A Domain-Specific Language for Ubiquitous Healthcare

Jennifer Munnelly Siobhán Clarke
Trinity College Dublin
munnelj, sclarke @cs.tcd.ie

Abstract

The development of ubiquitous healthcare applications has proved to be significantly more complex than traditional healthcare applications. In software engineering research, there are two approaches of interest to us for handling the kind of complexity that emerges. The first is the use of domain-specific languages, which abstracts the low-level domain knowledge required when using general-purpose programming languages into more expressive domain-specific constructs. The second is advanced modularity techniques, such as aspect-oriented programming, that provide for modularisation of concerns that complicate code by cutting across a broad code base and tangling with other concerns. In this paper, we identify a set of ubiquitous healthcare concerns that complicate their software development. We use advanced modularity techniques to provide good separation of these concerns and encapsulate their behaviour within a new domain-specific language, ALPH that provides the application developer with a high level of abstraction. The result is a means to develop ubiquitous healthcare applications more easily and in a more timely fashion, while improving software quality by increasing modularity in the code.

Keywords: Programming Languages, Domain-Specific Languages, Ubiquitous Healthcare

1 Introduction

The emerging discipline of ubiquitous healthcare applies ubiquitous computing features to applications deployed in the healthcare domain. However, advances in ubiquitous computing e.g., mobile devices and network technologies, have not been matched with a corresponding proliferation of ubiquitous healthcare applications. These applications must support not only the information management associated with healthcare applications, but must also incorporate ubiquitous computing concerns to enable distributed, adaptive applications. The development of such applications has proved significantly more complex than traditional healthcare applications due to the number of complex components and their interconnections.

To address the difficulties in ubiquitous healthcare application development, we tackle two sources of complexity and provide solutions to resolve difficulties in development. Many ubiquitous healthcare concerns cut across the entire system compromising the encapsulation of concerns. These concerns are referred to as "crosscutting concerns" and are difficult to modularise using traditional programming models. The lack of modularisation leads to complicated, unmanageable code. Complexity in such applications can be managed by the software engineering principle of "separation of concerns". The aspect-oriented programming (AOP) [6] paradigm exhibits modularisation capabilities that resolve traditional difficulties, enabling the clean encapsulation of crosscutting concerns.

Complexity is also aggravated by the implementation of applications using general purpose languages (GPL) whose constructs tend to be at a low-level of abstraction. This means that developers need significant domain knowledge to produce the required verbose, low-level code that is neither expressive nor semantically intuitive. Domain-specific languages (DSLs) address the issues of low-level syntax and lacking expressiveness by enabling application developers to programme at a high-level of abstraction using constructs

that encapsulate domain knowledge.

We identify a set of ubiquitous healthcare concerns that reoccur in applications and that have exhibited crosscutting characteristics. We make these encapsulated concerns available by means of a domain-specific language, ALPH [10]. ALPH defines a set of domain-specific constructs that model tasks and entities in the ubiquitous healthcare domain. These semantically intuitive constructs trigger the inclusion of ubiquitous healthcare behaviour throughout the application by means of construct compilation, generative programming and aspect-oriented programming techniques. The result is a means to develop ubiquitous healthcare applications more easily and in a more timely fashion, while improving software quality by increasing modularity in the code. The proposed approach is evaluated by the comparison of a case study ubiquitous healthcare application, MedHCP, which has been implemented in two ways; using traditional GPL developmental techniques and using the proposed ALPH approach. The application was deployed on the Motion C5 mobile clinical assistant (MCA) by Intel. Modularity and abstractness are measured using the Goal-Question-Metric (GQM) [3] approach to evaluate the effectiveness of the proposed approach.

2 Ubiquitous Healthcare Concerns

We performed a comprehensive domain analysis to identify functionality that increased complexity when incorporated in applications. A literature study, application development and the examination of codebases enabled the identification of a set of concerns classified as "crosscutting" i.e., affecting multiple implementation modules. The resulting comprehensive list of concerns takes into consideration many varied applications making use of a subset of the identified concerns.

