CUROCO: a distributed architecture for the dynamic generation, composition and use of context in highly dynamic and heterogeneous environments*

[extended version]

Gorka Guardiola Muzquiz
Laboratorio de Sistemas, Universidad Rey Juan Carlos
Madrid, Spain
paurea@lsub.org

ABSTRACT

The use of context is necessary for a rich interaction with the users and adaptation of the system to their needs. However, how to integrate context with the applications and the system is yet an unsolved problem. Looking at other services provided by our computing platforms, we ask ourselves: How can context be made a system service?. Making the context an application neutral system service would permit to share and update it from different sources and applications, making the user able to interact with it easily. Even applications which are not context aware would be able to benefit from this approach. We are designing and implementing an application called CUROCO to deal with this issues.

Categories and Subject Descriptors

D.4.7 [**Software**]: Operating Systems—*Organization and Design*; D.4.3 [**Software**]: Operating Systems—*Distributed File Systems*

Keywords

CUROCO, Ubiquitous, Pervasive, Context Aware, Sentient Computing

1. PURPOSE DESCRIPTION

In order to realize M. Weiser vision and make computer technology ubiquitous, user interfaces must be simplified, making them require less attention and input from the user. Obtaining non-explicit input from the user context is then made necessary. The problem with previous attempts to use context are middleware infrastructures or libraries which require recoding the application or writing wrappers for them. An example of this is Aura [4]. These solutions do not solve the problem completely, because they raise the following questions: How do we share context information between applications?. How do we integrate the user in all this?. How do the applications that we already have benefit from context?. Even extremely flexible architectures like The Context Toolkit [10], are far fetched from giving a uniform context support for applications that may not be context-aware by themselves. Systems like Gaia [3] introduce context services at system level, but use it in an ad hoc fashion. For example, Gaia uses context to adapt its filesystem, but this is a particular use, not a general independent context service. Applications benefit from it, but cannot interact with context directly in a general way that might not have to do with the filesystem.

My purpose is to build a general context-aware architecture I have named CUROCO, to address all these problems. It will enable non context-aware applications to exploit context and simplify its use for new applications. It should permit changing parameters of the system based on context information. The central idea is to be able to make partial calls on operations to the system and complete them using context. Another main idea is to keep context on files actualized by the system, permitting easy sharing, use of context and interoperability. By abstracting, aggregating and serving context in independent entities completely outside the application, we may reach better reusability and interaction with the user.

2. GOAL STATEMENT

My goal is therefore to build CUROCO, a context architecture specifically designed to solve the problems stated above. At the moment, its design is composed of two parts. One is the context service, a filesystem which is filled by context aggregators, outside programs which abstract and aggregate raw data and different context values. The other is a context-aware system commander, that loops completing requests relying on the context and user interaction.

2.1 Exporting context as a filesystem

The idea is to export a distributed filesystem containing context in order to share it. The filesystem makes CUROCO able to abstract context in a portable and easy to export way. Using simple text and naming conventions, it makes simplifies context sharing between applications on different systems and makes context a service that becomes independent of the application. It can also be viewed by some applications as the part of the state of the application which depends on the context and is kept outside it. In this way the infor-

*This work has been funded by Spanish Ministry of Science and Technology, MCYT (TIC2001-1586-C03-01) and the Rey Juan Carlos University PPR-2003-40

1st International Middleware Doctoral Symposium Toronto, Canada

mation is exposed and thus can be changed dynamically by external entities.

Exposing context as a filesystem enables new applications to share and access it easily. It also makes context aggregators easy to program (they may be even shell scripts). We have already written a simple infraestructure for gathering context about our lab from different sources that we expose as files. Some of the data comes from special file servers (e.g. the X10 [2] file servers), some comes from simple files on a linux or Macos X system, or is generated dynamically by scripts or programs. It has been trivial to export it to the web, generating a web page with information of presence [9], real-world mail, and other useful data.

The rationale behind using files to export context is that it is a well known technology used everywhere. On almost every operating system we have clients and servers for SMB or NFS for example, and files can be accessed from any programming language in a simple way.

