ALPH: A Domain-Specific Language for Crosscutting Pervasive Healthcare Concerns

Jennifer Munnelly
Distributed Systems Group
Trinity College Dublin
munnelj@cs.tcd.ie

Siobhán Clarke
Distributed Systems Group
Trinity College Dublin
sclarke@cs.tcd.ie

Abstract

Pervasive healthcare is an advancing discipline that applies ubiquitous computing features to applications deployed in the healthcare domain. In these applications, ubiquitous computing concerns and health informatics concerns are entwined with base functionality resulting in significant, complex crosscutting code. Domain-specific languages (DSLs) can reduce development effort by providing higher level programming abstractions for domain-specific functionality. We introduce ALPH (Aspect Language for Pervasive Healthcare); a DSL that provides domain-specific constructs for tasks and entities within the pervasive healthcare domain. The DSL is translated into an aspect language and the crosscutting behaviour is weaved. We describe an example implementation to illustrate the level of abstraction that can be achieved using domain-specific constructs.

Categories and Subject Descriptors D 2.3 [*Software Engineering*]: Coding Tools and Techniques

General Terms Languages, Performance, Design

Keywords ALPH, Domain-Specific Languages, Aspect-Oriented Programming, Pervasive Healthcare

1. Introduction

Applications in the advancing area of pervasive healthcare employ the features of ubiquitous computing to advance technology in the healthcare sector. Like many industries, healthcare has recognised the advantages to be gained by the applied use of technology. Globally, technology has reengineered the healthcare industry resulting in increased productivity, reduced human error and increased interoperability between various healthcare areas and facilities.

The term ”pervasive healthcare” represents two focal aims; firstly to enable access to healthcare information anytime, anywhere, and secondly to apply ubiquitous computing technology in order to create intelligent applications that can apply these benefits as required e.g., dynamically adapting to their environment.

To develop applications in the pervasive healthcare domain, many difficulties must be overcome. These include both ubiquitous computing and health informatics issues. Incorporating ubiquitous computing concerns requires the adoption of issues including mobility and context-awareness. These issues are inherently crosscutting as the entire application must adapt behaviour at many points in the base functionality. Health informatics introduces concerns including messaging formats and security. Healthcare messaging crosscuts entire applications and is diametrically linked with all communication both within applications, and between autonomous applications. These concerns are discussed in detail in Section 2.

Pervasive healthcare applications have typically been developed using traditional programming techniques. With the arrival of many ubiquitous middlewares and frameworks, the ubiquitous computing side of development has been aided, but there is little support for pervasive healthcare application development in its entirety. We propose a domain-specific language for pervasive computing, ALPH (Aspect Language for Pervasive Healthcare). Domain specific languages provide the means to program efficiently within a particular domain. High level constructs carry out tasks specific to the domain and provide a higher level of abstraction to developers. This leads to more expressive code and eases application development. The use of DSLs in lieu of general purpose languages (GPLs) reduces the amount of domain-specific knowledge required by the developer, as the constructs are more intuitive and semantically representative. The ALPH language aims to provide constructs which represent domain-specific tasks or entities. The DSL itself is transformed via a dynamically extensible translator into an aspect language, which is weaved into the base application. The rest of this paper is structured as follows: Section 2 discusses crosscutting concerns in pervasive healthcare, Section 3 describes the domain analysis, Section 4 discusses a prototypical implementation, Section 5 outlines related work and Section 6 illustrates our conclusions.

2. Crosscutting Pervasive Healthcare Concerns

The concerns that are scattered and tangled throughout pervasive healthcare applications include both ubiquitous computing concerns and concerns from health informatics.

2.1 Ubiquitous Computing Concerns

In previous work we focus on two key areas of ubiquitous computing; mobility and context. We studied their implementation in ubiquitous computing applications and identified the recurring concerns that were tangled within the base functionality. In mobility, eight primary concerns and a series of ancillary concerns were identified as being crosscutting. The primary concerns are roaming, service discovery, device discovery, ad hoc networking, limited connectivity, location, proximity, and quality of service [16]. The incorporation of context-awareness enables applications to identify changes in the environment and adapt accordingly. We have broken down context into eight sub-categories, each dealing with a different type of crosscutting context. The context categories are: user, social, device, location, environment, application, time and infrastructural context [2]. The ancillary concerns are well known

Workshop DSAL '07 March 12, 2007 Vancouver, British Columbia, Canada.
Copyright © 2007 ACM 978-1-59593-659-8. . . $5.00

issues including security and distribution. These have been heavily researched and have many existing solutions therefore we mention them only for completeness. Reasoning has also emerged as a frequent crosscutting concern in pervasive healthcare applications. Cross-checking prescribed medicines with medicines administered and inferring diagnosis from symptoms are examples of reasoning functionality identified. Our findings are supported by findings of requirement engineering studies in the domain [13][15].

