RELEVANT CONTEXT DISCOVERY FOR PERVASIVE SERVICES INVOLVING USER CONTROL

Marcel CREMENE[1], Alin DRIMUS[3], Jean-Yves TIGLI[2], Stephane LAVIROTTE[2], Gaëtan REY[2], Michel RIVEILL[2], Costin MIRON[1], Mircea VAIDA[1]
[1]*Technical University of Cluj-Napoca, Communications, {cremene, vaida}@com.utcluj.ro,miron@bel.utcluj.ro*
[2]*University of Nice Sophia-Antipolis, Ecole Polytechnique, {tigli, lavirott, gaetan.rey, riveill}@unice.fr*
[3]*University of Southern, SDU, Denmark,drimus@mci.sdu.dk*

Abstract: Dynamic context adaptation is a central concept in Pervasive and Ubiquitous Computing. In this paper we have analyzed the problem of context relevance and we have chosen an approach where the user is also involved also in the relevance control. The advantage of involving the user is motivated by the fact that a completely automatic context management cannot deal with unpredicted situations. The proposed solution was implemented on the top of WComp middleware. An easy to use interface let the user the possibility to easily manage its context.

Key words: *Ubiquitous computing, Relevant context, Context discovery, Intelligent Devices.*

I. INTRODUCTION

Background and terminology. *Pervasive* and *ubiquitous* computing domains have known a very dynamic development in the last years, especially due to the development of mobile technologies that need new services in order to become more efficient and profitable.

The majority of the research directions from these domains are starting from the same idea: the services/applications should adapt (dynamically) to their *context*.

The problem addressed in this paper is related to relevant context discovery in pervasive and ubiquitous computing. We are interested especially on the intelligent environments (buildings) domain.

The *context* concept is a well known one in the pervasive/ubiquitous domains. In our approach, the context *C* includes all the elements surrounding a user *U* that interacts with a service *S*.

The context is described mainly by the next aspects:

- Available (software) services that may be found around the user, other than the services used already by the user,
- Hardware infrastructure (devices, networks, etc.)
- User's need, preferences, social context, etc.
- Physical context (position, time, temperature, light, noise, etc.).

Motivation. Theoretically, the context is composed by all the external elements that may affect the usage of a service S by a user U.

If we push the things really to the limit, the context become extremely large and even infinite, and thus unusable.

In practice, usually, the context is finite and established a priori by a human expert. Here, there is another limitation because if all is established a priori, the context cannot evolve if the services or the user needs unpredictably change in time.

In order to enable the context dynamic evolution, a context-aware system should have a *context discovery* mechanism.

By *relevance* of the context we understand a function associated with each context element, that makes a difference between these various context elements, from the point of view of a specific user *U*, that is using one or more specific services $S_1 \ldots S_k$, at a specific moment *T*.

The relevance aim is to simplify the active space and help the user to focus on the aspects that are the most important for him (while using a certain service).

The relevance is partially deduced automatically from the context state (using some predefined rules) and partially specified by the user directly (through a control interface).

Scenario. In order to have a better understanding about the motivation and how the context influences a certain user U interacting with a service *S*, we propose the following scenario.

A professor P intends to make a presentation in the class room 1. He opens his PDA, where the presentation is stored, and starts the presentation service. For the moment, the service is reduced only to the presentation viewer from his PDA.

The viewer service tries to connect to a projector service. The professor is located in room 1. Here, the (detected) available services are: projector from room 1, projector from room 2, light from room 1, light from room 2, light sensor from room 1 and light sensor from room 2.

The PDA viewer service will connect to the projector service from room 1 and ignore the other services for the moment. In this first case, the relevant context is composed only by the projector located in the user's room. This fact may be deduced automatically because it does have no sense

Manuscript received Mai 02, 2009; revised June 14, 2009

to use a projector from another room.

Let suppose now that, at a certain moment, the external light becomes too intense. Now, the professor needs a new service in order to maintain a uniform light level in the room. In this second case, we see that it is necessary to add to the relevant context two additional elements: the light level given by the light sensor service and the light control service. This fact is triggered by the professor need for a certain level of light and it requires the user control.

Objective. The aim of this paper is to propose a context discovery mechanism based on the context relevance function and show how this mechanism may be integrated into a context-aware middleware. The user is also included in the relevance decision process.

