Towards a Domain-Specific AOP language for Ubiquitous Computing

Serena Fritsch, Jennifer Munnelly and Siobhán Clarke
Distributed Systems Group,
Department of Computer Science,
Trinity College Dublin,
Ireland.
email: {fritschs, munnelj, sclarke}@cs.tcd.ie

ABSTRACT

Ubiquitous computing is a vision in which computers are integrated into a user's environment and aid a user to perform everyday tasks. When developing applications in this domain, crosscutting concerns such as mobility, device characteristics, context awareness and network heterogeneity are encountered. Aspect-oriented programming (AOP) provides a means to separate these concerns. However, current state of the art AOP languages rely on low level general purpose language constructs. Using these constructs, a programmer cannot take advantage of domain knowledge to capture join points. In this paper we propose a new domain-specific aspect language which directly captures the crosscutting concerns that emerge in ubiquitous computing applications.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

[Programming Languages] Languages

Keywords

Ubiquitous Computing, Aspect-oriented programming, Aspect languages

1. INTRODUCTION

Current advances in portable devices and networking technologies have precipitated a vision for ubiquitous computing that includes access to information anytime, and anywhere. In a perfect ubiquitous environment the devices integrate themselves into the real world seamlessly [13]. However, developing applications for ubiquitous computing has been proven significantly more complex than developing for standard systems, since they have to cope with recurring crosscutting concerns, like mobility and adaptation.

The AOSD [4] paradigm has emerged as a promising means for separating concerns that cut across an entire system, while retaining the principle of the encapsulation of concerns. In essence, it provides mechanisms for the systematic identification, modularisation, representation and composition of crosscutting concerns.

In current state of the art aspect languages, like AspectJ [2], the means provided for the specification of predicates that identify relevant join points (i.e., pointcuts) relies mainly on the static language constructs of the base code, such as method calls, or access of fields. Pointcut definitions lack appropriate expression mechanisms for describing concerns related to specific domains. Since pointcuts are at a low level of abstraction, an explicit mapping from them to the parts of the application where crosscutting behaviour should be applied, is required from the application developer. This leads to erroneous applications, since join points might easily be missed and crosscutting behaviour at relevant places might not be applied.

In our approach, we overcome these problems by providing a new programming model for ubiquitous computing in which a programmer is given means to identify crosscutting behaviour based on the semantics of the application. We expect that this approach will decrease development time, and increase the timeliness of application deployment.

The rest of this paper is structured as follows. Section 2 discusses the crosscutting concerns which might arise in ubiquitous applications and describes the need for domain-specific pointcuts. Section 3 presents our approach in detail. Section 4 describes related work. Finally, Section 5 concludes with a discussion of possible implementation techniques for our approach.

2. BACKGROUND

In order to provide a context for this work, this section describes the concerns that arise in ubiquitous computing applications, and illustrates the deficiency of pointcut expressions in current AOP languages.

2.1 Concerns in Ubiquitous Computing

Ubiquitous computing concerns are aggravated by the range of processing constraints of the devices on which the applications reside and by the wireless networks in which they com-

municate. Mobility in ubiquitous computing applications encompasses concerns related to the dynamicity arising from the diversity of devices and the frequency of changing environments. Concerns such as roaming, ad-hoc networking, limited connectivity, location, proximity, service discovery, quality of service, security and heterogeneity are relevant here. The area of contextual reasoning is a central concern in ubiquitous applications. Modularising context into more defined concerns such as user context, social context, device context, system context, temporal context, application specific context and environmental context enables applications to capture and separate these concerns independently.

Inevitably, knowledge about these concerns creep into all parts of the program. This results in code tangling which makes modification and extension difficult. The use of aspect oriented techniques to separate ubiquitous computing concerns has been proven to enhance the modularisation of the application functionality [10].

2.2 The need for Declarative Pointcuts

AOP is characterised by the desire to make programming statements in the form [3]:

```
In programs P, whenever condition C arises

perform action A.
```

In traditional AOP languages like AspectJ, P is the application in a language compliable with the AOP compiler and weaver, C is a pointcut specified in the AOP languages join point model, and A is the advice in the AOP language. By separating the components, they still comply with the above definition, but allow a greater scope for the mechanism used to specify how, when and what takes place when crosscutting concerns arise.

Unfortunately, most aspect languages proposed so far rely too much on the base programs' static structure [12]. Pointcuts are currently defined in most aspect-oriented frameworks either by way of laborious enumeration or by referring to some structure of the code [1]. This leads to pointcut declarations which are harder to write and are also susceptible to changes to the base applications.

The need for pointcuts which directly express the property they are interested in, is identified in [6]. These declarative pointcuts can be specified by means of semantic properties of the application. These constructs do not rely on the syntax of the base application and thereby improve the reusability and robustness of aspects.

