On Design Framework of Context Aware Embedded Systems

Abhay Daftari Nehal Mehta Shubhanan Bakre Xian-He Sun

Department of Computer Science
Illinois Institute of Technology
{daftabh, mehtneh, shubh, sun}@iit.edu

Abstract

The primary goal of embedded systems is "Human-centered computing," that is providing service anywhere, anytime, and automatically. While electrical devices become smaller and smaller and more powerful, context awareness becomes more and more important for embedded systems. Although some of the later systems have been developed with context awareness in mind, how to design a context aware embedded system systematically is still an issue that eludes researchers in the field. This study introduces the importance of context awareness in today's embedded systems, divides the design of context aware systems into context aware applications and infrastructure. It further applies aspect orientation in the design of context aware infrastructure to model the architectural/system from the developer's point of view. We show that applying aspect orientation in the development of context aware embedded systems is feasible and has real potential.

Keywords

Context awareness, embedded systems, aspect oriented software development, context aware infrastructure, human-centered computing.

1. Introduction

The Early 1990s saw many efforts towards reducing the size of the computer and its portability. Small but powerful devices become widely available that can be embedded anywhere and can be carried by the user wherever he goes. During the same period, growth in the wireless technology gave rise the popularity of coordinating small powerful devices to form an embedded computing environment for the "Human-centered computing." The size and the power of the computing devices are no longer the determining factors in the design of embedded systems, but the quality of service of the "Human-centered computing" is a key factor. Context awareness is becoming an important factor in the embedded system design.

The vision of mobile computing [1] is to provide some form of freedom to the end user by providing computing support when the user is in mobile mode. Mobility plus the availability of embedded systems, or so-called "smart spaces" constitute pervasive computing, a broader view of providing 'Human Centered computing'. Pervasive computing talks about providing computing everywhere and at all times [1]. This is done through providing smartness into embedded devices and systems, and providing balance between assistance and interference to the end user [2].

It is clear that understanding the user's surroundings will play a significant role in realizing the goals of Pervasive computing as well as embedded systems. Context awareness is directed effort towards understanding the environment around the user with the help of smart devices embedded within its environment and improve the end-user's experience by using this knowledge. Hence, context awareness is the key research area in embedded systems. It has been explained what constitutes a context aware application, and what context is and what are the definitions and categories of context and context-aware (CA)[3].

To understand what context awareness is, first consider the traditional classroom scenario:

> Professor T informs students about the updated course website that contains lecture slides for the day's lecture and that they need to bring the slides in the class for better understanding as professor T is going to use projector for presentation. Even after receiving this notification, some of the students either did not read the notification or some of them forgot about it before the class. Hence, class is conducted with some students have the slides in front of them while some of them do not.

Now, consider another scenario of smart classroom environment, where:

> If professor T is moving towards the projector and lights in the room are off, then the environment pervasively transfers the presentation slides from the professor's handheld device to students' handheld device and the projector starts the presentation.

Discussion:

> The second scenario takes into account the environment context information in which it is executed. In the above example, the context information is the location of the professor (classroom), the state of the lights in the room (off), the activity of the professor (moving towards the projector), and the number of students in the classroom. Although the same set of information was available in the traditional classroom scenario, it was not utilized towards better end user experience. This type of utilization is in line with the vision of context aware computing. In the later sections, we will base our discussions upon the above smart classroom scenario.
>
> There have been efforts in achieving context awareness through various approaches. The major goal of these approaches is to capture surrounding context and adapt it as per end-user needs. But still, context awareness is in inception stage and current approaches provide limited context information. As anticipated [1], future context aware approaches expected to support variety of context information and in large number. Hence, these systems should be able to meet the requirements of scalability and ability to evolve to fulfill the future needs.

Aspect Orientation (AO) is a relatively new methodology for Software Development (SD) [4] that promises better design leading to properties including scalability and ability to evolve. This study analyzes context-awareness from the perspective of applying AO in context aware system/infrastructure design.