2.1 Ubiquitous Computing Concerns

During our domain analysis of ubiquitous healthcare applications it emerged that concerns relating to mobility [7], context-awareness [12] and infrastructure [11] were most significant.

Mobility
Mobility requires that applications deal with the inherent unreliability of wireless communications and the possibility of disconnected operation. We have identified eight primary mobility concerns that require support by developers. They are distribution, roaming, service discovery, device discovery, ad-hoc networking, limited connectivity, location, proximity and quality of service. Distribution addresses the geographical and technical dispersion of nodes and the underlying communication mechanisms used to support the transfer of data between nodes. Roaming refers to the movement of a user involving changes in network or devices. Discovery handles the location and knowledge management of other nodes and services. Ad-hoc networking addresses the routing of data between nodes in an ad-hoc environment. Limited connectivity deals with the unreliability of network connections in distributed mobile environments and the contingency plans required on disconnection. Location addresses node position and the technologies used to acquire such information. Proximity is closely linked with location with a focus on communication with nodes in the immediate vicinity. Quality of service refers to the capabilities and resources associated with particular networks and the management of application quality of service requirements.

Context-Awareness
Context-awareness enables applications to adapt their behaviour in response to changes in the deployment environment, enabling transparent self-adaptive personalisation to take place. Following the study of context-aware applications, we identify the requirement for support of different types of context information [7]. We divided the overall context space into eight sub-categories: device, location, user, social, environmental, system, temporal, and application-specific context.

Infrastructure
The infrastructure required by ubiquitous healthcare applications below the application layer entails the provision of fundamental communication mechanisms, systems software and resource and network management. It also considers adaptations that may occur in response to changes in system context i.e., performance, hardware availability and network conditions [12]. The identified concerns of roaming, discovery, ad-hoc networking, limited connectivity, quality of service and system context adaptation provide the fundamental infrastructural architecture required by applications in a ubiquitous healthcare environment.

2.2 Healthcare Concerns

In analysis, we encountered a number of healthcare specific crosscutting concerns common to ubiquitous healthcare applications. The Health Level Seven (HL7)[1] international standards institution promotes

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

and enforces the standardisation of electronic healthcare information to facilitate its exchange and management. Incorporation throughout the application introduces a large amount of crosscutting code related to messaging formats that are applied. Information management practices should also be considered such as the use of Electronic Health Records (EHRs) i.e., full patient records consisting of data from various hospital and healthcare systems. EHR management is intrinsically embedded in the routine functionality of ubiquitous healthcare applications i.e., the manipulation of patient records. The provision of new EHR opportunities such as Google Health and Microsoft's HealthVault reinforces the requirement for modular inclusion enabling EHR systems to be pluggable components. Persistence related functionality is also a prevalent concern in healthcare systems affecting multiple modules. In ubiquitous healthcare applications, remote connections are required throughout the application to persist this data. Context reasoning has also emerged as a frequent crosscutting concern in ubiquitous healthcare applications e.g., cross-checking prescribed medicines with medicines administered and inferring diagnosis from symptoms.

3 Separation of Concerns

Management of complexity in software has traditionally been influenced by the software engineering principle of "separation of concerns". However, many ubiquitous computing applications' challenges will cut across the entire system, compromising encapsulation of concerns. Current development technologies lack the necessary support to allow the developer to reason about crosscutting concerns separately from the rest of the system, resulting in software that is poorly modularised and therefore inherently more complex. Poorly modularised code reduces the quality of the software, as evident by its negative affects on software qualities such as maintainability, comprehensibility, manageability, scalability and reusability. These software quality factors are a concrete measurement of how complexity directly affects the developer, underlining the causes of untimely, deficient implementations.

Aspect-oriented programming (AOP) provides advanced modularity techniques to address the separation of crosscutting concerns. Concerns are modularised into "aspects" whose behaviours are triggered at various points in the application. We exploit the modularisation capabilities of AOP to encapsulate ubiquitous healthcare concerns to improve modularity, therefore reducing application complexity. In previous work we have established the positive increase in software quality using AOP in the modularisation of context adaptation [12] and infrastructure [11]. A library of aspects which provide the behaviour of ubiquitous healthcare concerns is implemented in an aspect language, providing the developer with reusable components for ubiquitous computing and healthcare functionality. The library of modularised concerns is made available to the developer via a domain-specific language.

