

System Imposed and Application Compliant Adaptations

Shangping Ren, Mattox Beckman, Tzilla Elrad
Department of Computer Science
Illinois Institute of Technology
{ren, beckman, elrad}@iit.edu

Abstract

As more and more inexpensive, small and light-weight computer devices become available, a lot of what were formerly thought of as in-office computer related activities are carried out anywhere and anytime. Reading web news is an example. We can surf the web at airports, on trains, and even in camp grounds. However, is it possible that more complicated distributed applications, such as scientific computing, multi-site video game playing, etc., can be adapted to the mobile environment? More interestingly, how can we achieve this without re-writing the complex applications that are designed for distributed platforms in which resource locations, resource types, network connectivity, and locations of the applications themselves are more stationary and stable than in a mobile distributed environment? Our research focuses on application level context-aware adaptations. In our new application paradigm, mobile applications are a composition of three independent parts: (1) application context which identifies the application itself, such as its location, application type, quality of service constraints, hosting device power limit, etc; (2) functionality of the application, which we will call the application's functional behavior; and (3) application adaptation behavior. In traditional distributed applications, it is sufficient to consider only an application's functional behavior, but for mobile applications it is also necessary to be able to adapt to a changing environment. Our research provides a new paradigm to achieve application level adaptations in mobile environment. We also distinguish between two forms of adaptations; imposed adaptations and compliance adaptations.

1. Introduction

Traditionally, when an application starts running on a computer or on a set of resources, the properties of the application such as the application's identity, type (i.e., computation intensive, data access intensive, communication intensive, etc.), quality of service

requirements, and the location of the application are not known, or not of interest to the environment in which the application runs. However, when applications and environments are allowed frequent changes of resource availability and location, the initial resource allocation may not be appropriate or sufficient for the applications to complete. For instance, a person plays an internet video game on a train. As the train moves into rural area, the initially assigned node for receiving his signal starts to become unreachable and the signal transmission becomes slower and weaker. This forces the person to stop the game unless he is willing to accept a lower quality of video or even a text display, unless a new signal processing node is assigned for this application.

This simple practical example implies two important requirements: from the system's point of view, the system resource allocator needs to know the application's context in order to supply appropriate resources for the application to complete. In the above example, the application's physical location, the application's type (communication intensive), and the QoS requirement (high quality video picture, fast response time) are the contents of the application's context. On the other hand, if the application has real-time knowledge about its operating environment, it is possible to adjust its behavior, such as by switching to a low quality video picture, or even switching to a text display. In other words, a mobile application needs to have knowledge about the system context in which it is running in order to adjust itself to the changing environment. However, it is not necessary for individual applications to have complete information about their running environment. Instead, applications desire only the minimum, necessary system context information to simply the process of making adaptation decisions.

In the previous example, the application was aware of a change in its environment and initiated a change to comply to the new situation. Another type of adaptation occurs when the environment automatically

changes the way an application unit operates without the application being aware of the change. The application might be oblivious to the imposed adaptation.

In this paper, we provide a new paradigm for achieving application level adaptations in a changing environment. The outline of the paper is as follows. In section 2 we review current approaches to providing application mobility and their limitations. In section 3, we discuss the components of a system containing adaptable mobile applications in a changing environment. In section 4 we introduce two language constructs, the AppContext and the SysContext, which represent resource needs and availability in a mobile environment. Sections 5 and 6 introduce the CA_Adapter construct, which provide context-awareness to applications, and the CA_Coordinator construct, which encapsulates the policy decisions a system must make regarding the resources allocated to its applications. Finally, sections 7 and 8 discuss our conclusions and directions for future research.

2. Our approach

We distinguish between two forms of adaptation.

Compliance Adaptations: these are responsibility of an application. When faced with changes in resource availability an application may consent to the new level of resource allocation and adapt its requirements accordingly.

