# Infrastructure Support for Contextual Applications - An Experience Report

Damián Arregui[1], Sophie Dupuy-Chessa[2],
Martin Muehlenbrock[1], and Jutta Willamowski[1]

[1] Xerox Research Centre Europe
Meylan, France
Firstname.Lastname@xrce.xerox.com
[2] Laboratoire CLIPS-IMAG
Grenoble, France
Sophie.Dupuy@imag.fr

**Abstract.** In this article we report our experience on building a contextual application with two different middlewares: the Context Toolkit from Georgia Tech and the Coordination Language Facility (CLF) from XRCE. In our email notifier sample application people receive audible or visual notifications about selected incoming emails at their current locations, e.g. in a colleague's room. Although rather straightforward, this application provides a good starting point for discussing requirements on software infrastructures for contextual computing.

## 1 Introduction

Context-aware computing [1], allowing an application to react to its actual context, is gaining increasing interest. Supposing that contextual computing requires appropriate support from the underlying infrastructure, we conducted a first experiment in order to help us identifying these requirements. We implemented a simple contextual application on top of two different middlewares, the Context Toolkit from Georgia Tech [2] and the Coordination Language Facility (CLF) from XRCE [3]. We chose the former because it is one of the few freely available context-aware platforms, and the latter because we wanted to evaluate our own in-house platform as support for contextual applications.

For the experiment we wanted to benefit from our existing environment, consisting essentially of iButtons[3] as directly available sensors. The system consists of sockets that are connected to desktop computers in a number of rooms, and coin-sized iButtons that are carried by the users and are plugged into an available socket when entering a room. We developed an email notifier application, which notifies users at their current location, e.g. in a colleague's room, about new and important email messages.

The idea of forwarding messages can already been found in early ubiquitous computing systems. Roy Want and his colleagues report that the most common usage of the active badge location system was by the receptionist who routinely used it when forwarding phone calls to the location of a recipient's current location [4]. More recently, a communication support system has been developed that uses location information that

[3] http://www.ibutton.com/

is available from the cellular phone operator to provide a dynamic phone pool with entries ranked according to the location information [5].

In the email notifier application, users specify relevant emails by means of criteria on its sender or subject. If such an email arrives and the system localises the user in another room through his hardware iButton, it sends a notification (sound and/or graphic) to the appropriated host. Therefore, the application first needs to gather sensor information as well as email data, then process this data and match it with user information such as preferences and filtering criteria, and finally, if necessary, trigger some user notification mechanism.

This project provided use with some conclusions on using middleware to support the development of an application that uses contextual information such as user location. In the next two sections we describe our experience with implementing the email notifier application on top of the Context Toolkit on one hand and the Coordination Language Facility on the other hand. Section 4 then exposes the main requirements we identified as a result of these implementations.

## 2 The Context Toolkit

### 2.1 Context Toolkit description

The Context Toolkit, developed at Georgia Tech [2], is a Java-based framework that facilitates the development and deployment of context-aware applications. It includes different building blocks for capturing, interpreting, and aggregating raw information stemming from sensors:

- *Context widgets* collect information from the environment through the use of software or hardware-based sensors. For instance, the `iButton widget` is a predefined context widget that sends a message to other components whenever an iButton is plugged into the sensor.
- *Interpreters* abstract away from raw context data into richer forms of information. A typical interpreter included in the context toolkit maps iButton identifiers to people's names.
- *Aggregators* collect context information related to the different entities in the environment (people, places, etc.). An aggregator can for example subscribe to several context widgets to receive information about a certain person.
- A *discoverer*, like a name service, provides information on locations and protocols of available components. This component is not available in the current version of the Context Toolkit[4]. This means that the structure of the application is rather static since no new sensors can be added at runtime.

These components communicate over a network to compute a contextual state. In general, the Context Toolkit provides processing of contextual information in the following manner: whenever a component receives a message from another component, it extracts the information from the message, processes the information in some way (this has to be implemented by the system developer), and sends one or more messages to other components.