4 ALPH

DSLs provide the means to program efficiently within a particular domain by addressing the issues of low-level syntax and lack of expressiveness that arise when using GPLs. Domain-specific constructs encapsulate domain knowledge to provide high-level abstractions aiding the development of domain-specific tasks and entities in a concise manner. The result is more expressive, semantically representative code that eases application development and reduces the amount of domain-specific knowledge required by the developer.

ALPH , "Aspect Language for Pervasive Healthcare", provides a domain-specific aspect language (DSAL) that includes a comprehensive set of extensible high-level constructs. Use of any construct by a programmer initiates the inclusion of ubiquitous healthcare concern implementations from a library. ALPH is a declarative language implemented as a pre-processor to an existing aspect language, in this implementation AspectJ [1]. This section describes how ALPH is implemented, and how the semantic, high-level constructs are mapped to lower-level application code.

4.1 Language

The constructs provided in the ALPH language model concerns specific to the ubiquitous healthcare domain, and are terms from common domain terminology. The application developer creates an ALPH file by using the available constructs as required. The constructs are parsed by the ALPH compiler and the generative compilation process triggers the use of code from the library of aspects that provide ubiquitous healthcare functionality.

```
1 discover (device, device, .new).
2 discover (service, service, PRINTING_SERVICE).
```

Discovery Construct

As an example of how an ALPH construct can be used, we describe the use of the *discover* construct which represents the concerns of service and device discovery. The *discover* construct prompts discovery

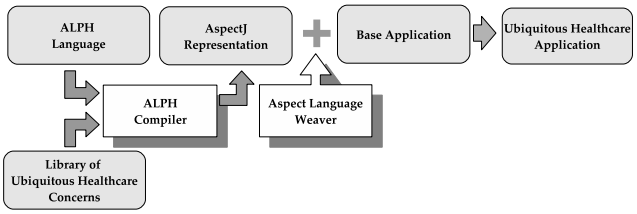

Figure 1. Application Development

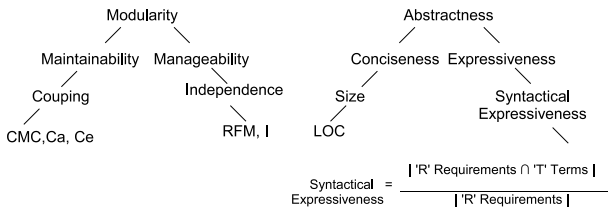

Figure 2. Goal-Question-Metric Model

functionality to be included in the application. Parameterisation enables the choice of discovery protocol, specification of services and indication of naming conventions required for the selection of points in the base code where crosscutting behaviour must be inserted. Listing 1 illustrates the use of the *discover* construct in two ways: to include device functionality at the construction of each new object, and to include service discovery at each method that has "service" in the declaration and to search for the service named PRINTING_SERVICE.

4.2 Application Development

The application developer implements the base application in a GPL. Ubiquitous healthcare concerns and the trigger points in the base application are defined using the provided domain-specific constructs in an ALPH program. The developer's ALPH program is then translated into an aspect language by the ALPH compiler. The result is one or more aspects containing the ubiquitous healthcare behaviour from the library of concern implementations, which are weaved into the base application using the aspect language weaver. The result is a complete compiled and executable ubiquitous healthcare application as illustrated in Figure 1.

4.3 Extensibility

Few domains remain static over time leaving DSLs susceptible to becoming obsolete. ALPH addresses extensibility in three ways to ensure new constructs can be added as the domain of ubiquitous healthcare widens and advances. The language, and its compiler, can be extended by adding new functionality to the formal definition of the language syntax and semantics. The aspect library must then be extended with code to support the new construct. Construct parameterisation also provides a means to customise a construct's behaviour e.g., distribution (RMI, JMS, Sockets). Finally, the translation of ALPH to a concrete GPL is defined in the formal definition provided to the compiler generator. To allow a choice of base languages for application developers, definitions can be provided to translate ALPH into multiple GPLs e.g., implementations provided in a C based aspect language would allow for the use of C as a base language.