2. Aspect Oriented Software Development

A requirement is analyzed and some design is made which then is implemented. But during the process, implementation is based on design documents that are in turn based on requirement documents. Most of the times, it is difficult to trace requirements into the design and then in the implementations. Modification in any one of these – requirements, design or implementation - will require propagating it into all of them, which is complicated and causes problems. The main reason behind this is that the system is in different forms at different levels.

The need for dealing with issues 'one at a time,' was coined as the principle of 'separation of concerns' [5]. Aspect Orientation Software Development (AOSD) is relatively a new software development methodology, based on principle of separation of concerns, for achieving a number of desired properties in a system such as extensibility, modularity, etc.

A 'concern' is a functional or non-functional property of a system like security, synchronization, logging, etc. During expression of design into implementation, due to the nature of some concerns and/or limitations in the expression techniques used, it is not implemented in a modular way and it tends to crosscut the rest of the implementation causing code tangling. Such a concern is called crosscutting concern. An aspect, on the other hand, is modular realization of a crosscutting concern. After incorporating aspects within the system, they need to interact with each other to form an operational system. The process of achieving this interaction is termed as weaving. An aspect consists of two parts, the first part – called advice – that contains the functional details of concern, and the other part – called pointcut – which identifies the points of interaction with the rest of the system. A joinpoint is a principled point in the execution of a program; for example, a method call is one form of joinpoint. A pointcut, defined above, can be imagined as a collection of such joinpoints.

3. CA Infrastructure requirements

Brief list of projects towards context awareness is Aura (CMU)[8], Oxygen (MIT)[1], RCSM (Re-configurable Context Sensitive Middleware, ASU)[9], GAIA (UIUC)[10], Context Toolkit (Georgia Tech)[11], SOLAR (Dartmouth)[12], Rome (Trigger Based Context awareness Infrastructure, Stanford)[13], Rapidware (Michigan State University)[14], Multi-Sensor context Aware Clothing (Lancaster, UK)[15], Charade (Gesture Recognition, University of Paris, France)[16], One.World (University of Washington, Seattle)[17].

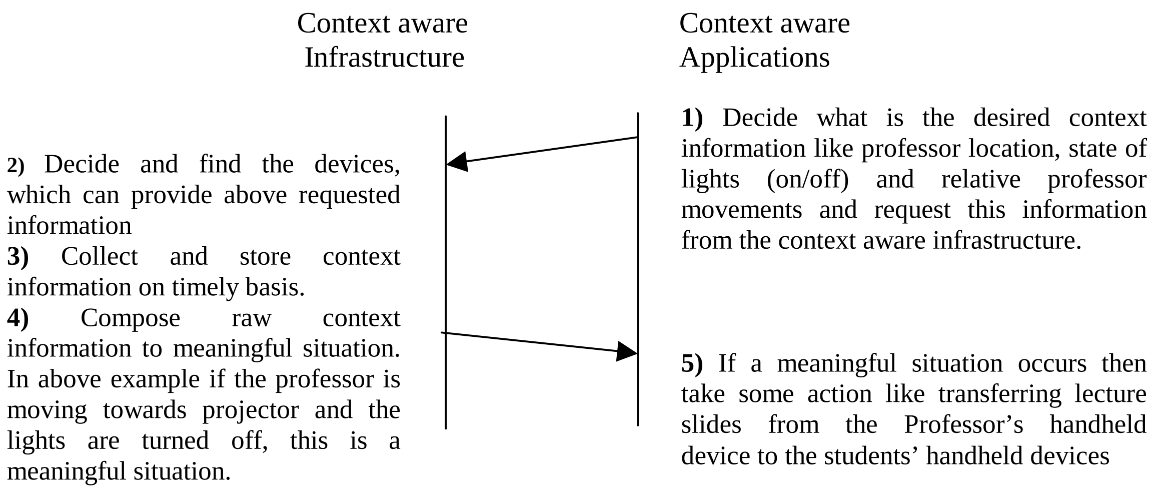
Based on the analysis of above system, we have observed that preliminary approach was to just provide context awareness for one or two context that too embedded into platform. This was platform specific and not sufficient for any complex context awareness.