[4] http://www.cc.gatech.edu/fce/contexttoolkit/

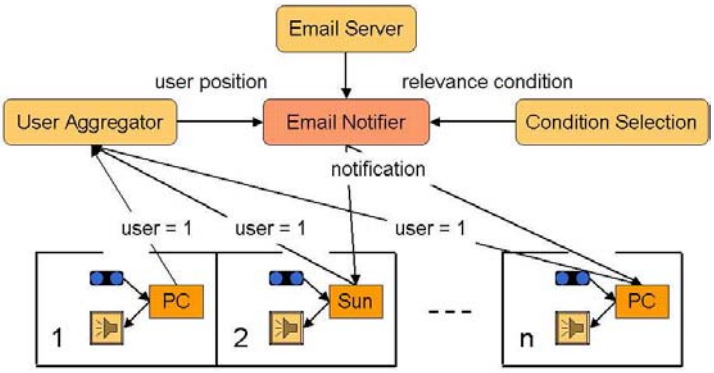


**Fig. 1.** Email Notifier Architecture in the Context Toolkit.

### 2.2 Implementation

For the email notifier we used and extended some of the pre-defined components and added some new ones. In our application, the context consists of the users' locations and the arrival of new email (see figure 1). This information is captured by two different widgets for a real world sensor and a service sensor, `WpersonNamePresence` and `WEmailDetection`, respectively.

Each `WpersonNamePresence` widget is connected to an iButton sensor and provides information about the presence of users (when having their iButton plugged in). An interpreter `IButton2Name` maps identifiers to user names.

The `WEmailDetection` is a new widget that automatically scans a mail server for incoming email. There is one widget for each user, connected to a user interface `EmailUI` that allows for specifying user related data (a simple user profile) such as a user name, a password, and a mailbox to scan as well as to define conditions on the sender or subject of an email to trigger the user notification. There is no generalized way in the Context Toolkit to define and check distributed conditions on contextual states. Here, it has been implemented by adding a listener to the thread that monitors the mailbox and by checking the conditions by means of ordinary if-then statements.

A pair of these widgets is running on each of the users' computers. In addition, for each user an aggregator `Suser` collects information on the user from all `WEmailDetection` widgets. Due to the limitation described above, no new users can be added at runtime. In case an important email arrival is detected by `WEmailDetection`, a message is sent to the component `PersonNamePresenceAgg`, which requests the current user's location from `Suser` and forwards the message to the appropriate widget.

### 2.3 Lessons Learned

The Context Toolkit represents a good starting point for doing your own contextual application by identifying components such as widgets, interpreters, and aggregators. It provides some concepts to elegantly design contextual applications. The architecture is rather easy to understand, so new developers can quickly build contextual applications. However, concrete support to developers is sometimes insufficient:

- Interpreters provide currently only very simple mappings, e.g. linking iButton identifier to user name. It would be interesting to develop them in order to integrate more complex reasoning like machine learning techniques.
- There is no high level abstraction for situations. For instance, the situation where a user is docked somewhere and receives an email cannot be expressed easily. Application logic and context interpretation must be encoded in Java. So they cannot be expressed by end users or dynamically changed by developers.
- Even if the toolkit supports distributed components and their interaction through conditions placed on the subscription or query request, it lacks a discovery service and support for distributed application setup and monitoring. In particular, new components cannot be added at runtime.

In conclusion, the Context Toolkit currently seems mainly targeted at small applications in rather static settings. This accords with findings elsewhere [6]: “The framework and toolkit is an elegant way to design and implement context-aware applications for simple and highly routine contextual situations.”

## 3 The Coordination Language Facility

### 3.1 CLF Description

CLF [3] is a middleware platform aimed at coordinating distributed active software components over a Wide Area Network[5]. Mekano complements the CLF platform with a library of reusable coarse grain components and component development tools. CLF/Mekano have been used in the implementation of various distributed applications [7] deployed across multiple intranets.

CLF relies on the resource-based programming paradigm [8, 9]. It models objects as *resource managers*, and the interactions between them as *transactional resource manipulations*. The *resources* managed by CLF objects are tuples of strings, each string element containing either text, string-encoded, or marshalled objects. They may represent events, data records, service offers, etc.

To manipulate their resources, CLF objects offer two kinds of interfaces: *direct methods* and *services*. Direct methods implement traditional remote method invocations on *single objects* while services allow to coordinate the manipulation of resources across *multiple objects*. The coherence with respect to the access and manipulation of resources across a set of object services is achieved through the CLF protocol, This

[5] Available under the name of STITCH at http://www.alphaave.com/

protocol, deployed on top of the object services, consists of three phases, *negotiation*, *performance*, and *notification*.

The *negotiation phase*, queries each service for resources matching a given filter. On such a request, the service returns a potentially infinite stream of offers identified by unique `actionIds`. The *performance phase*, unrolls a classical two-phase commit protocol over the set of services, ensuring the atomic execution of a selected set of actions obtained during the negotiation phase. To achieve atomicity, it first attempts to reserve all the resources corresponding to the selected `actionIds`. If successful, it enacts all actions. Otherwise, if any reservation fails, it cancels all previously executed reservations. Finally, the *notification phase* allows for asynchronous creation of new resources.

