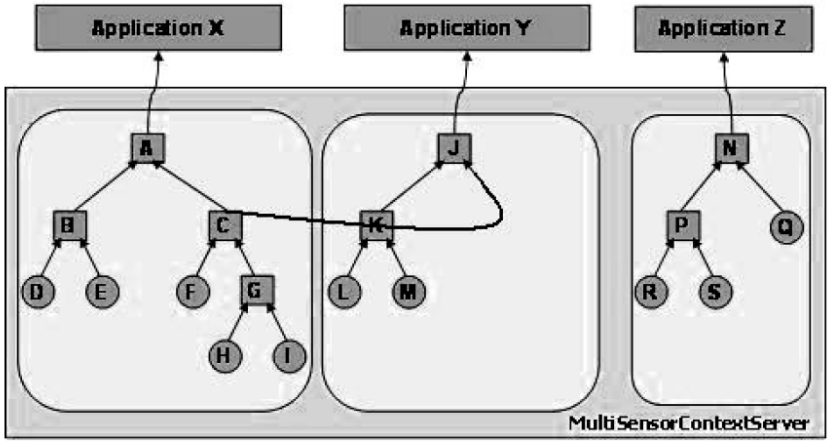

Fig. 6. Different Applications Using The Structure to Generate Information

Each tree, of course, requires some resource to produce its value. MultiS seeks to be cost-saving by suppressing the sub-trees whenever possible. This process is carried out at resource allocation time. Before allocating the resources for each node, the system searches for an existing means of providing the same information. It such a node is found, no new resource is allocated, and a pointer is added to the existing node instead.

The search for nodes providing the same information is carried out on the basis of the CD-XML definition of each node. If a sub-tree with a given definition is found, it provides the same value. It should be pointed out that this algorithm does not find all the possible nodes, but only the ones with the same definition (i.e., nodes with a different syntax but where the same semantics are not detected). A future enhancement may find a better algorithm for this.

The suppression of the sub-tree is shown in Figure 7. There is only one sub-tree in node C which has two pointers, each one for its consumer. Note that now the information is not arranged as a tree, but as a graph. However, the applications know nothing about the graph, but regard it as just an internal server representation.

The calculation algorithm must be changed to support the graphs. Whenever a node has its value changed, all the consumers must be re-evaluated (it may now be a list of consumers, not just one). If node C is changed, both nodes A and J must be re-evaluated. From the standpoint of the application programmer, the way the information is shared is transparent. Each application receives information as

described in its CD-XML.

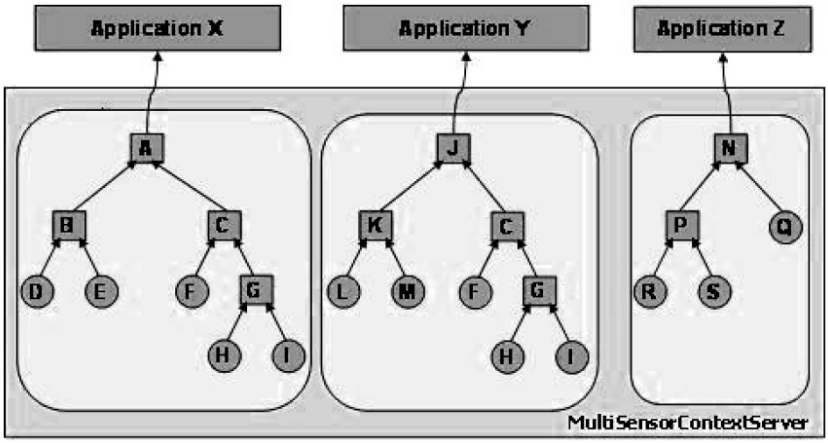

Fig. 7. Share of Context Information

3 Case Study

This section outlines the PerMuseum application, a pervasive context-aware application that enhances the user experience when visiting a museum. This application was used to test the characteristics proposed in MultiS project.

3.1 PerMuseum: A Pervasive Application

Some of the most promising uses of Pervasive Computing are designed to enhance the existing applications. The Context-Aware Computing characteristics enable a more valuable form of communication to occur between the system and the user. The Context-Aware Computing was used, for instance, to select the best content in a tourist site for each user [7] [13].