Approach. In our vision, three approaches are possible in order to determine the relevant context:

a) The relevant context is specified by the user (using an easy interface if possible). This approach may not be such a good solution if there are a very large number of context elements and it is difficult to the user to decide its zone of interest.
b) The relevant context is automatically deduced based on some general, predefined rules. But some of the user preferences cannot be known a priori by the system developer and the general mechanisms cannot deal with particular cases and with unpredicted situations. All predefined solutions are limited by definition.
c) The relevant context is deduced partially automatically and partially chosen by the user. This approach tries to combine the advantages of the both approaches described before.

In this paper we have chosen the third approach because it gives the user the possibility to freely decide its zone of interest and it is also a solution for unanticipated adaptation.

Outline. The paper is organized as it follows: the next section presents several context-aware platforms and analyzes how they address the issues discussed in the introduction section, showing their limitations.

Section three present our proposed solution, based on a context filtering mechanism.

Section four presents a prototype that was implemented in order to test our proposal and the results that we have obtained.

The last section contains the conclusions and the further work.

II. RELATED WORK

In this section we will analyze some existent context-aware platforms in order to see how the *relevant context* is determined. We also discuss briefly about the context models. The end of this section presents our conclusion about the limitations of the existent solutions.

GAIA [3] is middleware support for active space environments such as smart rooms and living environments. It essentially provides a distributed operating system where all inputs, outputs and processing units within a room are considered as a single computer.

GAIA uses a component repository and centralized approaches to events, and services discovery. Code can be updated replacing components in the repository.

The context is described using data collection called predicates. Some example context predicates are: Context(location, Chris, entering, room 3231), Context(temperature, room 3231, is, 98 F), Context(social relationship, Venus, sister, Serena).

First order logic formula may be applied on these predicates in order to deduce the action to be taken function on the context.

CORTEX [4] proposes a novel sentient object model to address the emergence of a new class of application that operate independently of human control. Infrastructure-based and ad-hoc based wireless environments are considered to address mobility. The middleware is highly configurable at run-time. It reacts on events by changing the behavior of objects.

In order to select the relevant context, CORTEX uses *filters* that provide a basic mechanism to allow objects to express interest, or lack thereof, in events of a certain type and *zones* that introduce a means of scoping or limiting the propagation of event notifications in the system.

Aura [5] is a context-aware middleware which can be used to create mobile applications. It represents the user by its aura, like a Personal Area Network (PAN), and brings the appropriate resources from the services of the environment to support the user's task.

Context changes are notified by events, and tasks can change while context is evolving. It's also interesting to note that it suspends tasks which cannot be processed anymore due to a context change, storing their state for a future resume.

A key context element is considered to be the user location information and various methods may be used in order to locate the user. Another key aspect of the context is considered to be the time (ex. scheduled tasks). Other context aspects are related to the user profile, personal data (that should be also secured). Some constraints can be expressed in task descriptions, and the middleware restricts some of its operations.

Oxygen [6] is an MIT project which addresses human needs using speech and vision technologies that enable the user to communicate with it as if the user were interacting with another user.

An application of this project is to define intelligent networks with dynamic topologies, according to devices locations. There are fixed and mobile devices with embedded software. Code can be automatically updated thanks to that. Network rules can be specified to allow sets of users to use particular resources.

The Oxygen project proposes the idea of pervasive human-centred computing: it supports nomadic users changing various terminals which adapt themselves to the user needs; provide multimodal interfaces; and it is *intentional* meaning that it must enable people to name services and software objects by intent (for example, "the nearest lamp").

Context models. The context can be defined as the information that surrounds and identifies a specific entity, being software, hardware or human based.

In [1] we can see that the two most important and relevant elements for providing a description of the context could be the human factors and the physical environment. Human factors related context is structured into three categories: information on the user (knowledge of habits, emotional state, bio-physiological conditions, etc.), the user's social environment (co-location of others, social interaction, group dynamics, etc.), and the user's tasks (spontaneous activity, engaged tasks, general goals, etc.).

Likewise, context related to physical environment is structured into three categories: location (absolute position, relative position, co-location, etc.), infrastructure (surrounding resources for computation, communication, task performance, etc.), and physical conditions (noise, light, pressure, etc.).

Conclusions about the existent solutions. In the technical report [7] were analyzed another several context-aware platforms. The conclusions about their main limitations, as well as for the platforms discussed in this section, are discussed bellow.

A first limitation of the existent solution is that the majority of the context-aware platforms are not taking into account the context dynamic discovery. A platform that integrates intelligent devices that may appear and disappear dynamically is WComp [2], [8] based on UPnP discovery protocol.