2.2 Health Informatics Concerns

When applications are deployed in a healthcare environment, additional concerns are encountered. The use of varying hospital and departmental information systems create the need for standardisation. Conformance to standards must be implemented in order to enable interoperability of both in-house heterogeneous systems and inter-establishment systems. The international Health Level Seven (HL7)[1] standards institution promotes and enforces the standardisation of electronic healthcare information to facilitate its exchange and management. These standards must be incorporated throughout the base code, resulting in badly modularised applications. Comprehensive Electronic Health Record (EHR) systems are full patient records consisting of files and components from various hospital and healthcare professional systems. Many pervasive healthcare applications aim to incorporate some form of EHR. EHRs have proven extremely difficult to develop as many issues again crosscut entire systems e.g., the manipulation of patient records. Huge duplication exists even in situations where EHRs have been implemented due to the requirement for logging and paper trails. The provision of national or international patient identification numbers is also a frequent problem.

3. Domain Analysis

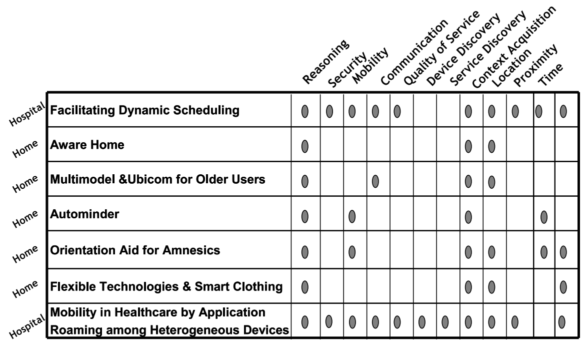
The purpose of providing a DSL is to ease application development in the target domain, by providing programming abstractions for domain-specific functionality. In order to identify this functionality it is necessary to perform a comprehensive domain analysis. This section describes the details of the domain analysis carried out, and the resulting output including the domain-specific concepts and terminology.

3.1 Application Analysis

The requirement for a DSL is often only recognised after substantial development of domain-specific software in the field using a GPL. We examined applications in the domain in order to assess both the common abstractions and the crosscutting nature of domain-specific functionality. The reoccurrence of functionality and the repetition of domain-specific tasks throughout the applications results in the identification of domain-specific concerns.

In the pervasive healthcare domain, various applications were examined and a scenario was also implemented. Applications were examined against the previously described set of ubiquitous computing concerns that have been shown to be crosscutting. Figure 1 illustrates the classification of applications according to the concerns they incorporate. Applications are generally either patient or health care professional based i.e., used in the home or within a hospital environment. Hospital based applications [3] [4] were found to include most ubiquitous computing concerns. The communication concern heading in Figure 1 implies the potential requirement for healthcare specific messaging formats and protocols as the standards outline communication protocols. A scenario depicting an intelligent pervasive healthcare application [6] was extended to make use of HL7 messaging standards. Implementation confirmed that

[1] http://www.hl7.org/



		Reasoning	Security	Mobility	Communication	Quality of Service	Device Discovery	Service Discovery	Context Acquisition	Location	Proximity	Time	
Hospital	Facilitating Dynamic Scheduling	●	●	●	●	●			●	●	●	●	●
Home	Aware Home	●							●	●			
Home	Multimodel &Ubicom for Older Users	●			●				●	●			
Home	Autominder	●		●					●			●	
Home	Orientation Aid for Amnesics	●		●					●	●		●	●
Home	Flexible Technologies & Smart Clothing	●							●	●			●
Hospital	Mobility in Healthcare by Application Roaming among Heterogeneous Devices	●	●	●	●	●	●	●	●	●	●		●

Figure 1. Concern Identification in Domain Analysis

similar concerns were again present. These commonalities in concerns form the basis for abstractions in the DSL.