CLF provides a scripting language exploiting the CLF object model and services through the above described protocol. It views coordination as a complex block of inter-related manipulations of resources held by a set of objects. CLF scripts describe, through rules, the expected global behavior of such blocks in terms of resulting resource manipulations. But they abstract away from the detailed sequencing of invocations of the CLF interaction verbs required to achieve this behavior. This abstraction feature considerably simplifies the design and verification of coordination scripts.

In a CLF application, dedicated CLF objects called *coordinators* enact the coordination scripts. As any CLF object, coordinators manage resources representing CLF coordination scripts and the rules which compose them. The possibility to dynamically modify the set of rules held by a coordinator allows to dynamically adapt the behavior of an application to contextual needs.

### 3.2 Implementation

It turned out to be straightforward to translate the application scenario described in section 1 into resource manipulations. We had to figure out which would be the components involved in the application and which services they should provide, and to write the appropiate coordination script to wire them together. The types of the application components and their associated services are:

- `Workstation`: Each machine runs one Workstation component connected to the application, providing the `IsButtonDocked` service for the local iButton dock, and the `Popup` service to show a window on the local display.
- `IMAP`: This component encapsulates the IMAP [10] e-mail server, and provides filtered access to the users' mailboxes through the `filteredMail` service.
- `DataCenter`: This component holds some application specific data, i.e. the mail filters and iButton identifier associated to each user, the notification tickets, and the list of registered machines. These are resp. accessible through the `UserFilter`, `UserButton`, `Notification` and `YellowPages` services.

Besides, two infrastructural components are used:

- `NameServer`: The NameServer allows to dynamically resolve service names into network addresses.

– `Coordinator`: The coordinator enacts the coordination scripts representing the application logic (more details below).

In fact, two CLF rules[6] are enough to express the desired behavior:

```
‘userFilter(userId, IMAPfilter) @
‘filteredMail(userId, IMAPfilter, mailId, from, to, subject) @
buildMsg(from, to, subject, msg)
<>- notification(userId, msg)

notification(userId, msg) @ ‘userButton(userId, buttonId) @
‘yellowPages(machine) @
‘isDocked(machine, ’IsButtonDocked’, buttonId, ’docked’)
<>- popup(machine, ’Popup’, msg)
```

The first rule retrieves regularly, for each user, the new incoming e-mails matching a particular filter and builds a notification message. It inserts a resource made up of this message and the user’s id in the `Notification` service.

Each notification resource triggers in the second rule for each registered machine a check whether the users iButton is docked there. In that case it inserts a new resource in the `Popup` service of the machine, which results in showing a notification window on the corresponding display.

### 3.3 Lessons Learned

The CLF middleware targets distributed applications in a general sense, but it seems to match well with the contextual aspects introduced in our scenario. Indeed, it provides a number of features that facilitate the implementation:

– The resource-based paradigm allows to uniformly encapsulate a very diverse set of components, contextual or not. So CLF does not provide a specific help (like interpreters or aggregators) to build contextual components; but their development is similar to any other one.
– The negotiation phase of the CLF protocol allows to capture the asynchronous and reactive nature of a contextual application.
– The performance phase of the CLF protocol supports the need to coherently verify distributed contextual conditions.
– The scripting facility allows to easily define, modify and reuse the application logic, hiding at the same time the complexity of the CLF protocol.
– The built-in component library and tools for application deployment and monitoring ensure a comfortable development environment.

Finally, it is important to stress the fact that the full application was running in only a week’s time. Most of the work went into encapsulating the external entities, namely the IMAP server and the iButton dockstation: as we were already familiar with CLF

[6] Output parameters are underlined. The backquote character indicates that the resource should not be consumed during the performance phase.

we didn't have to pay the price of learning the resource-based programming paradigm. Developers who are new to CLF should take the time to grasp this paradigm, which is essential to be proficient at building components and weaving them together with the scripting language.

Moreover if we can expect developers to learn writing application logic with CLF rules, these rules are not adapted to end users. End users cannot specify the situations where they want the system to react. This would require a simplified or graphical version of the CLF scripting language.

However, real-life applications would certainly be much more complex than our email notifier. People interact with increasingly diverse hardware which is often mobile itself. Contextual data provided by sensors is abundant but noisy, and of many different kinds (e.g. temperature, sound, movement). Making sense of all these inputs will require an infrastructure capable of advanced multi-layered reasoning. Such issues are further discussed below.

## 4 Where to Go from There

Having reported on our experience, in this section we point out some of the issues that still need to be addressed in a software infrastructure supporting the development of contextual, mobile and device-based applications.