The improvement over the past approach identified the need for many types of context and the design of standalone applications. Going back to our old example of Smart Classroom, in order to provide a smart environment a number of activities are needed including:

1. Decide what the desired context information is, like professor location, state of light (on/off) and relative professor movements.
2. Decide and find the devices, which can provide above context information
3. Collect and store the above context information on a timely basis.
4. Compose raw context information to meaningful situations. In the above example if the professor is moving towards the projector and the lights are turned off, this is a meaningful situation.
5. If a meaningful situation occurs then action should be taken like transferring the lecture slides from the Professor’s handheld device to the students’ handheld devices.

It can be observed that the context aware applications themselves would have to deal with activities like communicating with context sources, collecting context data, storing and managing context data for future, and finally using and adapting context data as per user’s needs. The approach of storing context data on individual basis resulted in data redundancy. Moreover, to design a similar system, all the steps need to be redone from the beginning. So the issues in terms of extensibility and ability to evolve to future needs of context aware system needed to be addressed. Learning from this approach, the current research in context awareness took an architectural approach to towards system design, in which the total responsibilities are divided between CA infrastructure and CA applications. The context aware infrastructures are responsible for collecting, managing and delivering context on behalf of CA applications, whereas the CA applications are responsible for adaptation.

In the above example, such a division would be:



Context aware Infrastructure

Context aware Applications

1) Decide what is the desired context information like professor location, state of lights (on/off) and relative professor movements and request this information from the context aware infrastructure.

2) Decide and find the devices, which can provide above requested information

3) Collect and store context information on timely basis.

4) Compose raw context information to meaningful situation. In above example if the professor is moving towards projector and the lights are turned off, this is a meaningful situation.

5) If a meaningful situation occurs then take some action like transferring lecture slides from the Professor's handheld device to the students' handheld devices

Responsibilities in steps 2, 3 and 4 lie with the infrastructure and those in steps 1 and 5 should lie with the application. Applications only need to describe the required context information and request context aware infrastructure to collect this information on its behalf. The responsibilities of finding the sources for the required context, and dealing with context source specific details are given to context aware architecture. This approach will help the application designer to focus on designing adaptation behavior of context aware application and provide freedom from dealing with unnecessary source specific details. On the other hand, similar approach can be used for many other context aware applications because the infrastructure for collection and delivery is already in place, making architecture scalable to many context aware applications. So, a context aware computing environment should consist of two parts, the system infrastructure and the application software. The CA system is responsible for collecting, storing - managing and delivering it to context aware applications, and the CA applications are responsible for adapting it.

Based on the separation of infrastructure and application principle and based on the analysis of existing works, we have identified the following goals that any context aware system should fulfill:

Goals:

- Collect context on behalf of application
- Provide ways to subscribe and deliver collected context to respective applications
- Store and manage past context for future needs
- Mask Heterogeneity and provide independence in terms of programming languages used, types of system support.
- Systems should be modular, and allow only required components to load.
- Systems should be extensible to future context need
- Systems should be scalable to many context types and many context aware applications
- Components can be reusable for future context need

Context aware is still a relatively new concept. Many of the proposed context-aware systems are the result of convergence of independent projects targeted at realizing different sets of scenarios in pervasive computing. For example, CODA and Odyssey file system – although in the beginning they were designed for mobile data access, and eventually converged to provide basic form of context awareness into Project Aura.

The abstract goals can be transformed into two types of requirements for system design: (1) functional requirements that support the core functionality of system, and (2) non-functional requirements for orthogonal system needs like extensibility and scalability etc. Functional requirements are those that are specific to a CA system whereas, non-functional are orthogonal requirements that are generally true for any system. Non-functional requirements are also known as quality requirements.