Another limitation of the existent solutions is related to the relevant context management: the large majority are based on the idea that the context (and also the reaction mechanism rules) should be established a priori by the service/system developer. But, as we have explained in the introduction, the user intervention is absolutely necessary in order to enable an unanticipated relevant context discovery.

III. PROPOSED SOLUTION

The solution that we describe here is a context-aware system that allows the user to choose the relevant context elements in order to select efficiently the most relevant services that may be found around the user at a given moment.

The environment that we are targeting is related to the intelligent buildings where various devices offering services may be found (and may disappear) spontaneously.

After analyzing our problem, we have decided that the following issues need to be solved:

1) *Propose the architecture for our context-aware system*. In order to do that we reuse and adapt the context discovery mechanism existent in the context-aware WComp middleware [2], [8], which is based on the UPnP devices/services discovery mechanisms. However, this mechanism will give us the perceived context, not the relevant one.
2) *Propose a context model*, according to the context definition presented in the introduction. Find a technical solution to add the necessary meta-data to the WComp entities (devices, services, users, etc.).
3) *Propose a filtering mechanism* that will filter the relevant context from the perceived context. Provide this mechanism with an *intuitive user interface* that will enable the user to indicate its needs.
4) *Integrate the filtering mechanism* with the WComp platform and with the AoA mechanism (Aspect of Assembly compositional and adaptation patterns) and make some tests in order to show the difference between the adaptation with and without the filtering feature.

Architecture. The proposed architecture, realized on the top of WComp platform, is depicted in figure 1.

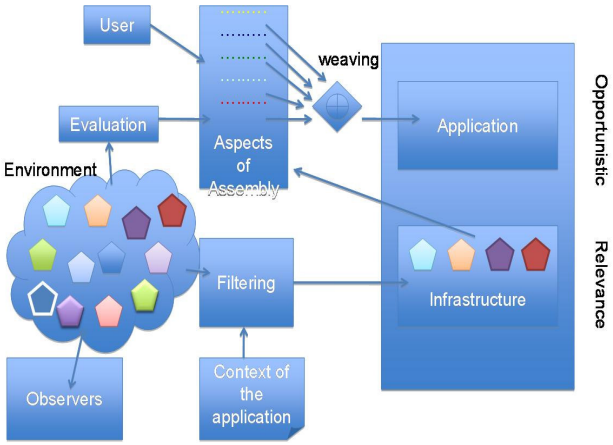

Figure 1. System architecture

This architecture includes the following parts:

- The *Infrastructure* is based on WComp middleware and it manages the services that may be used at a given moment. The service instantiation respects the services relevance for the user and it is controlled by the *Filtering* module.
- The *Filtering* module is responsible for selecting, among all the available services, only the services that are relevant for the user.
- The filtering operation takes into account the user and the application (the new service) context. The context is observed using some probe/sensors components.
- The *Application* represents the new service, created according to the user needs. The application is created (weaved) according to some assemblage patterns called AoA (Aspects of Assembly), described in [2] and [8]. These patterns not only describe how to create an application from basic services but also how that application evolve if the context changes dynamically.

Figure 2 depicts a detail about the context filter part. As we may observe, from all the available UPnP (the underlying protocol of WComp) devices, only the relevant ones are instantiated in the WComp container.

Examples of observer components are: a badge reader that controls the doors so we can know when a person have entered into a specific room and a temperature sensor.

Context types. Several types of context can be defined as it follows:

- The *global context* (utopia context), *Cg*, contains all the entities that exists in a given place and may go, at limit, to the entire world: an internet connection means a possible connection to all internet available services. This kind of context is

almost impossible to use because it is too complex and the majority of the included entities are not relevant for a certain service.

- The *perceived context*, *Cp*, for a given entity O, is the context that may be perceived, or observed, by this entity O. For instance, the professor's PDA detects all the Bluetooth services from a specific range, from room 1 and room 2 but not from room 3 that is too far.
- The *relevant context*, *Cr*, for a given entity O, is a part of the perceived context that has a specific relevance for that entity O. In our example, we may say that, the relevant context contains only the services from room 1 because the user and its terminal are located in the room 1.