5 Evaluation

The goals of this work are the separation of ubiquitous healthcare concerns to increase modularity and hence decrease complexity, and to provide the developer with a higher-level of abstraction resulting in concise, expressive code. Both goals contribute to the objective of reducing development effort in ubiquitous healthcare applications, which would encourage their proliferation.

5.1 Goal-Question-Metric

The goal-question-metric (GQM) [3] approach enables the quantitative evaluation of high-level conceptual characteristics e.g., our goals of increased modularity and high-level abstraction. Using GQM we map measurable quantitative-level metrics to operational level questions, which in turn map to conceptual level goals.

Figure 2 illustrates the GQM approach applied to modularity and abstractness. Modularity involves the breaking up of an application into smaller, more independent elements known as modules. The operational level questions we use to model our goal of modularity are based on Parnas' benefits of modular programming [13] namely manageability and maintainability. Maintainability relates to the modifiability of the code base and is negatively affected by dependencies between modules i.e., coupling [2]. Manageability is measured by metrics that can identify the level of autonomy of a module, enabling the module to be developed and modified in isolation e.g., independence metrics [3].Abstractness is achieved by providing the developer

[2]Coupling on method call (CMC), afferent/efferent (Ca)(Ce)
[3]Response for a module (RFM) and instability (I)

with concise, expressive constructs which shield low-level functionality. The goal of abstractness is therefore modeled by the operational level questions of expressiveness and conciseness. The question of conciseness is answered by straightforward size metric, lines of code (LOC). We define expressiveness in a programming language as the ability of a language to allow the developer to perform tasks using natural, semantically intuitive syntax. The syntactic expressiveness of a language can be defined as its ability to provide constructs that enable the developer to adequately fulfill the requirements of an application [5]. Fabbrini [8] suggests a relationship between the language used in the requirements and the ability of people to understand the language. We use a quantitative level metric based on Zipf's law [15] to measure syntactic expressiveness.

Figure 3. Mobile Clinical Assistant

5.2 Case Study

To evaluate the contribution of the library of ubiquitous healthcare concerns and the DSL that enables their incorporation, we implemented a case study application known as MedHCP based on a scenario from the ubiquitous healthcare domain [14]. The scenario was conceived by the Centre for Pervasive Computing and staff at a collaborating hospital in Aarhus county, Denmark. The scenario depicts an ideal use for mobile and pervasive computing devices in the medication of patients in a hospital setting and requires support for many of our identified ubiquitous healthcare concerns. The MedHCP application was implemented using a traditional object-oriented approach and also using the proposed ALPH approach. The object-oriented approach used a general purpose language, Java, for the implementation of all functionality. The ALPH approach was implemented using ALPH to define the incorporation of the required ubiquitous healthcare functionality and Java for the base application functionality. The application was deployed on the Motion C5, the mobile clinical assistant created by the Digital Health Group (DHG) at Intel Health.

5.3 Results

This section summarises results of the evaluation using the GQM approach to deduce empirical measurements for our goals of modularity and abstractness.

Modularity

Maintainability: Coupling was reduced by the ALPH approach by 33-75% in modules most influenced by crosscutting code. 5 modules were found to be coupled with aspects after modularisation indicating that significant crosscutting code existed that profited from modularisation. Dependencies on external modules were reduced by up to 40% by the ALPH approach. These illustrate the beneficial effect of modularising these concerns away from base functionality as modules are more easily maintained when coupled with fewer modules. Inward dependencies were increased by the ALPH approach due to the aspect's dependencies on the syntax of base code, but in our opinion are offset by the modularisation achieved.

Manageability: Dependencies were reduced by 20% in one module and 60% in two modules using ALPH. Fewer dependencies indicate that modules can be developed and managed in isolation more easily. Stability is also an indicator of manageability and was increased in all modules using the ALPH approach, indicating a reduction in the modules' resilience to change.