### 4.1 "Standard" Aspects of Infrastructures

Contextual applications benefit from a number of features common to distributed computing platforms such as reliability, scalability, manageability, re-usability, etc. But they are particularly demanding in the areas of privacy, reactivity and adaptability.

Privacy turns out to be the outstanding issue if contextual applications will ever be actually deployed in real-world settings. Some kind of generic access control mechanism is needed to protect each user's private information. The right balance between granularity and complexity has to be found to produce a system both flexible and usable. A privacy context could be introduced into existing models. For instance, do not notify me about any new personal email when I am in my boss' office, or hide my calendar data from all applications but the corporation's meeting scheduling server.

Furthermore, contextual systems have to cope with fast-evolving inputs. Thus they have to take into account the limited lifetime of data (sensed or inferred) and to react accordingly. For example in the case of a contextual city guide, I want my personal profile, interaction history and current location to be aggregated with the central tourist guide database to provide me with timely indications about interesting sights to visit. For a system with tens of thousands of widely distributed users this requires an infrastructure capable of delivering the required efficiency, probably involving hierarchical structures with some level of replication.

In fact, being able to cope with as many unforeseen situations as possible is probably the strongest requirement for the underlying infrastructure. At the lowest level, this implies acting as an interoperability layer, hence providing a rich library of components able to interact with various hardware devices. Moreover, some components may

be mobile, and should thus be able to dynamically join and leave the system. In our email notifier application for example, new workstations and iButtons should be able to seamlessly enter and leave the application. To support this the infrastructure needs appropriate lookup and discovery mechanisms, similar to those provided by Jini [11]. Once the different components are aware of each other, the infrastructure should provide the means to bring these ad-hoc networks to life by defining collaborative behaviors as discussed in the following section.

### 4.2 Manipulating Contextual Data

Another issue for contextual applications is how to support the intelligent processing of contextual data at the infrastructure level. Indeed at this level raw data needs to be collected, filtered, aggregated and interpreted in order to trigger useful actions automatically, without user intervention. In fact, events often appropriately represent contextual data. In parallel with our experiment, we therefore enhanced CLF with an event model allowing to quickly and easily integrate context sensors or devices as event producers and/or consumers [19].

Aggregators and interpreters as proposed by the Context Toolkit are a step towards processing contextual data, but there is a need for higher abstraction levels. The Cybreminder [12] prototype extends the Context Toolkit in this sense, providing some support for defining “situations” as conjunctions of elementary “sub-situations”, constituted by constraints on contextual data delivered by widgets and aggregators. The Cybreminder prototype thus supports rule-based reminders. But there is no discussion on how to extend this situational approach to other applications, making it a feature of the infrastructure. As far as we know, the Context Toolkit does not integrate these results up to now ; hence, in our application, we had to encode the email notifier logic inside a Java class, a solution lacking suitable dynamicity and abstraction levels.

Concerning CLF, the scripting feature reminds the rule-based definition of situational reminders in Cybreminder: it allows to declaratively specify the behavior of an application, as shown in section 3. However, CLF was designed from a slightly different perspective: providing a generic tool for application developers to express arbitrarily complex interactions between distributed components. It allows for example to coordinate context sensors and more traditional services like databases or document transformers, which broadens the horizon of possible applications. It even gives the possibility to reflexively control the behavior of the middleware itself [13] to perform contextual adaptation.

Lastly, the uncertainty of data appears as an open issue. Indeed sensors provide data with varying confidence levels. This has to be taken into account when reasoning on these data to infer higher level contexts. Therefore a probabilistic model could be integrated in the approaches discussed earlier in this section. If we consider the rule-base scripting in CLF, a probability level could be assigned by each service to the resources it holds. Each resource inserted on the right hand side of a rule could be tagged with a probability value calculated by some function (e.g. product, average) of the probability values associated to the resources on the left hand side of the rule. Then the new resource could trigger an action depending on its probability value, for example only if this value exceeds a preset threshold.

Related work tackles the issue of manipulating contextual data. [14] proposes a contextual extension of the context widget which models relation between contextual data and identifies six basic operations on contextual data. [15] proposes an architecture where sensor data are abstracted by the concept of cues (e.g. average, standard deviation) and the current situation is derived from cues. The authors also evoke scripting as the top layer of the architecture. In SOLAR [16] the collected events flow through a graph of operators (i.e. filter, merge, aggregate and transform). In [17] the authors propose to use automated path creation to dynamically combine a set of operators. [18] presents a location service that interprets location sensed data into a location-technology independent format by representing location information in terms of regions and interactions between regions.