Imposed Adaptations: these are responsibility of the system, and occur when the system automatically makes changes in the way an application unit operates. The application might be oblivious to the imposed adaptation. When moving into a new environment an application may not be aware of a potential new requirement such as a new security policy. The system has the authority to weave code into an application.

Having real-time knowledge of the operating environment, applications are able to be programmed in such a way that they can adjust their behavior as the environment changes. However, instead of letting the application adaptation behaviors spread into and become intermixed with the application functional behaviors, we separate and isolate the adaptation functionality into a special entity called **APPLICATION-CA-ADAPTER** a Context Aware Application Adaptor. The **APPLICATION-CA-ADAPTER** is responsible for implementing

compliance adaptations. This separation will not only minimize the code changes to existing applications when moving from distributed computing to mobile computing, but it will also increase code modularity and re-usability in both the functional domain and adaptation domain.

On the system side, a new entity called the System Context Aware Coordinator is defined. The coordinator is not only aware of the system context, but also the application context. It computes the necessary but minimum system context for individual applications and keeps them updated when these contexts change. The Coordinator negotiates with the **APPLICATION-CA-ADAPTER** to reach an agreement.

The system is also equipped with a special entity called **SYSTEM--CA-ADAPTER** a Context Aware System Adaptor. This entity is responsible for enforcing mandatory applications requirements. For example, the system might need to monitor certain kinds of application behaviors or enforce a security policy. The **SYSTEM--CA-ADAPTER** is authorized to weave code into an application where the application is oblivious to the adaptation made by the **SYSTEM--CA-ADAPTER**.

3. Related Work

To achieve application mobility, researchers have been studying different approaches, such as providing new context-aware middleware, context-aware toolkits and designing new languages.

Reconfigurable Context-Sensitive Middleware (RCSM) from Arizona State University[1] uses an object request broker similar to those found in CORBA to handle context information and defines a Context Aware IDL to specify context. With this approach, it is difficult to achieve dynamic adaptation because a context change requires recompilation of the CA IDL code to generate adaptive objects.

The Scarlet project from Illinois Institute of Technology[4] uses a layered approach to provide context information to applications. The Scarlet system has four different interacting layers that work together to provide context awareness. The context aware application layer includes context providers, context consumers and context aggregators. Examples of such applications include temperature sensors and location sensors. The context provided by the context aware application layer is used by human users who

directly interact with the system. The adaptation is done at the user level.

The Context Toolkit from Georgia Institute of Technology[5] is a system that focuses solely on context awareness. It has six main services: encapsulation of sensors, access to context data through a network API, abstraction of context data through interpreters, sharing of context data through a distributed infrastructure, storing of context data and history, and basic access control for privacy protection. Each of the elements in the system is able to run independently of the others and hence allows the components to be reused by multiple context-aware applications.

Another approach is to utilize special features of the Java programming language to provide a complete framework for pervasive computing, such as One.world by University of Washington[2,3]. The interesting feature of One.world is that adaptability lies in the program and application level. This frees the infrastructure from having to handle different adaptations for the applications. Furthermore, the application is given more freedom to adjust its behavior independently.

We use a similar approach in that applications first obtain system context and then adapt their behavior based on the obtained operation context.

Our work is also influenced by AOSD technology [15,16,18]. Aspect Oriented Software Development allows the kind of adaptabilities we proposed to be realized. *Crosscutting concerns* are those concerns that have descriptions which crosscut the descriptions of other concerns. For example, security, logging, synchronization and fault tolerance are a few instances of crosscutting concerns. The implementation of a security policy might be scattered in many different places. As a result, the implementation of each crosscutting concern is tangled with implementations of few others. This makes the adaptability of such concerns practically impossible. The key strength of Aspect Orientation is the modularization of crosscutting concerns. A module that captures a crosscutting concern is called an aspect. Once the system aspects are modularized we need mechanisms to weave them back in the right places. This mechanism is called “weaver”. Aspect orientation provides quantification and obliviousness. [19] Quantification means the ability to express all the right places an aspect should be inserted. Obliviousness means the modified unit is completely unaware of a potential aspect being weaved into it. These two properties make both system imposed

adaptation and the application compliance adaptation realizable by aspect orientation technology.