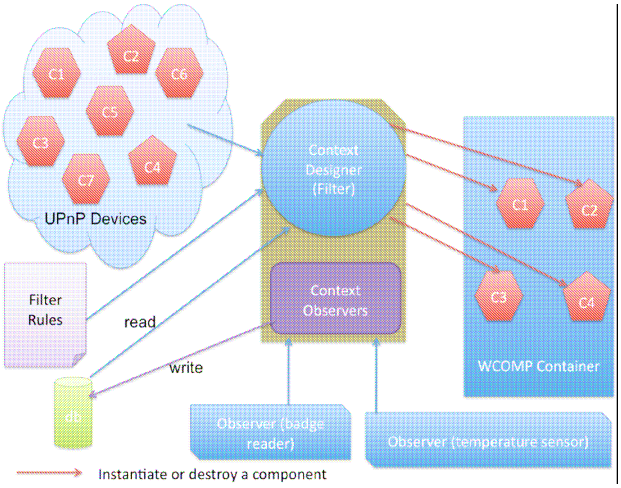

Figure 2. Context filter

Context model. Context information can be represented in lots of ways. There are different opinions about how to represent context, ontology, different forms of databases, xml or other forms. Many see context highly connected to the application or to what we consider useful.

Keeping in mind that the simpler is the better, we tried to imagine a context as the minimum useful information that can be used in order to allow the context awareness.

As context information is not yet agreed upon, having a lot of alternatives, we propose a way to allow the user to define the meaning of context.

This is very important because it will keep the application relevant to how the user imagines context. We propose representing the context information by context elements, grouped in a tree, each node having contextual information attached to it. The user may add or remove elements, or can modify the structure of this context model.

This is a very simple representation, but yet very effective, as important elements can be grouped, and can be hierarchically arranged in order to reflect the real life scenario. This is helpful if the user has not yet decided on a full model of context, or in case the user wants to decide later about the context model.

The model of the context that the user can build is called the context skeleton and it is an xml file that has the contents the same as the tree elements defined by the user. This context skeleton can be exchanged by different users, and can be easy read or edited (figure 3).

Context Information Repository. When dealing with context information it is important to decide if we want to have a centralized approach for the context (a database with all the context information for all devices in the network) or we want to have each element from the context hold it's own contextual information.

This problem is highly debatable, but if we consider that we work with relatively simple devices (most of them are sensors) that have limited functionalities and do not possess the resources to hold some sort of contextual information or representation, then we may say that a more simple approach should be taken into consideration. This approach is the database or the centralized repository.

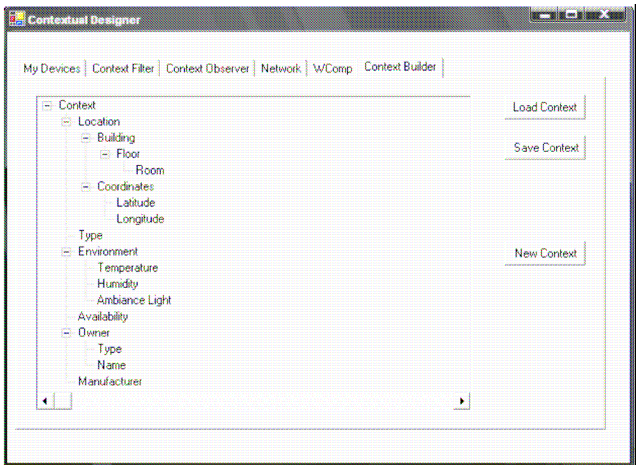

Figure 3. Context Builder

In such a manner, we could build our own model of context, and the device should not need to carry the context description. This description is only in the repository. Of course, devices don't have any possibilities to store context information at this moment, but let's suppose they did, we may face another problem: what if we think of the context to reflect only location of the device or user, and somebody else would like to view their context as only to reflect their type (laptop, PDA, smart phone). In this case, the devices carrying their own context information will have difficulties providing the context that we desire and in the meantime providing another context to the other user.

Such a perspective would require intelligence in the devices, and let's face it, most companies that build cheap devices, don't bother with even thinking about this. Unfortunately the centralized approach is not an ideal alternative, because we may consider that the database should be known before and this database should be always on and always accessible. This is not always the case in the real scenarios, where we may want to have different kinds of applications that do not have access to all the resources.

Observers. Let's suppose we work with static sensors that are initialized and have all context parameters fixed. In this case we do not have a changing context, because nothing changes for any device, except their sensed values.

If we want to work with mobile devices for example, or other kinds of devices that change their parameters, then we may get in the situation that we cannot address all the changes in the environment (because it changes too fast) and

we need someone or something that will keep updating the information for all the mobile devices in our central repository.

There are devices and services that can observe the environment and can advertise changes in it. These devices or services work under the principle publish/subscribe (protocol implementations like UPnP or DPWS) and can provide different kinds of updating information for our repository of context.