Abstractness

Conciseness: ALPH reduces application size experienced by the application developer i.e., before compilation, by 25%. This is achieved by the conciseness of domain-specific constructs and their ability to produce domain-specific behaviour through declarative, generative constructs.

Expressiveness: Constructs in ALPH can fulfill 50% of domain-specific requirements by only 20% of action terms from the domain. The demonstration that by including the most frequently used terms in the domain in ALPH, the expressiveness of the language is increased as according to Ziph's law, a large amount of behaviour in the domain is likely to use these few terms. We can also conclude that from the achievement of a more concise language, expressiveness is also achieved as the same semantics can be expressed by a shorter body of material.

6 Related Work

MUMPS [4] is a DSL developed to enable healthcare applications manage information. However, it did not support healthcare functionality and was used as a database DSL. Middleware has been proposed for clinical based applications that address many ubiquitous computing concerns including mobility, heterogeneous devices, discovery and security [9]. However, it does not provide any higher level constructs with intuitive semantic meaning for the application developer. YABS [2] is a DSL for pervasive computing that provides means for defining and coordinating the behaviour of entities in pervasive environments. YABS focuses on the composition of components in pervasive environments rather than the provision of a domain-specific language for crosscutting concerns. Context-awareness, propositioning and non-intrusiveness are design principles that are suggested to be requirements for ubiquitous application development. This supports our domain analysis but lacks the progression to higher level abstractions. The World Health Organisation's International Classification of Functioning, Disability and Health (ICF) provides a common language for disability description but is specific to medical terminology and does not address any technological issues.

7 Conclusions

In this paper we present ALPH, a language for ubiquitous healthcare. As a DSAL, ALPH exhibits both the modularisation benefits of AOP and the abstractness, and resulting expressiveness and conciseness, of DSLs. Domain crosscutting behaviour is encapsulated as concerns in a library of aspects and made available through domain-specific constructs. Evaluation using the GQM model is based on a case study application, MedHCP, implemented using the proposed approach and traditional object-oriented techniques. Results illustrate that ALPH exhibits the benefits of improved modularity and a higher-level of abstraction due to the use of AOP and DSL principles.

Acknowledgments

The work described is funded by Science Foundation Ireland under the Research Frontiers Program.

References

[1] Aspectj. http://www.eclipse.org/aspectj/.

[2] P. Barron and V. Cahill. Yabs:: a domain-specific language for pervasive computing based on stigmergy. In *International conference on Generative programming and component engineering*, 2006.

[3] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. 1994.

[4] J. Bowie and G. O. Barnett. Mumps–an economical and efficient time-sharing system for information management. *Comput Programs Biomed*, 1976.

[5] S. Boyd, D. Zowghi, and A. Farroukh. Measuring the expressiveness of a constrained natural language: An empirical study. In *International Conference on Requirements Engineering*. IEEE, 2005.

[6] G. Kiczales et al. Aspect-oriented programming. In *European Conference on Object-Oriented Programming*, 1997.

[7] N. Loughran et al. A domain analysis of key concerns - known and new candidates. *AOSD-Europe, Deliverable D43*, 2006.

[8] F. Fabbrini, M. Fusani, V. Gervasi, S. Gnesi, and S. Ruggieri. Achieving quality in natural language requirements. In *11th International Software Quality Week*, 1998.

[9] H. B. Christensen J. E. Bardram. Middleware for pervasive healthcare. In *Middleware for Mobile Computing*, 2001.

[10] J. Munnelly and S. Clarke. Alph: a domain-specific language for crosscutting pervasive healthcare concerns. In *AOSD Workshop on Domain specific aspect languages*, 2007.

[11] J. Munnelly and S. Clarke. Infrastructure for ubiquitous computing: Improving quality with modularisation. In *AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*. ACM Press, 2008.

[12] J. Munnelly, S. Fritsch, and S. Clarke. An aspect-oriented approach to the modularisation of context. In *International Conference on Pervasive Computing and Communications*, 2007.

[13] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 1972.

[14] K. Raatikainen, H. B.Christensen, and T. Nakajima. Application requirements for middleware for mobile and pervasive systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2002.

[15] George K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley (Reading MA), 1949.