3. UILE: AN AOP FRAMEWORK FOR UBIQUITOUS COMPUTING

In our approach, we envisage a framework that is accessible to the developers within this domain who might have little or no knowledge of AOP. More intuitive, and domain-specific constructs, coupled with the developers existing knowledge, simplify application development. These constructs state how and when to apply a crosscutting behaviour in terms of the semantics of the application, e.g., changes to the location of a user or to network access. The application developer should be able to express a statement in the form:

```
whenever the user's location changes
execute the method locationUpdate() of aspect A
```

We are using declarative pointcuts [6] for ubiquitous computing, which support the identification of joinpoints by their semantic properties. Since the domain of ubiquitous

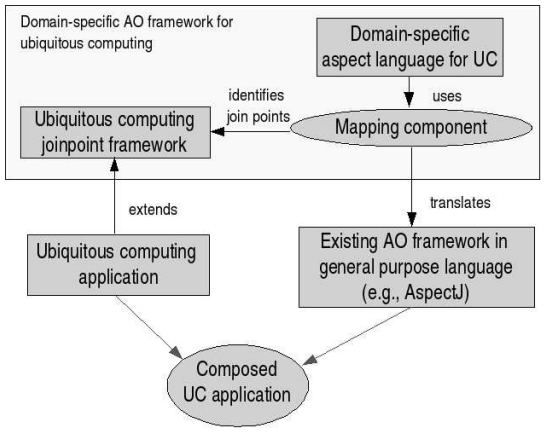

Figure 1: Framework Architecture

computing is extensive, it is important for the language to facilitate extensibility. The application developer should be able to extend the language by defining additional declarative pointcuts and abstraction mechanisms with minimal effort and in a timely manner. In order to achieve this, we propose a framework which is depicted in Figure 1. The framework is comprised of the aspect language, a mapping component and a join point framework. The application developer builds parts of the application where crosscutting behaviour might occur within the joinpoint framework. The bindings between the application and the crosscutting behaviour is specified with the domain specific aspect language.

For the domain-specific aspect language, we intend to develop on an existing AOP framework, such as AspectJ, in order to make use of existing weavers. Since these general purpose languages provide only low-level language constructs, the domain-specific notations of our language need to be mapped to their equivalent pointcut and advice expressions.

This mapping is achieved by the Mapping Component which identifies the relevant join points using the join point framework. Figure 2 shows an example translation from a domain specific construct of our language to an AspectJ compliant pointcut, based on abstract classes of the join point framework. To define new declarative pointcuts, the developer is only required to provide an association between the declarative pointcut and classes of the join point framework.

The join point framework offers abstract classes to which the domain-specific pointcuts are mapped. The developer places the parts of the application at which crosscutting functionality should be applied, in appropriate places in the framework. The framework's classes operate as hooks for

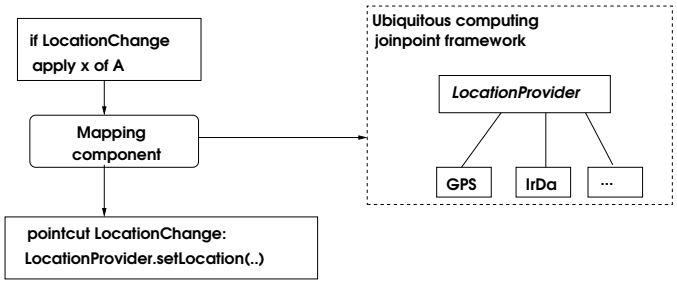

Figure 2: Example Mapping from domain-specific construct to general AOP language

the declarative pointcuts defined in the aspect language. By using this framework, the developer can be ensured that no relevant join points are missed without the need for explicitly enumerating all related methods and classes.

From the proposed framework, open questions arise, mostly pertaining to the implementation. Firstly, language extensibility is a requirement in such a dynamic domain as ubiquitous computing. The means by which extensibility is provided is dependant on the implementation of the aspect language. Implementation decisions have been identified, and possible approaches are discussed in Section 4. Potential techniques for the realisation of the association between declarative pointcuts of the aspect language and the join point framework are open to investigation.

4. RELATED WORK

In this section, we focus on approaches towards more expressive pointcuts and frameworks supporting the development of domain-specific languages.

The use of developer-extensible program annotations in the base code from which join points can be specified by means of semantic properties of the programs was discussed in [11] and more recently in Setpoint [1].

Setpoint [1] is an aspect oriented framework built on top of Microsoft's .NET architecture [9] and based on semantic pointcuts. These are realised by annotating the source code through metadata, which can later be used in the construction of semantically rich pointcuts to guide aspect weaving. Semantic concepts are captured in ontologies and are then referenced by the annotations, which are .NET custom attributes. A domain-specific language called *LENDL* was developed for advice and pointcut declarations. This language is translated into C# and parsed into RDF at a later stage.