4. System Architecture

In a mobile system, there are mobile computer devices connected by different types of networks, such LAN, WLAN, etc. Applications run on mobile computer devices. When these applications need more resources, they send requests to a system context aware coordinator which gathers information about the application’s resource requirements, and responds with customized information about the resources that will be made available to the application, along with handles or access points to those resources. This information is an encapsulation of the application’s *system context*, and can be thought of as a contract between a mobile application and its mobile environment promising that the specified system resources will be available based on the current system conditions. Applications adapt their behaviors according to the given system contexts. Meanwhile, the coordinator keeps these contexts updated as the system evolves. Figure 1 depicts the abstract level system architecture.

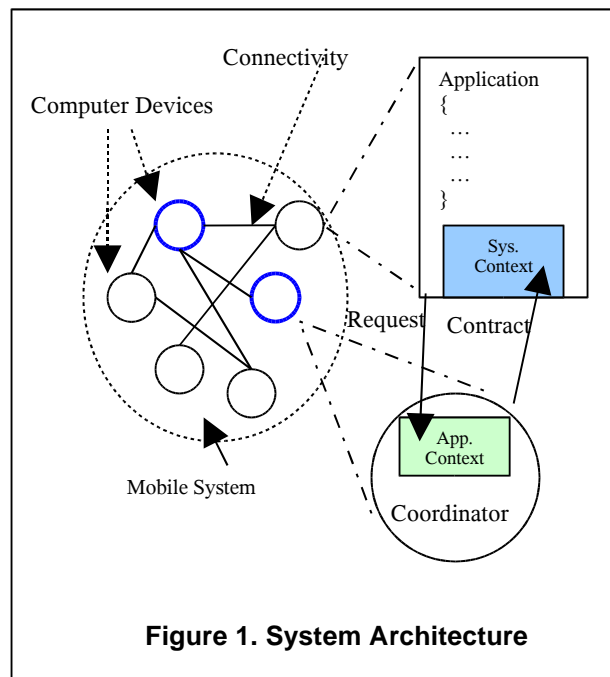


Figure 1. System Architecture

As shown in Figure 1, mobile applications in our paradigm contain an independent module, a context aware adapter. System coordinators provide real-time context information to adapters. Based on the context information, the adapters make corresponding adaptation decisions and adjust the application’s

functional behaviors. Coordinators reside on more stationary computer devices to provide more continuous support to mobile applications.

4.1 Two Levels of Applying Aspect Orientation.

We have two levels of aspect orientation: The first level is the implementation of system adaptations. Here the system can quantify over all places in any potential application where modifications need to be inserted. Applications are oblivious to these changes. The second level is the implementation of application compliance adaptations. Here each application is allowed to quantify over local places within the application where modifications are needed. As in the first level, the local modules are oblivious to the application adaptations.

5. Application Context and System Context

As we described in the previous sections, a mobile application needs to have system environment information, i.e., system context, in order to adapt its behavior appropriately. A system coordinator could either send the complete system resource information to the application, or it could send only application related system context to the application. The first approach implies that a more sophisticated decision strategy must be in place at the application side in order for the application to adapt itself to the changing environment. The process of searching through the system context information for relevant information will take more computation, which means more energy consumption and thus more adversity on applications running on mobile devices. Therefore, we choose relatively stationary computer devices to host system coordinators. The coordinators in turn customize application related system contexts and send the smaller amount of context information (relative to whole system context information) back to the applications.