3.2 Concepts and Terminology

To identify the terms and concepts to be used in the DSL, existing representations of entities and tasks in the domain were examined. SOUPA [7] defines a standard ontology for pervasive and ubiquitous computing applications. It describes core entities including person, event, action, time and space and location. Ontologies have been used to establish semantic models in a domain [8]. The basic concepts represented are actor, location, time, activity, and device. The terms used in these ontologies, along with the previously identified concerns, create the basis for terminology used in the vocabulary of the DSL.

4. Prototypical Implementation

The aim of ALPH is to provide a complete DSL, including a comprehensive library of high-level constructs and a framework including a library of concerns to carry out the domain-specific functionality. This paper presents an initial implementation as an example of how ALPH raises the semantic level of programming using higher-level notations. It is a limited prototypical implementation and does not cover many of the deeper issues involved in DSL creation. ALPH is mapped to an aspect language so the resulting GPL translation may be weaved into the base application. The initial implementation is mapped to AspectJ, and deals only with the notions of aspects, pointcuts, joinpoints and advice. It does not address any other available AspectJ functionality.

4.1 Design

We describe the design of our implementation in two parts. The first describes a high level view of how the domain-specific constructs are translated to executable GPL code using a translator. The second describes how the translator itself is created.

4.1.1 DSL Translation

The constructs available to the developer include a subset of the concepts and terminology discussed in Section 3.1.2. The application developer implements the base application in a GPL without having to include the crosscutting concerns. The crosscutting sections are then implemented using ALPH. The developer's ALPH program is then translated into an aspect language using a provided dynamically extensible translator [1]. The result is one or more compiled aspects, which are then weaved into the base application on execution using the aspect language's existing weaver. The aspects may contain generated code from a library of existing aspects of crosscutting concerns.

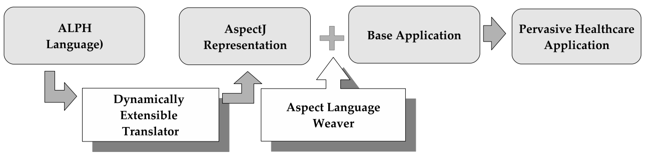


Figure 2. DSL Translation

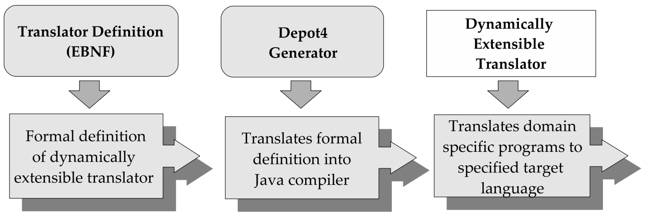

Translator Definition (EBNF)
Depot4 Generator
Dynamically Extensible Translator
Formal definition of dynamically extensible translator
Translates formal definition into Java compiler
Translates domain specific programs to specified target language

Figure 3. Translator Generation

The end product is a complete pervasive healthcare application as illustrated in Figure 2. The implementation details of this step are described in Section 4.2.

4.1.2 Translator Generation

The translator that carries out the translation from the domain-specific constructs to the GPL must first be created as shown in Figure 3. It requires the ability to recognise what functionality each construct is mapped to and what output to produce. A translator generator Depot4 [1] was used for the creation of the ALPH translator. Depot4 is a pre-processing metasystem that generates dynamically extensible translators for DSLs. Depot4 requires a formal definition of the DSL grammar in order to create an appropriate translator. It does this by means of the metalanguage Ml4, an extension of Extended Backus-Naur Form (EBNF) [9]. Once a formal definition of the DSL and its semantics are provided, Depot4 generates the translator as a collection of Java classes. These classes are invoked when the developer translates the domain-specific constructs. This then triggers the process described in Section 4.1.1 i.e., the DSL is translated into an aspect language.

4.2 Implementation

This example implementation illustrates how ALPH might provide domain-specific constructs to pervasive healthcare application developers.