Therefore, we should subscribe to such services to keep us informed when elements from the environment change their context parameters.

Adaptation. The mechanism of adaptation of the middleware should address both the infrastructure (devices or services) and the application that the user is interested in. To address the changes in the infrastructure one of the most important aspects of such a mechanism is the filtering, seen as the process of filtering out the unwanted devices or services.

In this way, we consider the infrastructure adaptation as being a selection and grouping of only the same context related devices.

In order to get the same context related devices, we need to define somehow the context we are interested in. This context can be called a narrow context because it doesn't need to reflect the context model entirely, being just a subset of the context model.

For example, we may want to describe that we belong to a context consisting in only the location, devices type, and their owner type. From all the devices and the services that are present in the environment, we consider in "our context" only the devices that have elements in common with what we want.

If we may want to use only the devices that are present in Room 102, or we may want to use only the devices that have their type: PDA then we define the context that we are interested in as having just one element: Room: 102, or type: PDA. We have defined the notion of our context and its boundaries; therefore this context is actually the context of the application.

We want to see only the devices from Room: 102 because we need to use only these devices, and no other devices. We want to see only the devices of type PDA because they offer services that we want to use, and we do not want the services from other types of devices. In every case, we take the device's full context and we compare it with the context of the application that we defined. If it does not match, then we filter out that device. Because each device already has a full context associated with, we can decide based on what narrow context we define if that particular device or user is actually what we need.

As a result, we will filter out all the unwanted devices or services, and we will obtain a selection of entities of interest for us. The reunion of these entities that are of interest can be seen as the relevant infrastructure. It is actually an infrastructure because it contains the entities that are used to build higher-level services with, and it is relevant because it will always keep an up to date list of entities that belong to our context, reflecting changes in the environment. If some devices change their context in the environment, the relevant infrastructure will be up to date considering those changes. New entities will be added or existing entities will be removed from the relevant infrastructure to reflect their belonging to our context or the opposite.

The *relevant infrastructure* will consist of entities that are in our context, and this is actually the first phase of adaptation. The second concept is called opportunistic and it is based on the fact that the application can change its components and links internally in order to provide a specific service for the user. It can be seen as an opportunistic approach because it tries to obtain the best possible configuration in order to provide a specific service. The application can readapt based on aspects of assembly, but it needs to consider a relevant infrastructure in order to obtain the desired result.

IV. IMPLEMENTATION AND RESULTS

As we have said, for implementing our solution, we have chosen as support platform the WComp middleware [2], [8] in order to reuse the existent infrastructure and features offered by the WComp platform and a special feature called AoA - Aspects of Assembly.

We will describe in this section the main part of this implementation.

Centralized repository. To avoid some disadvantages for the cases when networking resources and computing power is limited, we will consider an approach with a centralized repository, but only for the current application.

This makes the database local and not dependent on other resources available on the network. This local database approach should consider the fact that a database model implemented in a structured query language can be difficult to work with and this local database should be accessible for reading or editing not necessarily using complicated mechanisms, but more easily. We have created an interface that enables us to set the context.

```
<?xml version="1.0" encoding="utf-16"?>
<ContextDatabase>
<Device                                  Name="Light
(4D891D893F914BC)(http://10.211.55.3:64452/)"
UDN="12345" tag="::Device::">
<node                                    text="Light
(4D891D893F914BC)(http://10.211.55.3:64452/)"
imageindex="-1" tag="::DEVICE::">
<node text="Location" imageindex="-1" tag=""><node
text="Building" imageindex="-1" tag="">
<node text="Floor" imageindex="-1" tag="1"><node
text="Room" imageindex="-1" tag="4" />
</node></node>
<node text="Coordinates" imageindex="-1" tag="">
<node text="Latitude" imageindex="-1" tag="" />
<node  text="Longitude"  imageindex="-1"  tag=""
/></node></node><node text="Type" imageindex="-1"
tag="" />
<node     text="Environment"     imageindex="-1"
tag=""><node  text="Temperature"  imageindex="-1"
tag="" />
<node  text="Humidity"  imageindex="-1"  tag=""
/><node  text="Ambiance  Light"  imageindex="-1"
tag="" />
</node><node text="Availability" imageindex="-1"
tag="" />
```

Figure 4 XML Context database

Therefore, a local XML database (fig.4) file is seen as a convenient and easy alternative to any structured query language database. The application should consider the context information as a centralized repository for better context representation, but also should allow the possibility for users to use any existing repository. In this case an xml database file could be exchanged between different users.