It can be argued that by using files we just push complexity to another layer, to the contents of the files, but that is not the case. Using files we solve for free (in a very portable way) the problem of having different hierarchies and a naming system. This is normally done using CORBA or some other middleware, like in the Solar naming system [1] or Gaia's context filesystem, or the nested structure of XML documents [11]. The approach of using files is better because it is more portable and simple. Not even CORBA with its multiple bindings for languages and systems can match the interoperability of using files. Even simple devices like a mobile phone using Java have a way to access files exported by bluetooth. In addition, writing a filesystem is much simpler than is normally acknowledged. An example for this is [6] where a filesystem was written using very few lines of code to fit in a Lego Mindstorm. It is also easy to reexport a filesystem on another format if it is not available for the client. For example, a Windows system with an NFS client can reexport the filesystem through SMB. Using Plan 9 or Plan B which have binding and union directories we can go even further and merge the trees from the different systems before reexporting.

This part of our architecture has the same purpose as "The Context Toolkit" [10] or the "Solar System" net of aggregators and filters. The key difference is that our architecture is much simpler and interoperates better by using only a filesystem instead of a middleware to export it to other applications.

2.2 Execution service

The system commander is an execution service that changes parameters to operations depending on the value of the context filesystem and user interaction, combining the user and the context into the execution loop for general purpose applications.

In our current design for CUROCO, all commands to the system are built out of description slots. These slots are specifications of the kind of parameters that might fill a command line, making it possible for programs to complete commands automatically. Whenever an operation is sent to the system, these slots are filled in a loop by using interaction with the users and context aggregators. They use the filesystem interface to obtain the context information. Once all mandatory slots are filled, operations can execute. To make an application context-aware the only thing needed it its slot description.

An operation to the system is a set of description slots that we call *request*. An request is composed of two parts. One is the verb. The other is the suffix. The verb is only one slot corresponding normally with a command and the suffix are the rest of the slots. A request starts with the verb or part of the suffix already completed. If it is the verb, this generates a lookup on a database for the verb suffix description (each verb has a suffix description attached). On the other side, if part of the suffix is specified (for the moment just one slot), the correct verb for that (partially incomplete) suffix is looked up in the database if none is filled in by an aggregator. This is done because suffix descriptions are attached to verbs. So we need a verb in order to use its request slot description. After getting the order slot description the execution loop starts.

During the execution loop the slot description is exposed to the context aggregators which using the context filesystem and the already specified slots can fill in the gaps till all the mandatory slots are completed. Once the operation is completed it is sent to execution.

An example of the use of this execution service can be seen on figure 1. On this figure,

1. A user clicks on a file of type *.wav*.
2. As no context aggregator has any verb for this file, some matching verb is looked up on the database (some things may happen in the middle, like running command `file` over it or looking if it exists). A verb is found in the database, a program called `wavplayer` .
3. The volume parameter is filled in (e.g. based on user state)
4. The headphone parameter is filled in (e.g. based on localization)
5. The equalizer parameter is filled in (e.g. user interaction).
6. The request is complete (all the mandatory commands are filled in) so it commits and is executed.

2.3 Using context at system level

Our main goal is understanding how to use context and how to aggregate it in a simple way at system level. Our way for doing this is taking context out of the applications and making it a system's service. Once we finally achieve our goal, it will be much simpler to program context-aware applications and to make the ones which are not context aware. CUROCO will use a completely new approach which is not comparable to any existing toolkit or middleware as far as we know.

3. METHODOLOGY

We plan to develop a prototype in three phases.

1. Exploring different architecture designd and the use of context as files.
2. Writing a simple prototype of the architecture and trying it with applications.
3. Integrating it in the OS we are writing, Plan B.

3.1 Exploring architecture designs and context usage

We have been experimenting for some time now with the use of context, considering alternative architectures. We have also tried exporting context information of our lab using files. This context comes from very heterogeneous sources, user activity programs on Linux, Plan 9 and Mac OS, fileservers exporting X10 sensors and image processing of our physical mailbox. We are using it to program applications, scripts and to show the lab state on a web page as can be seen on [9]. There is also a video demo of our everyday use of the context file system in [7].