3.1 Functional Requirements

Following are the identified functional requirements for a design of a context-aware system:

1. Context Collection: This deals with collection of data from the sensors. This involves dealing with questions about the data model to be used for collection (Push / Pull / Request / Reponses), Insertion / Removal of Context Providers.

2. Context Storage/Management: This requirement pertains to the storage of the collected context data and its management. The storage of context data is significant for a number of purposes such as reproduction of context, logging, mining and future predictions. Also, context data needs to be communicated at various places such as from sensors to storage, from storage to consumers and most probably proxy consumers. Often the data is time sensitive. Hence when, where and how to move the data, needs to be managed.

3. Context Subscription/Delivery: There needs to be ways for consumers to acquire the collected and stored context data. The System thus needs context Subscription methods, and a model for distribution (Push / Pull / Request / Reponses).

4. Context Analysis and Composition Ability: This is the most significant functional requirement for context aware systems. Looking at various definitions, one can derive that a context could be a composition of multiple raw data provided by sensors. And not all the permutations of raw data will result in sensible contexts. Hence, the task of context-composition is complex and is believed to involve decision-making on the basis of history and experience.

3.2 Non-Functional Requirements

1. Quality of Service: Due to the dynamic nature of resources like sensors, devices, and network bandwidth, it becomes essential for context-aware systems to provide the ability to the applications for specifying such resource constraints.

2. Security: As with any other system, security is one of the prime non-functional requirements. This requirement arises from the basic question - To what extent should the private information be exposed to context-aware services transparently to the user? Also, how should the issues like access control, authentication, authorization, and data encryption be addressed in such an environment?

3. Heterogeneity: Heterogeneity arises due to different hardware platforms and also due to the varying capabilities of the devices used. For achieving portability of applications across multiple platforms, it is necessary to fulfill this requirement.

4. Scalability: The Ad-hoc discovery of resources, the changes in the number of users and resources, and the limited computing capabilities of devices, introduces the problem of scalability.

5. Adaptability: In pervasive computing when a user switches from a resource-rich device to a resource-strapped device, the applications either should be able to adapt to these changes to provide seamless service, or alternatively scale down according to the new surroundings.

6. Fault Tolerance: The Ability of the system to adapt to compensate for errors can be termed as fault tolerance. Doing things right even if pre-conditions deviate to a limited extent is especially desired in CA systems.

7. Extensibility: Evolution is a part of the software life cycle. In order to provide the support for new features, extensibility at different levels is necessary. For example, in a context-aware system context should be extensible to new context and composition.

8. Functional Modularity: There should be clear distribution of responsibilities between the devices, the applications, the infrastructure and the components within this infrastructure. All the components of the system should be developed in an independent manner. This requirement arises out of the principle of separation of concerns that inherently brings the number of advantages to the system such as comprehensibility, reusability, re-configurability, and other such properties.

9. Mobility: Pervasive computing is synonymous with user-centric computing. Unlike traditional computing that forces the user to follow computing and the data, pervasive computing focuses on providing the computing and data whenever and wherever the user needs it. This introduces mobility constraints on context-aware applications.

Discussion:
Functional requirements present the core functionality of the architecture and are necessary to fulfill the need of the context aware application support. For a better understanding, consider the ‘smart classroom environment’ example scenario. Once the application has decided the types of context information such as the information about the location, the state of the lights, and the movement of the professor, it describes these context needs to the context aware architecture.

1. Context Collection:
   The infrastructure decides what context providers to use to get the context information. Once they are known, it will collect the information from them.

2. Context Storage/Management: The collected context data for the location and the status of the light are stored for future use. Although this data is not currently used by ‘Smart classroom environment’, it could be utilized in the future. It is not necessary that all applications need it, but as context awareness is an evolving area, this data will be useful to predict the system’s behavior on the basis of past history.