Existing Application: Several museums such as New York's Guggeinheim in New York and Malba in Buenos Aires make an audio guide available to the visitors. This material holds information about several museum attractions. The user identifies the desired attraction by typing the attraction number on the keyboard of the device. Whenever the user moves and changes his focus from one attraction to another, it is necessary to type the I.D. of the new attraction. This constant explicit interaction can interfere with the user experience.

Context-Aware Computing can be used to enhance the user experience by automatically selecting suitable information for each user. Several data sources about the user and the environment can be used to filter the available content and select what is appropriate.

The user location can be used to identify the attraction which holds the users interest. Apart from the location, the device characteristics might only select the supported types of information (e.g. if the device does not support audio reproduction, or if the headphone is currently unplugged, the audio information will not be

selected). Finally, the user characteristics are also an important filter constraint. Supported languages, user preferences, or the user's physical handicaps are also important sources for the selection process (e.g. the system will not select visual information for visitors with sight defects).

The PerMuseum Application: The first step in using the PerMusem is to fill in a short form about the user's characteristics. This information will be used as a source for some sensors later on. After the registering process, the Application sends the CD-XML request to the MultiS, and subscribes to receive content about the museum attractions. From this moment, the system is ready to work and adapt the available content automatically.

When the user changes his location from one place to another, this is detected by the location sensors. The MultiS checks if the user focus has moved from one attraction to another. If the focus has really changed, it starts searching for the appropriate content about the new attraction required.

At the same time as the location monitoring, the system searches for other museums events that may suit the user's characteristics. A scheduled guided tour, a lecture or even a movie video are some examples of such events. If anything useful is detected, the system tells the user at a suitable moment.

Programming PerMuseum with CD-XML: Figure 8 shows the context-recognition tree generated up on a CD-XML request from a PerMuseum application. The labels inside the circles show the label of information or sensor name for the leaf nodes. The letters display the filters used to produce the information, as indicated. Some information used by PerMuseum is of specific interest for each user (e.g. his location), but there is some information that is common to all the users and can be shared across the applications on a given environment (e.g. events). The data which can be shared is displayed in the form of gray circles.

4 Tests

This section examines two sets of tests of the PerMuseum application which are set by the existing MultiS implementation. The first test studies how the system reacts to changes in the environment during a given execution. It focuses on the application behavior. The second compares the system's response time with and without the feature of sharing information nodes, the focus is on the overall performance of the system.

The tests were carried out using the current MultiS implementation. The prototype was developed in Java because of its platform independence. Apart from the filter repository, all the other features were fully implemented. The tests were conducted using an AMD Athlon X2 64 4200+, 1Gb, and all the sensor data were emulated.

4.1 Reacting to Environmental Changes

The graph in Figure 9 demonstrates the application behavior found in an executed test. The test shows what happens when the location sensor changes over a period of

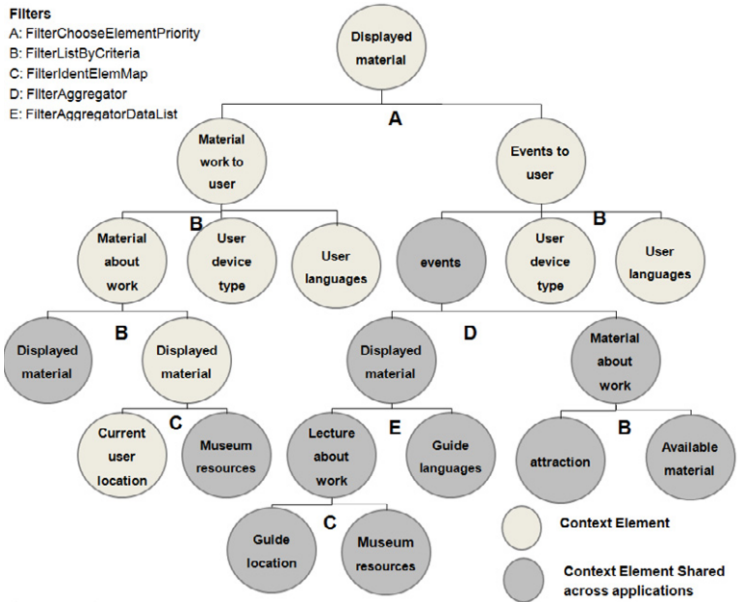

Fig. 8. Graphical Representation of PerMuseum CD-XML

time, and indicates if the user has moved to another attraction. Each sensor change triggered the sub-tree calculation and the adaptation of the application. The y axis represents the ID information, and the horizontal lines show the application state on each event. The vertical lines represent the momentum at which the application behavior has changed.