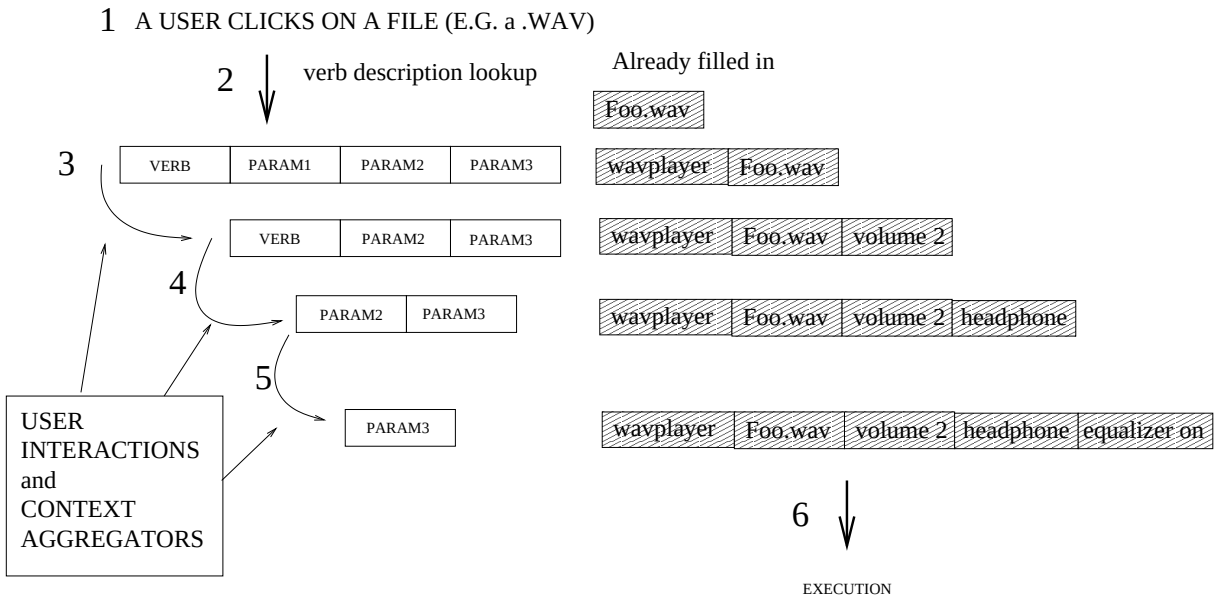

Figure 1: Example of execution loop

3.2 Prototyping CUROCO

We are currently writing a prototype of the architecture of CUROCO and the infrastructure it needs to work. This means mainly writing the slot description database and the mechanism to expose this description for the context aggregators to fill in. We will also write some context aggregators and description slots for some applications. As we need more applications and functionality, we will add the implementation of more components, rewriting whatever part of CUROCO limits us in this goal. The writing of CUROCO will be done using wherever possible simple prototypes to be able to get the feel of it and evaluate it at each stage of development.

3.3 Integrating CUROCO and Plan B

In the near future, CUROCO prototype will be integrated into the Plan B operating system using its PCM [5], becoming part of the ubiquitous operating system we are writing.

4. EVALUATION

Evaluation will take the same three steps as development. On each stage we plan to have working prototypes and experiment with different context inputs and uses in order to be able to determine its usefulness and try different alternatives. Some of this work has already done. We have found that even simple context data (presence, localization and state determined by heuristics and movement sensors) can be very useful and that exporting it through files is very portable and easy. See for example [2].

We plan to use CUROCO daily as part of the user interface for the Plan B [8] operating system. That way we will be able to see how difficult is to add new context sources and use it from applications whether they are context aware or not.

Using our prototype we will measure what effect the use of CUROCO has on the users, seeing if it makes their interaction with the user more comfortable and richer. We will measure also the extra load generated by using our architecture to make sure it doesn't slow the system too much or generate too many network traffic.

5. REFERENCES

[1] G. Chen and D. Kotz. Supporting Adaptive Ubiquitous Applications with the SOLAR System. Technical Report TR2001-397, Hanover, NH, 2001.

[2] E. S. Francisco J. Ballesteros, Gorka Guardiola and K. Leal. Traditional systems can work well for case study: Plan 9 from submitted for publication, 2004.

[3] C. K. Hess and R. H. Campbell. A Context File System for Ubiquitous Computing Environments. Technical report, University of Illinois at Urbana-Champaign, 2002.

[4] D. J.P.S. Aura: An architectural framework for user mobility in ubiquitous computing enviroments. *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, pages 23–31, 2002.

[5] G. G. M. Katia Leal Algara, Francisco J. Ballesteros and E. S. Salvador. Plan b's personal command module. commanding user activities in mobile and ubiquitous enviroments, *Submitted for publication*, 2004.

[6] C. Locke. Styx-on-a-brick, http://www.vitanuova.com/mkt/press/Styx-on-a-Brick.pdf.

[7] LSUB. Lsub web site http://lsub.org, 2001.

[8] LSUB. Plan b web site http://lsub.org/planb, 2001.

[9] LSUB. Lsub web site for localization http://lsub.org/who, 2004.

[10] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

[11] UPNP. Upnp web site http://www.upnp.org, 2001.