3. Context Subscription/Delivery: In the beginning, the application will describe the context needs. When some specific situation occurs, like ‘presentation is going to start,’ the application will be notified for the same.

4. Context Analysis and Composition Ability: This function accumulates many raw contexts information like ‘light is off and professor is moving towards projector,’ then decides that the required situation ‘presentation is going to start’ is generated.

Here, we showed that the above functional requirements are necessary and sufficient for any context aware architecture to support context aware application.

4. Applying AO to CA Infrastructure

The above section has mapped the goals into functional and non-functional requirements. Based on the principle of separation of concerns, AOSD tries to modularize the requirements into the aspects. Once the aspects are identified and their weaving accomplished, one could easily trace the requirements to the design and from the design to the implementations. Many approaches within AOSD understand the system in terms of the core functional components and the aspectual components that constitute the non-functional components [19], whereas, others understand it in terms of the aspectual components composed of both the functional and the non-functional components [20]. But the major issue of identifying and distributing concerns into core and/or aspects still remains an open issue.

Before discussing application of AO in Context Awareness, we mention few research projects that have understood the potential of AO and acknowledged it by applying it to their system. Some of the areas are operating system [24], middleware [25], etc. It can be observed that all these areas have crosscutting concerns and code-tangling phenomenon. It has been demonstrated AO has helped in separating these concerns and achieved better non-functional goals. As can be observed from the discussion above regarding requirements of context aware system, there is good extent of crosscutting concerns in CA system that makes it prospective candidate for application of AO.

We will walk you through an example by keeping context awareness in focus and show how the application of AO will result in better design. To analyze the application of AO in context awareness, let us focus on the design and implementation of one of the functional requirements – 'Context Collection'.

> To design a context collection module to support number of context providers. Currently the system needs to support only a few number of context providers, but this number could increase in future. The location of context providers can possibly be local and/or remote thereby involving communication over the network.

Considering the functionality needed for context collection, the pseudo code for the context collection module can be represented as follows:

Looking at the requirements the concerns in the system are not clearly visible. At first instance it looks that entire functionality is related to context collection. Further examination clarifies that the functionality of context collection is tangled with the policies related to scalability. Following are the part of functionality that refer to scalability,

1. What should be the frequency at which the context data is collected from the providers? – Context Collection Frequency.

```
/* Start up work */
Set some default value for context collection frequency.
Initialize the context location store
Find the network latency and how fast then can switch between connections

/* Loop for new context providers */
Loop
        Look for new context provider
        If found any,
                Get location information
                Store in the location store for future need
        Get the context collection frequency information
        …….
Collect context from all registered context providers periodically.
/* end of loop*/

Context collector is shutting down.
```

2. What are the capabilities of individual providers in terms of operating frequency? – Context Provider Details.
3. What are the communication constraints such as bandwidth availability, network protocols, etc.? – Other Dependencies.

Ideally, the modifications or changes in the above policies should be independent to the context collection functionality. This becomes crucial when there is change in the number of context providers of the system or in the network bandwidth, for example. As these kind of changes will require modifications in the policies that are scattered in the number of functional components of the system. But, in the above design of context collection module establishing such independence is difficult to achieve and sometimes impossible due to the tight coupling between the functional and non-functional requirements. Following section explains how AO will make above design better and makes to achieve better scalability.

When aspect oriented technique is used, emphasis is given to the separation of concerns such as the basic context-collection concern and scalability concern that relates to a number of variables mentioned above. Following figure shows what would be the implementation of solution of above issues using an AO approach.

```
class ContextCollector {
        public void collect() {//Collect the context Information}
        public void send(){//send the collected information}}

aspect Scalability {
        pointcut repeat() : call(ContextCollector.collect());
        pointcut send() : call(ContextCollector.send(//context information));
        before() : repeat () {
                //Decide the frequency for context collection
                        //and introduce the loop   }
        after () : repeat () {
                //end the loop  }
        before () : send () {
                //Get the network parameter and adapt accordingly }}
```

In the above pseudo-code for context collection, using AO, the developer is oblivious to any other concerns while implementing of context collection functionality. The two concerns for context collection and scalability are clearly understood and realized as two separate modules.