4.2.1 Grammar Definition

Domain-specific constructs must be transformed from their high level state to a target GPL or output. This is performed by means of the translator generated according to a formal definition as described in Section 4.1.2. A section of the metalanguage formal description of the example ALPH translator is illustrated below in Listing 1.

```
ALPH =  lang -> lang_.
lang = aspect func -> '\n ' aspect_ func_.
 assign = ident [',' assign] ->  [ident_ ';'] ident_.
 aspect = assign -> '\n public aspect ' assign_ '{' .
 func = assign actor -> '\n pointcut ' assign_ '():
      execution(* ' actor_ '(..)); ' .
```

Listing 1. Translation Description in the Ml4 Metalanguage

4.2.2 Dynamically Extensible Translator

The formal definition shown in Listing 1 is used by the translator generator to construct the ALPH translator. The output is a compiled Java translator which is then used to translate the domain-specific constructs into the target general purpose language, in this example, AspectJ. A section of the generated code is shown in Listing 2. Extensibility is one of the defining features of Depot4. The newly created language, and its associated translator, can be easily extended by adding the new functionality to the formal definition. This is crucial in DSL development as new constructs may be required as the domain widens and advances.

```
public final class ALPH extends  Dp4.NTprocedure {
  protected ALPH(int i){
    this.m= i;
  }
     public void init(Dp4.Translator translator){
    ALPH[] y_ntp_= new ALPH[1];
    for (int i= 0;i<y_ntp_.length; i++)
      y_ntp_[i]= new ALPH(i);
    translator.installNT("ALPH", y_ntp_[0]);
    for (int i= 0;i<y_ntp_.length; i++){
      y_ntp_[i].y_translator= translator;
      y_ntp_[i].y_nt_ALPH= translator.registNT("ALPH");
      y_ntp_[i].y_nt_lang= translator.registNT("lang");
      y_ntp_[i].y_ntp_ALPH= y_ntp_[0];
 }

     }
```

Listing 2. The ALPH Dynamically Extensible Translator

4.2.3 ALPH Language

The ALPH language itself is comprised of the terminology discussed in Section 3.1.2. The constructs include tasks and entities specific to the pervasive healthcare domain. An example of a small program is illustrated in Listing 3. The constructs in Listing 3 correspond to some of the implemented concepts discussed in Section 3.2. The location indicates that location functionality is required. This is provided by means of an aspect. The translator generates a location aspect and according to the language constructs, inserts the appropriate code into the generated aspect. Some constructs may trigger the use of code from an available library of aspects that provide crosscutting ubiquitous concern functionality.

```
location changed patient
```

Listing 3. Example ALPH Program

This prototypical implementation is limited in its functionality. The language itself and the constructs it provides are evolving. Constructs corresponding to many of the identified domain tasks and entities in are envisaged as in Listing 4.

```
discover location devices

crosscheck prescription

update EHR

create HL7 dischargeNotification
```

Listing 4. Example ALPH Program

Discover represents the need for service and device discovery. Using the DSL construct, the discovery functionality will be included in the resulting aspect. The *location* instruction causes location monitoring to be included as discovery will be carried out within the immediate proximity. The type of discovery that is required, and the objects that will be returned, are denoted by *devices*. The *crosscheck* command invokes reasoning and rule based decision functionality. Both *EHR* and *HL7* objects are available for creation and manipulation. These constructs illustrate the behaviour that can be incorporated using a language that not only offers higher level domain-specific constructs, but also modularises concerns that crosscut entire applications.

4.2.4 Translated Output

The output delivered following the translation of the DSL constructs is, in this example, an AspectJ aspect. The example output is illustrated in Listing 5. The translation to a concrete GPL is defined in the formal definition provided to the Depot4 translator generator. Various definitions to multiple GPLs can be provided enabling the developer to use a choice of base application development languages.

```
public aspect location{
pointcut changed():execution(* *patient(..));
```

Listing 5. Translated Output

4.2.5 Weaving

Weaving is carried out by the aspect languages existing compiler, in this case AspectJ's. The translated aspect is weaved into the base application at the points in execution specified by the appropriate DSL constructs. As a result the base application executes with the required pervasive healthcare functionality included.

5. Related Work

MUMPS, also known as M [10] is a DSL developed to enable healthcare applications access databases and utilise resources efficiently. It was widely used in the 70's and 80's and is still used in newer implementations today. MUMPS provided database specific functionality that was useful for its initial purpose of healthcare applications, however, it did not support enough healthcare specific abstractions and was used as a database DSL. Bardram and Christensen [5] propose a middleware for clinical based applications that addresses many ubiquitous computing concerns including mobility, heterogeneous devices, discovery and security. However, it doesnt provide any higher level constructs with more intuitive semantic meaning for the application developer. YABS [11] is a meta-level domain specific language for pervasive computing. It provides means for defining and coordinating the behaviour of entities in pervasive environments. Concerns including mobility and adaptation are addressed. An interpreter takes the defined behaviours via a script and translates them into intermediate level objects. YABS focuses on the composition of components in pervasive environments rather than the provision of a domain-specific language for crosscutting concerns. Executable use cases have been used to describe the requirements for the pervasive healthcare domain [13]. Context-awareness, propositioning and non-intrusiveness are design principles that are suggested to be requirements for pervasive application development. This supports our domain analysis but lacks the progression to higher level abstractions.