In order for system coordinators to provide tailored context information to applications, the applications must provide their identities and requirements in the form of an *application context*. From an object oriented point of view, an application context can be a hierarchy of context objects. However, in our system, the base application context must contain the following four elements: the application's location, the application's type (such as computation intensive type, communication intensive type, etc), the non-functional constraints (such as QoS requirements, real-time

constraints, etc.), and the available power supply, which is defined as a function of device usage, i.e., *power: usage->time*.

We define a new type of entity to specify application context as follows:

```
AppContext ()
{
    Location loc;
    RequestType reqType;
    Constraints constraint;
    Power power(usage);
}
```

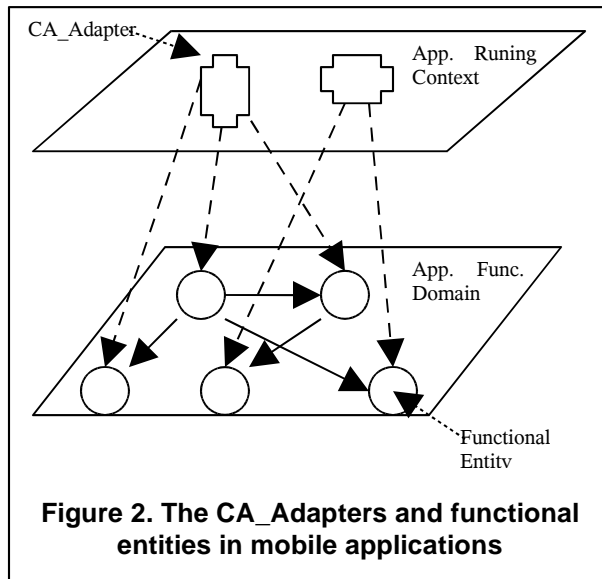
A System Context is a collection of resources and their quantity. The context needed for an individual application is a subset of the system context. The identifying characteristics of each resource include the resource location, resource type (CPU, memory, data storage, etc.), and quantity, its connectivity with other resources, and the power limit of the devices on which the resource resides. The system context and resources are defined as a new type of objects as following:

```
SysContext ()
{
    Resource rec[n];
};
Resource ()
{
    Location loc;
    ResourceType rType;
    Quantity quality;
    Connectivity conn;
    Power power(usage);
};
```

6. Context Aware Application Adapter

As the application's runtime environment changes, it is desirable that the application be able to adjust itself to the new environment. The context-aware application adapter is designed to provide applications with adaptability.

An adaptive application has two orthogonal behaviors---functional behavior and adaptation behavior. Separation of an application's functional behavior and adaptation behavior will not only minimize the code changes when moving distributed applications to mobile distributed applications, but it will also allow the implemented adapter to be re-useable in other applications.



Using the aspect-oriented programming approach, we define the adapter as a separate type of object. The adaptation logic based on the context is programmed inside the adapter and it is weaved at run-time with other functional code. Further more, the system context that the adapter is aware of is dynamically updated by the system coordinator when the environment changes.

In order to facilitate the specification of such a context aware application adapter, a new language construct Application- CA-Adapter is defined. This construct is used to specify adaptation rules. The rules are checked at run time to decide the application's execution path. The definition of CA_Adapter is as follows:

```
CA_Adapter appAdapter (SysContext context)
{
  State
  SysContext sysContext = context;

  Policy
  sysContext.resource1.quantity < threshold1
  ==>path 1;
  ...

  Update
  Update (SysContext newContext);
}
```

Each application may have multiple CA_Adapters; they work conjunctively assuming that there are no conflicting rules among different CA_Adapters.

The behaviors of application adapters are independent of the application's underlying functional entities. The

relationship between these two different types of entities is depicted in Figure 2.