Discussion:
In the AO technique, concerns are separated and modularized in one place with the specification of the interaction between these separated modules. These modules (aspects) are then weaved together to form a complete system. The functional code for context collection is completely free from the code for scalability due to separation achieved. Moreover, if the scalability policies change in the future, as it will be modularized at one place, its replacement will be multiple times easier due to reuse of weaving code as a result of the properties of quantification and obliviousness inherent in the system. Meaning, the need to visit the places of crosscut and modify the code will be eliminated.

With conventional implementation techniques, such as procedural or object oriented programming, it is clear from [19][21][22][23] that if a requirement like 'need to add new context provider,' is not adequately addressed in the design phase then it is difficult to incorporate it later on in the life cycle of the software system. On the other hand, with AO, requirements would be easily traceable into design in terms of aspects and also in the implementation, as aspects are weaved by a weaver to generate the implementation of the design. Hence incorporating modifications in existing requirements and also adding new requirements using AO will result better design due to added flexibility, extensibility and ability for evolution. Thus, following needs to be done in order to add a new context provider that needs to use existing authentication policies, while using AO in the system: With AO, the existing authentication policies is modularized at one place and it is 'inserted' into appropriate places within the functional code through a standardized weaving mechanism similar to the logging example presented in the introduction of AOSD.

The above discussion is presented as a thought-provoking concept. Similar discussion holds true for applying AO to fulfillment of other non-functional requirements of a CA system.

In order to see if AO can help towards improvement of existing CA systems, one needs to analyze their design and implementation. We have observed that current literature in context aware work does not mention AO at all or just started [9] [14] to use it as a tool for their system design. Due to limitation in available literature, it is difficult to support this observation. But, there is support available from AO in terms of language extensions, frameworks, and pre-processors that would be helpful to examine and improve existing CA systems. Some of the examples of AO tools are: AspectJ [19] - an extension to Java, HyperJ [20] – a different approach to AO in Java, AspectC++ [26] - an extension to C++, etc.

5. Conclusion

We present our position in embedded system development: context awareness is becoming an increasingly important factor; the need of separating the context aware infrastructure development and context aware application development; and applying aspect orientation in context aware system design. A context aware scenario is used to illustrate the proposed concepts and arguments at various levels during analysis of the CA system. The goals and requirements of context aware system are identified and formally introduced. Through an example, we illustrate how aspect orientation can be applied in the development of a context aware system. The ability to evolve at a high rate has become crucial for the context aware systems because of the ever changing and the diverse needs in context awareness. This study highlights the importance of context awareness in embedded system design and builds a framework for future context aware system development. Currently, we are using the framework in developing a context aware system, Scarlet, at IIT.

6. References:

[1] "Project Oxygen Homepage," http://oxygen.lcs.mit.edu/index.html (current Aug. 2003).
[2] M. Satyananrayanan, "Pervasive Computing: Vision and Challenges," IEEE Personal Communications, Aug. 2001.
[3] Survey – context aware refer term report
[4] AOSD http://www.aosd.net/
[5] E.W. Dijkstra, "A Discipline of Programming," Prentice Hall, Englewood Cliffs, NJ, 1976.
[6] R.E. Filman and D.P. Friedman, "Aspect-Oriented Programming is Quantification and Obliviousness," Workshop Advanced Separation of Concerns, OOPSLA 2000, October 2000, Minneapolis.
[7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of AspectJ," ECOOP, vol. 2072, pp. 327--353, 2001
[8] D.Garlan, D. Siewiorek , A. Smailagic and P. Steenkiste, "Project Aura: Toward Distraction-Free Pervasive Computing," IEEE Pervasive Computing, vol. 1, no. 1, Apr.-Jun. 2002, pp. 22-31.