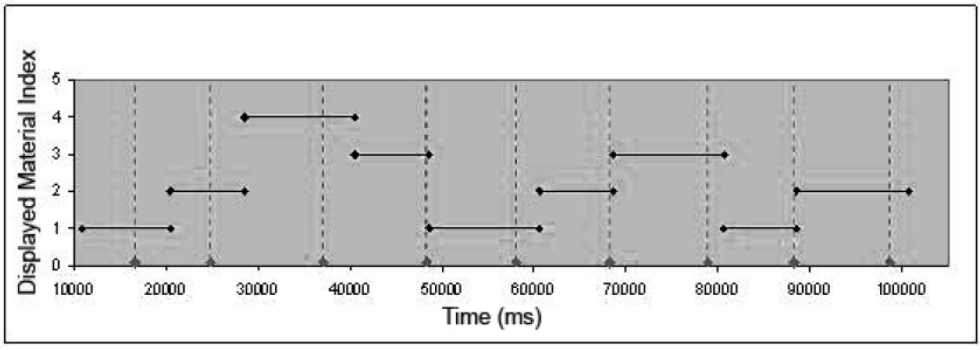

Fig. 9. Reacting to Environmental Changes

It can be observed that the application starts the execution by showing the material whose ID is 1. After some time, the location changes and the behavior of the system is updated, reflecting the location of the new user.

The monitoring system used in this test was emulated and not supports the ability to notification of changes in sensor values. In view of this, the pooling strategy was used to detect environmental changes and queries were made every second. When the sensor changed nearly at the end of this period (as in the fourth sensor change) , the system reacted quickly. However, when the sensor changed at

Table 1
Test Results Without Information Sharing (times in seconds).

Number of Applications	1	5	10	20	50	100
Average Time	12.7	49.35	79.8	179.45	252.75	957.8
Avg Time / Application	12.7	9.87	7.98	8.9725	5.055	9.578
Shortest Time	2	6	20	62	112	225
Longest Time	26	137	176	613	762	2528
Standard Deviation	6.148	34.717	38.717	145.369	152.391	841.069

the beginning of the pooling interval, the system had its reaction time delayed. The reaction time also includes the time required to perform the sub-tree calculation. Several dozen tests were conducted and it was found that the system kept a constant behavior.

4.2 Testing the Performace With and Without Shared Information

The next test demonstrates how the sharing of some contextual information can impact on the systems overall performance. The system was analyzed with and without the feature of shared contextual information. The performance criterion was the time required for the calculation of all the subscribed applications after an environmental change.

Each filter operation takes up some processing time. The total amount of time for processing the whole required inference tree is the sum of the processing time of each tree. This time increases in proportion to the number of subscribed applications. This processing time states the number of supported applications that there are in a given environment. If this time is too long, the time to adapt the application to the new behavior may be greater than the sensor refresh rate.

Each module subscribe to the sensor or processed information node. A list of all subscribers is kept on each node and when an event is triggered, each subscriber is notified. There is a cost associated with communicating the change for the different subscribers. In the current design a thread is used to perform the communication to each subscriber.

Table 1 shows the test results for PerMuseum without the feature of sharing data across applications. The sensor refresh rate was 1s. The results were calculated on the basis of 20 executions for each number of applications.

It can be seen that there was a degradation of performance when 100 applications were being executed at the same time. There were cases where the processing time took more time than the sensor refresh rate. That is, the system was still processing information when new sensor data was being perceived by the system. There were big differences in time between the maximum and minimum processing time, as in the case of the standard deviation. The system was very close to supporting the maximum number of applications.