In the multi-site video game application example, let us assume that the system context returned by coordinator is a single resource, i.e., bandwidth. And if the bandwidth is less than a threshold, say, 10mbps, the application will use a text display, instead of a high quality graphic display. Using a CA_Adapter, the adaptation policy of the multi-site video game application can be specified as following:

```
CA_Adapter videoAdapter (SysContext context)
{
  State
  SysContext sysContext = context;

  Policy
  sysContext.rec[0].rType == "bandwidth" &&
  sysContext.rec[0].quantity < 10mbps
  ==>textDisplay();
  ...
}
```

When the mobile system changes, for instance, new resources become available to the system, the system coordinator may change the context previously provided to the application. Such an update is seamless, automatic, and will affect future adaptations.

The context aware adapter and its adaptation policies are dynamically weaved with application's functional behavior at runtime. It facilitates the application with dynamic adaptation to the changing circumstances.

7. Context Aware System Coordinator

As shown in Figure 1, special entities, called context-aware coordinators, are installed on relatively stationary computer devices. They monitor system resource changes, gather system context for individual applications, and update the applications with the current context. Since the coordinators have both knowledge about the system and individual application context, they are able to deliver *precisely* the necessary and applicable system context information to the applications.

Coordinators have state variables which are directly connected with sensors that monitor resources, such as connectivity among computer devices, network bandwidth, available data storage, and available CPU cycles, etc. An environment change is reflected by value changes in the state variables. In addition, there are set rules for gathering system contexts for each individual application. The rules are not only

application context dependent, but also coordinator state dependent. In other words, if an environment change causes a coordinator state change, the state change can activate different rules and in turn cause the coordinator to generate different system contexts for given applications. The language construct for coordinators follows:

```
CA_Coord (SensorValue sValue[n])
{
  State
  SysContext sysContext = context;
  // context is obtained by reading the sensor values
  AppContext appContext;

  Policy
  sysContext.resource1.quantity > threshold1
  &&
  appContext.RequestType ==
  sysContext.resource1.type
  ==> resource1 is available for the application;
  ...

  Operations
  updateContext (SensorValue sValue);
  // sysContext is updated by the new sensed values

  SysContext getContext(AppContext appContext);
  // generate system context tailored for given
  // application context
}
```

The key portion of the context aware coordinator is its *policy* which determines how the system contexts are tailored for a given application. It is obvious that, before the system can reserve a resource for an application, that the system must have enough of that type of resource. However, there might be more than one such resource that satisfies the quantity requirement. Hence, we may need other rules to assist the coordinator in gathering runtime contexts for the applications. For instance, the resource that is closer to the application should be considered first.

When an application needs a resource which is not available on the device that the application resides, it communicates with the system coordinator and presents its 'getContext' request together with the application's own context. Upon receiving the *getContext* request, the coordinator will use the policy to determine the context in which the application will operate.

The coordinator keeps track of the current mobile system's state. As new mobile devices move into the system, existing devices move out of the system, or

other events occur such as a device exhausting its power supply, system sensors will detect these changes. This in turn will trigger the *updateContext* operation and cause the coordinator's state to reflect the mobile system changes.

Similar to context-aware application adaptors, there can be multiple context-aware coordinators residing in different computer devices in the mobile system. If the rules conflict with each other among different coordinators, the coordinator which is closest to the application physically will dominate the others.

8. Imposed adaptations

Up to now we have described how to deal with compliance adaptation. Imposed adaptations are implemented by the system in a similar way presented in the chapter *Be Aware: Dynamic Aspect-Oriented Infrastructure* of [15]. One of the main obstacles in implementing imposed adaptations is crosscutting concerns---concerns whose implementations crosscut the implementations of other concerns. Because aspect orientated methodology provides explicit support for the modularization of crosscutting concerns, we propose to use aspect oriented software development technologies to implement "application awareness" for distributed applications.