[9] S.S. Yau, F. Karim, Y. Wang, B. Wang, and S.K.S. Gupta, “Reconfigurable Context-Sensitive Middleware for Pervasive Computing,” IEEE Pervasive Computing, Jul.-Sept. 2002, vol. 1, no. 3, pp 33-40.
[10] M. Román, C.K. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell and K. Nahrstedt, “Gaia: A Middleware Infrastructure to Enable Active Spaces,” IEEE Pervasive Computing, Oct.-Dec. 2002, vol. 1, no. 4,pp. 74-83.
[11] A.K. Dey and G.D. Abowd, “The Context Toolkit: Aiding the Development of Context-Aware Applications,” Workshop Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland, Jun. 2000.
[12] G. Chen and D. Kotz, “Solar: An Open Platform for Context-Aware Mobile Applications,” Proc. 1st Int’l Conf. Pervasive Comp., Zurich, Switzerland, Jun. 2002, pp. 41-47.
[13] A.C. Huang, B.C. Ling, S. Ponnekanti and A. Fox, "Pervasive Computing: What Is It Good For?" Workshop Mobile Data Management (MobiDE) in conjunction with ACM MobiCom '99, Seattle, WA, September 1999.
[14] RAPIDware, http://www.cse.msu.edu/rapidware/
[15] K.V. Laerhoven, A. Schmidt and H.W. Gellersen, “Multi-Sensor Context-Aware Clothing,” Proc. 6th Int’l Symposium Wearable Computers, ISWC 2002.
[16] T. Baudel and M. Beaudouin-Lafon, "CHARADE: remote control of objects using free-hand gestures," Comm. the ACM, Jul. 1993, vol. 36, no. 7, p. 28-35.
[17] L. Arnstein, R. Grimm, C. Hung, J.H. Kang, A. LaMarca, G. Look, S.B. Sigurdsson, J. Su and G. Borriello, “Systems support for ubiquitous computing: A case study of two implementations of Labscape,” Proc. 2002 Int’l Conf. Pervasive Computing, Zurich, Switzerland, Aug. 2002.
[18] A K. Dey, “Providing Architectural Support for Building Context-Aware Applications,” doctoral dissertation, College of Computing, Georgia Institute of Technology, Atlanta, Nov. 2000.
[19] AspectJ, http://eclipse.org/aspectj/
[20] HyperJ, http://www.research.ibm.com/hyperspace/index.htm
[21] C. Constantinides, A. Bader, and T. Elrad, “A framework to address a two-dimensional composition of concerns,” Proc. 1st Workshop Multi-Dimensional Separation of Concerns in Object-Oriented Systems at OOSPLA’99, 1999.
[22] G. Kiczales, J. Lampoing, A. Mendhekar, C. Maeda, C. Lopez, J. Loingtier, and J. Irwin, “Aspect-oriented programming,” Proc. European Conference on Object-Oriented Programming (ECOOP), LNCS 1261, 1997.
[23] B. Tekinerdogan and M. Aksit, “Deriving Design Aspects from Conceptual Models,” Position paper ECOOP ’97 Workshop Aspect-Oriented Programming, pp. 410-413, 1998.
[24] P. Netinant, C.A. Constantinides, A. Bader, and T. Elrad, “Supporting the Design of Adaptable Operating Systems Using Aspect-Oriented Frameworks,” Int’l Conf. Parallel and Distributed Techniques and Applications, PDPTA 2000, Oct. 2000
[25] AspectIX - Aspect Oriented Middleware, http://www.aspectix.org/
[26] A. Gal, W. Schroder-Preikschat and O. Spinczyk, “AspectC++: Language Proposal and Prototype Implementation,” OOPSLA 2001 Workshop Advanced Separation of Concerns in Object Oriented Systems, Tampa, Florida, Oct. 2001