These proposals, much like the CLF scripting language, clearly target application developers. It remains a challenge to build appropriate tools to empower end-users for working with context. We are currently exploring the possibilities of a graphical interface which, even if it will not provide all the features of the scripting language, may allow end-users to taylor contextual applications to their own needs.

## 5 Conclusion

In this article we reported our experience with using two different-purpose infrastructures to implement the same contextual email notifier application: the Context Toolkit as dedicated contextual platform, and the CLF as general purpose distributed component coordination framework. Our goals were to gain a first expertise in developing contextual applications on one hand, and to better understand requirements on software infrastructures for contextual computing on the other hand. In conclusion, we want to emphasize two points. First, as contextual applications are heterogeneous and distributed by nature, their implementation *requires efficient interoperability and distribution support*. Second, context processing becomes quickly complex and *requires high level abstractions*, like situations.

Concerning the first point our experiment seems to show that currently available contextual platforms, or at least the Context Toolkit, lack some basic features which come for free in generic distributed computing platforms. Concerning the second point, both the Context Toolkit through Cybreminder, and the CLF through its scripting language adopt a rule-based approach which allows to declaratively specify behavior. Rules seem appropriate to easily express conditions on contextual data, generate intermediate context, and finally trigger the required application behavior.

Overall, adapting and enriching a general purpose platform like CLF with contextual features seems feasible and appropriate. According to the requirements introduced in section 4, the CLF middleware provides a good support for interoperability and distribution, even if it can be improved for look-up and discovery mechanisms. It also proposes a high level of abstraction to manipulate contextual data through its scripting rules. Of course they must be adapted for fast-evolving and uncertain data. We still need to refine our understanding of how a “generic” platform like CLF must be adapted to also tackle other problems of particular interest in contextual computing. Efficient solutions to issues such as privacy and data uncertainty remain to be found.

## References


1. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, Computer Science (2000)
2. Dey, A.K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology (2000)
3. Andreoli, J.M., Arregui, D., Pacull, F., Riviere, M., Vion-Dury, J.Y., Willamowski, J.: CLF/Mekano: a framework for building virtual-enterprise applications. In: Proc. of EDOC'99, Mannheim, Germany (1999)
4. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The active badge location system. ACM Transactions on Information Systems **10** (1992) 91–102
5. Nakanishi, Y., Takahashi, K., Tsuji, T., Hakozaki, K.: iCAMS: A mobile communication tool using location and schedule information. In Mattern, F., Naghsineh, M., eds.: Pervasive 2002, Berlin, Springer (2002) 239–252
6. Greenberg, S.: Context as a dynamic construct. Human-Computer Interaction **16** (2001)
7. Arregui, D., Pacull, F., Willamowski, J.: Yaka: Document notification and delivery across heterogeneous document repositories. In: Proc. of CRIWG'01, Darmstadt, Germany (2001)
8. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems" (1985)
9. Andreoli, J.M., Arregui, D., Pacull, F., Willamowski, J.: Resource-based scripting to stitch distributed components. In: Proc. of ECIS 2002, Beijing, China (2002)
10. Crispin, M.: Internet Message Access Protocol. RFC 2060, IETF (1996)
11. Arnold, K., Wollrath, A., O'Sullivan, B., Scheifler, R., Waldo, J.: The Jini specification. Addison-Wesley, Reading, MA, USA (1999)
12. Dey, A.K., Abowd, G.D.: Cybreminder: A context-aware system for supporting reminders. In: HUC. (2000)
13. Arregui, D., Pacull, F., Willamowski, J.: Rule-based transactional object migration over a reflective middleware. In: Proc. of Middleware 2001, Heidelberg, Germany (2001)
14. Coutaz, J., Rey, G.: Foundations for a theory of Contextors. In: Proc. of the 4th International Conference on Computer-Aided Design of User Interfaces - CADUI'2002, Valenciennes, France, Kluwer Academics (2002)
15. Schmidt, A., Laerhoven, K.V.: How to build smart appliances? IEEE Personal Communications **8** (2001)
16. Chen, G., Kotz, D.: Supporting adaptive ubiquitous applications with the SOLAR system. Technical Report TR2001-397, Dartmouth College, Computer Science (2001)
17. Hong, J.I., Landay, J.A.: An infrastructure approach to context-aware computing. Human-Computer Interaction **16** (2001)
18. Naguib, H., Coulouris, G.: Location Information Management. In: Proc. of UbiComp'2001. Number 2201 in LNCS, Atlanta, USA, Springer Verlag (2001)
19. Arregui, D., Fernstrom, C., Pacull, F., Rondeau, G., Willamowski, J.: Stitch: Middleware for ubiquitous applications. In: Proceedings of Smart Objects Conference 2003, Grenoble, France (2003)