Table 2
Test Results With Information Sharing (times in seconds).

Number of Applications	1	5	10	20	50	100
Average Time	11.15	37.85	44.55	106.7	137.3	254.85
Avg Time / Application	11.15	7.57	4.455	5.335	2.746	2.548
Shortest Time	1	2	12	29	51	134
Longest Time	33	98	150	221	251	529
Standard Deviation	9.404	29.612	35.826	54.0789	53.783	125.602

Table 2 show the results for the execution test of the MultiS application when the sub-tree sharing is enabled. When data that has already been processed was used, the processing time was much shorter. There was no performance degradation during the executions of 100 applications

The standard deviation kept within the normal limits. None of the tests had an update time greater than the sensor refresh rate. It can be concluded that, when both Table 1 and Table 2 are compared, this MultiS will be able to support more applications with the strategy of sharing information nodes. It should also be noted that the processing times are highly dependent on the request type of application, and its capacity for sharing data.

5 Related Work

Some important studies that have sought to solve the problems related to the context-recognition process, are reviewed in this section. Among them are Solar, Context Toolkit, Aura/CIS and JCAF. Context Toolkit [8] is focused on the development of a new methodology for context-aware applications. This was very important and inspired several other works in this area. However, in the proposal of the Context Toolkit, the treatment of the context forms a part of each application, since it is hard to share context data across applications.

Solar [6] makes an interesting proposal to share contextual information. Each information provider is a data flow that is processed and organized by using operators, and shown in graphs. Each operator has several characteristics that must match to allow a connection. However there is no mechanism for dealing with mobility and disconnection, which are very important features of context-aware applications.

The Aura/CIS [10] proposes a software layer called query synthesizer, which simulates a database management system. Applications query the Context Server by using a language that is close to SQL. Each of these queries must be sent to the context-management system. However, there is no subscribing mechanism, which is very important to react to changes in the environment. The applications must pool the sensor to detect and react to changes on the environment.

The JCAF project [4] provides a framework to help in the development of

Table 3
Characteristics analyzed in related research studies and MultiS.

	[8]	[6]	[11]	[4]	MultiS
Sharing of processed data	X	X		X	X
Independence between treatment and application		X	X		X
Support for mobility / disconnection			X		X
Monitoring / treatment of type publish/ subscribe	X	X		X	X
Treatment customizable, adaptable to each application			X	X	X
Composition of new contextual data without any programmingn		X	X		X

context-aware applications, although this does not deal with mobility and disconnection. Moreover, it does not enable the produced information to be shared across different applications.

MultiS is capable of producing contextual data based on several sensor data that use contextual inference trees. The produced information may be shared in the applications. There is independence between the application code and the treatment of contextual production. This coupling occurs by means of CD-XML. This language makes available a mechanism for dealing with undetermined values and may be used to handle disconnection situations. Applications may make a query by using explicit call or using publish/subscribe. The handling of contextual information is carried out through filters that can be shared across applications, and thus save programming effort.

Table 3 illustrates the characteristics of these related studies. On the basis of this Table, it is clear that none of these studies deals with all the analyzed features, except for Multis. This means that a scheme which covers all these context-server features, can make a contribution to context-aware applications.

6 Conclusions and Future Work

In the last few years, there has been a significant rise in the popularity of Mobile Computing. PDAs, cell phones with Bluetooth or WiFi enable the access to environment resources. And according to [10], the wireless revolution is just beginning.

This is the current scenario where context-aware computing is increasing its importance by enabling devices to use the available resources. Despite this, a number of challenges are still waiting to be resolved. Several works are being undertaken to support the execution of this new kind of application. But with regard to mobility and context-aware applications, none of them has yet attained the required characteristics. The MultiS project seeks to fulfill those requirements, and is used as an infra-structure support for context-aware applications.

Today, every programmer that needs to develop a context-aware application, must program all the stages of the adaptation process: the monitoring, the context recognition and the adaptation. This adds significant development costs and inhibits the development of this kind of application. However, there have been some studies to support the execution of this application which have been carried out recently. Despite this, with regard to the stage of context recognition, none of the related work has been able to acquire the characteristics needed for context-aware applications.