The International Classification of Functioning, Disability and Health (ICF) [14] is a framework for the classification of health and disability. It provides a common language for disability description but is specific to medical terminology and does not address any technological issues.

6. Conclusions

Pervasive healthcare has many crosscutting domain-specific concerns. DSLs ease application development by providing a level of abstraction through more expressive domain-specific constructs [12]. We propose a DSL with constructs that specifically target domain-specific crosscutting concerns. This removed the need for the implementation of crosscutting concerns throughout the base application, increasing modularity. We demonstrate an example implementation of ALPH which uses a translator to transform the DSL into a chosen target language, in this case, an aspect language.

References

[1] J. Lampe. Depot4 - A generator for dynamically extensible translators. Software - Concepts and Tools, 19:97-108, 1998.

[2] Neil Loughran et al, A domain analysis of key concerns - known and new candidates, KUL Leuven, AOSD-Europe Deliverable D43, AOSD-Europe, February 2006

[3] C. Driver et al, Facilitating Dynamic Schedules for Healthcare Professionals, 1st International Conference on Pervasive Computing Technologies for Healthcare, Innsbruck, Austria, 2006, Nov, IEEE.

[4] Jakob Bardram et al, Supporting Local Mobility in Healthcare by Application Roaming Among Heterogeneous Devices, Lecture Notes in Computer Science, Volume 2795/2003, 161-176, October 2003.

[5] Jakob E. Bardram, Henrik Brbak Christensen. Middleware for Pervasive Healthcare, A White Paper. In G. Banavar, editor, Middleware for Mobile Computing, Heidelberg, Germany, 2001.

[6] J.E.Bardram, T.R.Hansen, M.Mogensen, M.Soegaard. Experiences from Real-World Deployment of Context-Aware Technologies in a Hospital Environment. Ubicomp 2006, pages 369-386, CA, USA, Sep 2006.

[7] Chen, H., Perich, F., Finin, T. and Joshi, A. (2004) 'SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications', In International Conference on Mobile and Ubiquitous Systems: Networking and Services, Boston, MA.

[8] Renato de Freitas Bulco Neto, Maria da Graa Campos Pimentel, Toward a Domain-Independent Semantic Model for Context-Aware Computing, la-web , pp. 61-70, 2005.

[9] Wirth, N., What Can We Do about the Unnecessary Diversity of Notation for Syntactic Denitions? CACM 20: 11, pp. 822 -823 (1977).

[10] Bowie, J., Barnett, G. O. MUMPS–an economical and efficient time-sharing system for information management. Comput Programs Biomed. 1976. Apr;6(1):11-22

[11] Barron, P. and Cahill, V., YABS: a domain-specific language for pervasive computing based on Stigmergy 5th International Conference on Generative Programming and Component Engineering, Portland, Oregon, USA, October 22 - 26, 2006.

[12] T. Sloane M. Mernik, J. Heering. When and how to develop domain-specific languages. Technical Report, SEN-E0309, CWI, 2003.

[13] Jrgensen, J. B. and Bossen, C. 2003. Requirements Engineering for a Pervasive Health Care System. In Proceedings of the 11th IEEE international Conference on Requirements Engineering (September 08 - 12, 2003). RE. IEEE Computer Society, Washington, DC, 55.

[14] International Classification of Functioning, Disability and Health: ICF. Geneva: WHO, 2001.

[15] Cysneiros, L. M. 2002. Requirements Engineering in the Health Care Domain. In Proceedings of the 10th Anniversary IEEE Joint international Conference on Requirements Engineering (September 09 - 13, 2002). RE. IEEE Computer Society, Washington, DC, 350-356.

[16] A. Schmidt, M. Beigl and H.W. Gellersen, There is more to context than location, Computer and Graphics 23(6) (December 1999) 893-901.