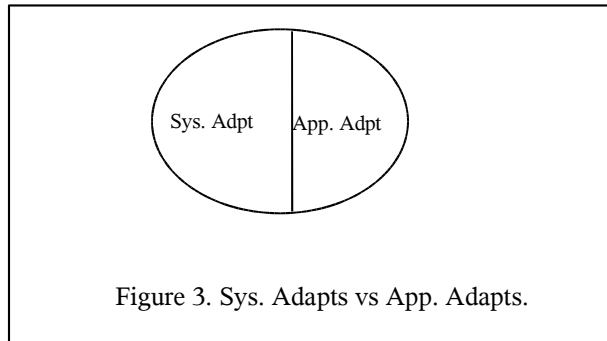
Application awareness is characterized by two main requirements: that the computing environment be aware of all applications running along with their properties, and that the environment provides the applications with the necessary adaptations to local conditions. The fact that the responsibility for adaptation of all (possibly dynamic) applications is on the environment is one source of many crosscutting concerns.

With a conventional application design, the ability to react to changes is encoded in each of the appliances before they are deployed. This scattered implementation limits the applications' ability to coordinate and orchestrate the desired environment awareness. With dynamic aspect orientation one can implement adaptation by weaving aspects through mobile devices when they enter and leave a network.

9. Imposed adaptations vs compliant adaptations

As we have discussed, the imposed adaptations are system wide and they are implicitly weaved into all applications that are currently executing on the system. In contrast, the compliant adaptations are contained

within each individual application, and it is the applications' decisions to comply with the adaptations. One way to think of this is that adaptation tasks are divided into two separated domains as shown in Figure 3. System imposed adaptations are independent of application compliant adaptations and vice versa. Furthermore, although these two types of adaptations are specified at different locations (the system level and application level, respectively), they both are eventually weaved into the applications.



Another view is a containment view, i.e., application compliant adaptations are contained in system imposed adaptations. This is because application compliant adaptations can always be specified as a special case of system imposed adaptations. We are open to discussion of these two approaches.

10. Conclusion

One of the key differences between mobile computing and distributed computing is the mobility of computation devices. Hence, it is imperative that applications running on such mobile devices be able to adjust their functional behaviors to the changing environment seamlessly. Our research focuses on providing such adaptability at the application level. We separate mobile applications' adaptation behaviors from their functional behaviors. The new language construct (Application-CA-Adapter) is designed to facilitate the separate specification of a mobile application's adaptation behaviors. Separating a mobile application's functional and adaptation behaviors not only increases the application program's modularity and code re-usability, but also simplifies the process of extending existing distributed applications to a mobile environment.

In addition, since a coordinator is able to determine which context information is relevant to a specific application, the adaptation process is greatly simplified within the context-aware adaptor --- the

adaptor need not consider the whole system environment, only the pertinent information.

Coordinators and adaptors together allow us to achieve application-level adaptation in a mobile environment. An independent unit of System-AW-adapter allows us to achieve imposing adaptations.

11. Future Work

We presented two types of adaptations. For the compliance adaptations, we have presented a new paradigm for achieving application-level context aware adaptation in mobile systems. There are many areas we would like to investigate from this point.

We would like to be able to specify more precisely the semantics of the language constructs used to manipulate contexts and coordinators, and be able to characterize the interactions between applications and coordinators. One interaction of interest is the "meta-coordinator". Since coordinators themselves are also applications and resources, they can be created and managed by other coordinators in real time.

One of the difficult issues in this paradigm is for the coordinator to identify the minimal but necessary system contexts for individual mobile applications. It involves optimized resource allocation strategies, policy specification and context interpretations, which will be our research focus in the future.

Another interesting topic is that of failure modes. While individual applications may have standard strategies to handle the loss of a critical resource, the problem becomes more interesting if the loss of resource occurs to a coordinator. Possible solutions include having the fading coordinator chose a replacement for itself, having the applications be able to locate a replacement coordinator, or even having the applications move themselves to a different environment.

12. References

- [1] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K.S.Gupta, "Reconfigurable Context-Sensitive middleware for pervasive computing", *IEEE Pervasive Computing*, 1(3):33-49, July—September 2002.
- [2] Robert Grimm, "System Support for Pervasive Applications", *PhD Thesis*, University of Washington, December 2002.