We propose MultiS to fill this gap. By using MultiS, the integration of information from different sources can be made possible through CD-XML. This simple language enables contextual data to be produced while containing a mechanism for handling disconnection events. Through the subscribe/notify mechanism of CD-XML, applications are able to react to environmental changes. MultiS middleware also supports the execution of several different context-aware applications concurrently.

Some ideas for further studies arose while undertaking this paper. They could be the subject of future work and are as follows:

- CD-XML extension to enable searching for sensors.
- Examining the question of the privacy of the produced information.
- Identification of sharable information with different specifications.

References

[1] Andrew, A., *The New Age of Wireless*, Scientific American,**295** (2006), 20.

[2] Augustin, I., A. Yamin, L. Silva, R. Real, G. Frainer, and C. Geyer, *ISAMadapt: Abstractions and Tools for Designing General-Purpose Pervasive Applications*, Software, Practice & Experience - Special Issue "Auto-adaptive and Reconfigurable Systems", Wiley InterScience "SP&E" journal, **36** (2006), 1231–1256.

[3] Baldauf, M., S. Dustdar, and F. Rosenberg,*A survey on context-aware systems*, Int. Journal of Ad Hoc and Ubiquitous Computing,**2** (2007), 263-277.

[4] Bardam, J. E., The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications, *Proceedings of the Third international conference on Pervasive Computing*, (Berlin, Heidelberg, 2005), 98-115.

[5] Bettini, C., O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, *A survey of context modelling and reasoning techniques*, Journal of Pervasive and Mobile Computing, Special Issue on Context Modelling, **6** (2010), 161-180.

[6] Chen, G., " Solar: Building A Context Fusion Network for Pervasive Computing, " Doctorate Tesis, Dartmouth College, Hanover, New Hampshire, USA, 2004.

[7] Cheverst, K., K. Mitchel, and N. Davis, *The role of adaptive hypermedia in a context-aware tourist GUIDE*, Communications of the ACM, **45** (2002), 47–51.

[8] Dey, A., "Providing Architectural Support for Building Context-aware Applications, " 183 p. Doctorate tesis, Georgia Institute of Technology, Atlanta, USA, 2000.

[9] Hofer, T., W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger, Context-Awareness on Mobile Devices - the Hydrogen Approach, *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, (Washington, DC, USA, 2003), 292.1–.

[10] Hong, J., E. Suh, and S. Kim, *Context-aware systems: A literature review and classification*, Expert Systems with Applications. **36** (2009), 8509–8502.

[11] Judd, G., P. Steenkiste, Providing Contextual Information to Pervasive Computing Applications, *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications* , (Washington, DC, USA, 2003) 133–.

[12] Kappel, G., W. Retschitzegger, E. Kimmerstorfer, B. Prll, W. Schwinger, and T. Hofer, Towards a Generic Cust om isation Mo del for Ubiquitous Web Applications, *2nd International Workshop on Web Oriented Software Technology*, (Malaga, Spain, 2002) 79–104.

[13] Londsdale, P., R. Bealel, and W. Byrne, Using context awareness to enhance visitor engagement in a gallery space, *HCI2005 (Human-Computer Interaction)* , (Edinburgh, England, 2005) 101–112.

[14] Satyanarayanan, M., *Pervasive Computing: Vision and Challenges*, IEEE Personal Comunications, New York, **4** (2001), 10–17.

[15] Schaeer, A., A. Yamin, I. Augustin, L. Morais, and C. Geyer, *PerDiS: A Scalable Resource Discovery Service for the ISAM Pervasive Environment*, International Workshop on Hot Topics in Peer-to-Peer Systems, Los Alamitos, **0** (2004), 80–85.

[16] Yamin, A., I. Augustin, L. Silva, R. Real, and C. Geyer., EXEHDA middleware: Aspects to Manage the ISAM Pervasive Environment, *XXV International Conference of the Chilean Computer Science Society* (Valdivia, Chile, 2005) 84–.

[17] Weiser, M., *The computer of the 21st Century*, Scientific American. **265** (1991), 94–104.