The devices that we want to work with should be initialized at their first use in the environment. This can be seen as a registration process.

After this registration process, the device or service can be easily associated with context information, and it's context information can change over time under the surveillance of the observers. The registration process is important because it lets us describe some information according to the device's context that we know about (like the owner, the type of the device, the utility) or any other static parameters that will be used in the application (like location for static sensors). This registration needs to be done only once, and after this, every time the device will reappear in the network, it will be associated with it's known contextual information.

Observers. There are many intelligent systems that provide means for notifying if changes have occurred in the environment.

For example, we can have a badge reader service that will provide information for a specific user (what room is he located in).

This service should update the context for the user, in the central repository of the application. This should be done automatically, but how to define these observers? To define such an observer mechanism, we need to say: what device with what service will observe a certain device (or user) to what contextual element.

For example in the badge reader case: BadgeReader Device with Location Service will observe a user and will modify his Room parameter for that user's context.

Filtering. Rules are added with AND or OR as operators to allow the building of a more complex context of the application. Suppose we have a professor that will want to build an application with a video projector device inside the classroom 202 and it will need also to have in his application all the PDA's of the students that are present in the classroom. He will need to define the context of the application as: Room: 202 AND Devices Type: PDA. If there are badge readers that act as observers, any student that will come inside the classroom, will be observed, and his device will change it's context to reflect the new location.

The device will be added to the relevant infrastructure, along with the fixed video projector from the room 202. If there are students that leave the room, they will also be observed, and they will be not considered anymore in the relevant infrastructure. From all the devices that are present in the school building, he will get only those devices he is interested in.

The relevant infrastructure will reflect the context he defined (the room and the type of devices) and after that, a set of aspects of assembly are applied and inside the application we will obtain an assembly of components that reflect the infrastructure, and have all the links between all the components. The role of the aspects of assembly was to make the links and to adapt the application according to the infrastructure.

V. CONCLUSIONS

In this paper we have analyzed the problem of context relevance and we have chosen an approach where the user is also involved in the relevance control process, which represents the original aspect of our paper.

The proposed solution was implemented on the top of WComp platform that has the great advantage of discovering dynamically the devices and services available for a certain user. An easy to use interface let the user the possibility to choose the services interesting from him among several existent around him.

ACKNOWLEDGEMENTS

This research was supported by the project PN II "Idei" 1062 financed by UEFISCSU.

REFERENCES

[1] Abowd, G. D. and Mynatt, E. D. "Charting past, present, and future research in ubiquitous computing". *ACM Trans. Comput.-Hum. Interact.* 7, 1 (Mar. 2000), 29-58. DOI= http://doi.acm.org/10.1145/344949.344988, 2000.

[2] Daniel Cheung-Foo-Wo, Jean-Yves Tigli, Stéphane Lavirotte et Michel Riveill, *"Self-adaptation of event-driven component-oriented Middleware using Aspects of Assembly"* in 5th International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC), California, USA, novembre 2007.

[3] Manuel Román, Christopher Hess, Renato Cerqueira, Roy H. Campbell, Klara Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces", *IEEE Pervasive Computing*, 2002.

[4] Verissimo Paulo, Cahill Vinny, Casimiro António, Cheverst Keith, Friday, Adrian and Kaiser Jörg. *"CORTEX: Towards supporting Autonomous and Cooperating Sentient entities."* In: Proceedings of European Wireless 2002 (EW2002), 26-28 February 2002, Florence, Italy, 2002.

[5] Garlan, D.; Siewiorek, D.P.; Smailagic, A.; Steenkiste, P.; *"Project Aura: toward distraction-free pervasive computing"*, Pervasive Computing, IEEE Volume 1, Issue 2, April-June 2002 Page(s):22 – 31, 2002.

[6] http://oxygen.lcs.mit.edu/

[7] Jean-Yves TIGLI, Michel RIVEILL, Gaëtan REY, Stéphane LAVIROTTE, Vincent HOURDIN, Daniel CHEUNG-FOO-WO, Eric CALLEGARI, *"A middleware for ubiquitous computing: Wcomp"*, Projet RAINBOW, Rapport de recherché, ISRN I3S/RR–2008-01–FR, January 2008.

[8] J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, D. Cheung-Foo-Wo, E. Callegari, and M. Riveill. "WComp Middleware for Ubiquitous Computing: Aspects and Composite Event based Web Services". *Annals of Telecommunications (AoT)*, 64(3-4), Apr. 2009.