- [3] Robert Grimm, Tom Anderson, Brian Bershad, and David Wetherall, "A System Architecture for Pervasive Computing", *Proceedings of the 9th ACM SIGOPS European Workshop*, pp 177-182, Kolding, Denmark, September 2000.
- [3] Robert Grimm, Janet Davis, Eric Lemar, Adam acBeth, Steven Swanson, Steven Gribble, Tom Anderson, Brian Bershad, Gaetano Borriello, and David Wetherall, "Programming for Pervasive Computing Environments", *Technical Report UW-CSE-01-06-01*, University of Washington, Department of Computer Science and Engineering, June 2001.
- [4] Patrick Wagstrom, "SCARLET --- A Framework for Context Aware Computing", *Master Thesis*, Illinois Institute of Technology, Department of Computer Science, July 2003
- [5] Asim Smailagic and Daniel Siewiorek, "Application Design for Wearable and Context-Aware Computers", *IEEE Pervasive Computing*, pp20—29, October - December, 2002.
- [6] Cynthia A. Patterson, Richard R. Muntz, and Cherri M. Pancake, "Challenges in Location-Aware Computing", *IEEE Pervasive Computing*, pp80—89, April – June 2003.
- [8] Mark Weiser, "Some Computer Science Issues in Ubiquitous Computing", *Communication of the ACM*, pp 75—84, Vol 36, No.7, 1993.
- [9] David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste, "project Aura: Toward Distraction-Free Pervasive Computing", *IEEE Pervasive Computing*, pp 22—31, April – June 2002.
- [10] Joshua Anhalt, Asim Smailagic, Daniel P. Siewiorek, Francine Gemperle, Daniel Salber, Sam Weber, Jim Beck, and Jim Jennings, "Toward Context-Aware Computing: Experiences and Lessons", *IEEE Intelligent Systems*, pp 38—46, May/June, 2001.
- [11] Kimmo Raatikainen, Henrik Barbak Christensen and Tatsuo Nakajima, "Application Requirements for Middleware for Mobile and Pervasive Systems", *Mobile Computing and Communications Review*, pp 16—24, Vol. 6, No. 4, 2002.
- [12] Guruduth Banavar, James Beck, Eugene Gluzberg, et.al., "Challenges: An Application Model for Pervasive Computing", *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking 2000*, pp 266—274.
- [13] Kris Gybels and Johan Brichau, "Arranging Language Features for More Robust Pattern-based Crosscuts", *Proceedings of the 2nd international conference on Aspect-oriented software development 2003*, pp 60--69
- [14] Mati Shomrat and Amiram Yehudai, "Obvious or Not? Regulating Architectural Decisions Using Aspect-Oriented Programming", *Proceedings of the 1st international conference on Aspect-oriented software development, 2002*.
- [15] Tzilla Elrad "Aspect Oriented Software Development"
In McGraw Hill 2003 Yearbook of Science and Technology.
- [16] Tzilla Elrad, Robert Filman and Atef Bader "Aspect Oriented Programming" *Special issue of the Communication of the ACM, October 2001.*
- [17] Andrei Popovici, Gustavo Alonso, Thomas Gross "Be Aware: Dynamic Aspect-Oriented Infrastructure" *To appear in Aspect Oriented Software Development edited by Robert Filman, Tzilla Elrad, Siobhan Clacke and mehmet Aksit. To be published by Addison Wesley 2004.*
- [18] www.asod.net
- [19] Filman, R. E., and Friedman, D. P. "Aspect-oriented Programming is quantification and implicit invocation". *Workshop on Advanced Separation of Concerns, OOPSLA 2000, Oct. 2000, Minneapolis.*
<http://trese.cs.utwente.nl/Workshops/OOPSLA2000/papers/filman.pdf>