Using annotations as a means of recording and locating the points at which crosscutting concerns arise increases the semantic information at these points. However, according to [7], this approach requires the scattering of the annotations in the base code and hence might lead to evolution and scaling problems. Furthermore, inserting annotations involves altering the base application at the lowest possible level, within the existing source code. This modification does not reduce the dependency on the syntax. Consequently, this approach will not be investigated further.

The design of the Aspect Markup Language (AML), an XML based AOP language, is described in [8]. In this approach, aspects are seen as first-class entities which are written in a general-purpose language. The binding between components and aspects is achieved by XML-based binding specifications. The application developer is required to provide a plugin to be able to define custom elements. The plugin defines a mapping between the custom elements and AML. This feature enables the development of domain-specific aspect languages on top of AML.

Since XML supports extensibility, this approach may be beneficial to our framework. Further evaluation of the use of AML as an underlying model for the proposed aspect language will be undertaken. However, this approach has drawbacks relating to its use of XML. The developer has to provide a plugin, which translates the domain specific pointcut definitions into the core elements of AML, introducing a requirement for an explicit mapping to all relevant join points in the application. By providing a mapping on the abstract classes of the join point framework, some of these drawbacks could be alleviated, since the application developer can be ensured that no join points will be missed when placing parts of the application inside the framework hierarchy.

A logic based pointcut language outlined in [5] aims to provide more expressive mechanisms to describe the underlying pattern of the points in a program which are crosscut by an aspect. Pointcuts are identified by expressions written as logic queries over join points. The language uses key features of logic languages, e.g., unification, recursion and the reasoning about properties to allow the writing of more expressive pointcuts. By introducing new rules, the user is able to define new predicates in the language.

By providing a more expressive mechanism, the pointcut constructs are raised to a higher semantic level. We will investigate the design of rules which are targeted towards ubiquitous computing in the logic based pointcut language.

5. CONCLUSION

In this paper, we propose an aspect language for ubiquitous computing which enables the developer to reason more intuitively about crosscutting concerns in the domain. By providing declarative pointcuts, the application developer can employ domain knowledge of the problem in semantic form with the assurance that all relevant join points will be included.

The aspect language leverages existing AOP frameworks, like AspectJ. We provide a Mapping component which translates the domain-specific declarative pointcuts into language constructs of the underlying aspect language. The mapping component makes use of classes defined in the join point framework. The framework offers abstract classes, to which the declarative pointcuts of the language refer.

Our next step is to define the constructs of our language, based on an evaluation of crosscutting concerns in the domain of ubiquitous computing. We will provide a mapping and the corresponding framework classes for several key concerns, like service discovery and network roaming. We will

evaluate the language by using it in the development of a range of ubiquitous computing applications.

6. ACKNOWLEDGMENTS

This work is funded by Science Foundation Ireland under the Research Frontiers Program. The authors would like to thank Alexandre Bergel, and Andrew Jackson for their valuable comments on earlier drafts of this paper.

7. REFERENCES

[1] R. Altman, A. Cyment, and N. Kicillof. On the Need for Setpoints. In *European Interactive Workshopn on Aspects in Software*, 2005.

[2] AspectJ. http://www.eclipse.org/aspectj/.

[3] R. Filman and D. Friedman. Aspect-oriented Programming is Quantification and Obliviousness. In *Workshop on Advanced Separation of Concerns, OOPSLA 2000*, 2000.

[4] R. E. Filman, T. Elrad, S. Clarke, and M. Akşit. *Aspect-oriented Software Development.* Addison-Wesley, 2005.

[5] K. Gybels and J. Brichau. Arranging Language Features for More Robust Pattern-based Crosscuts. In *OOPSLA'04 Workshop on Building Software for Pervasive Computing*, 2004.

[6] J. Hugunin. The next Steps for Aspect-Oriented Programming Languages (in Java). In *Workshop on New Visions for Software Design and Productivity*, 2001.

[7] G. Kiczales and M. Menzini. Aspect-oriented programming and modular reasoning. In *ICSE '05: Proceedings of the 27th international conference on Software Engineering*, 2005.

[8] C. V. Lopes and T. C. Ngo. The Aspect Oriented Markup Language and its Support of Aspect Plugins. Technical report, Institute for Software Research and Bren School of Information and Computer Science, University of California, Irvine, 2004.

[9] M. .NET. http://www.microsoft.com/net/default.mspx.

[10] A. Rashid and G. Kortuem. Adaptation as an Aspect. In *OOPSLA'04 Workshop on Building Software for Pervasive Computing*, 2004.

[11] V. Sazawal. Separation of Concerns for Ubiquitous Computing. In *ICSE '01 Workshop on Advanced Separation of Concerns*, 2001.

[12] M. Stoerzner and C. Koppen. Pcdiff: Attacking the Fragile Pointcut Problem. In *EIWAS'04 Workshop on European Interactive Workshop on Aspects in Software*, 2004.

[13] M. Weiser. The computer for the 21st century. *Human-computer interaction: toward the year 2000*, pages 933–940, 1995.