HL7 Healthcare Information Management Using Aspect-Oriented Programming

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

Abstract

Given the heterogeneity of healthcare software systems, data from each system is often incompatible inhibiting interoperability. To enable the sharing and exchange of healthcare information interoperability standards must be adhered to. Health Level Seven (HL7) is the international standards organisation that promotes and enforces the standardisation of electronic healthcare information to facilitate its exchange and management. Incorporating HL7 functionality into existing applications requires significant modification and intrusive extensions. Using Aspect-Oriented Programming (AOP), we can introduce HL7 functionality into existing applications without the requirement for refactoring or modification. HL7 data formatting affects multiple parts of an application and hence is a "crosscutting concern". These concerns which entwine with base functionality introduce complexity and reduce modularity. A second benefit of AOP is its advanced modularisation capabilities which are capable of modularising "crosscutting concerns". We illustrate the benefits of using AOP in HL7 by example and measure the effects of the approach on healthcare applications.

1. Introduction

Hospitals, clinics, healthcare organisations and patients themselves have specific interests in health related data, yet the systems supporting each one are markedly different. Data from each system is often incompatible with other systems. To enable the sharing and exchange of healthcare information that supports co-operative healthcare work interoperability standards must be adhered to.

Health Level Seven (HL7) is a standards organisation that aims to enable the global exchange of healthcare data. HL7 aims to provide a comprehensive framework (and related standards) for the exchange, integration, sharing and retrieval of electronic health information [7].

Healthcare systems are large, complex systems. The incorporation of HL7 into an existing application would require substantial refactoring and/or extension. The functionality to create, parse and manage HL7 messaging formats is required around all data creation and exchange. Additionally, this functionality cuts across the entire application compromising the encapsulation of concerns. Functionality of this form is known as "crosscutting" and is difficult to modularise using traditional programming paradigms. Poor modularisation leads to systems which are harder to understand, modify and reuse.

Aspect-Oriented Programming (AOP) [5] provides a means by which to introduce new behaviour into applications without the requirement for modification in the base application. This facility enables the extension of applications which manage healthcare information to conform to HL7 standards, allowing them to share and exchange information with other compliant health related systems. AOP also exhibits modularisation capabilities which are capable of modularising difficult crosscutting concerns.

We investigate the use of AOP in HL7 data management by implementing aspects to introduce the creation and parsing of HL7 messages within an application.

2. Health Level 7

The HL7 international standards institution promotes and enforces the standardisation of electronic healthcare information to facilitate its exchange and management. The implementation of the HL7 messaging standard will improve quality, efficiency and effectiveness of healthcare delivery and sharing medical information. Application developers in the area of distributed healthcare must incorporate HL7 in each system which requires the ability to exchange and share of data with internal and external systems.

3. Aspect-Oriented Programming

The Aspect-Oriented Programming paradigm is an extension to object-oriented programming. It facilitates the injection of behaviour at given points in the execution of an application. The application with which AOP interacts,

978-1-4244-4878-4/09/$25.00 ©2009 IEEE

referred to as the base application, is oblivious to the behaviour and has no reference to it within its codebase. This enables the introduction of new functionality into the execution of an application without the requirement to change the base application.

AOP also addresses the difficulties that arise in the separation of concerns while using traditional Object-Oriented (OO) techniques. While each class in OO aims to address a single concern, often there are concerns which are entwined with many classes. These concerns are difficult to modularise using OO techniques. AOP exhibits modularity capabilities for these types of concerns, known as "crosscutting concerns".

Crosscutting concerns can be modularised into "aspects" using an aspect language. AspectJ [4] is the most commonly used aspect language. These aspects define the crosscutting functionality in one component and also indicate where in the base application the crosscutting behaviour should occur. The aspects augment the basic application at the outlined appropriate points in the application's execution. To implement behaviour in AspectJ four key concepts must be understood; aspects, pointcuts, joinpoints and advice. The aspect itself encapsulates all aspect features. Operations known as joinpoints are points in program's execution where aspect behaviour may be applied. These may be calls or executions of methods, field accesses, constructors, etc. Pointcuts are predicates for the selection of join points. During execution, the joinpoints are subject to alterations from advice. Advice outlines the behaviour that will occur before, after or around when a joinpoint is matched to point in execution. This behaviour is fully removed from base application, resulting in modular code.

4. Modularisation of HL7

To enable the introduction of healthcare information exchange using HL7 in a modular fashion, we use the AOP approach to implement the most commonly used functionality e.g., creating HL7 messages and parsing HL7 messages.

Healthlink [2] is an electronic communications project funded by the Health Service Executive in Ireland. The project objective is to provide a healthcare communications network which interacts with primary care practitioners, acute hospitals and agencies. Healthlink facilitates data exchange using HL7 version 2.4 in XML format. We used Healthlink templates to create XML DOM handler classes to create and parse messages. Aspects are then designed to make use of these handler classes and to introduce the required behaviour into the base application at the appropriate points in execution. The customisation of these aspects enables the introduction of HL7 data management in any application allowing it to communicate with other participants in such a healthcare communications network.

HL7 version 2.4 includes over 70 message types. Many sections of the messages are constant, but each message has particular elements relating to its purpose e.g., an OMD_O03 message representing a Dietary Order Diet message needs to receive specific information, the most important being the diet order itself. To comprehensively support all message types an existing API can be used in conjunction with AOP. HAPI [1] is an open-source, object-oriented HL7 parser for Java and can be used to manage all forms of HL7 version 2.x messages. As HL7 is an evolving standard, newer versions of HL7 must also be supported. HL7 v3 includes more detail and complexity as messages are seen as interactions between elements. In order to fully support HL7 v3 the HL7 Java SIG Project API [3] can be used in the same way.

5. Example

To demonstrate the use of AOP in the incorporation of HL7 functionality, we implement a scenario from a hospital healthcare information system. On the discharge of a patient, a "Discharge Notification" HL7 message is created. This message can then be stored or dispatched to appropriate systems elsewhere. This scenario makes use of the HL7 template currently in use by the Irish Health Service.

The application carries out its usual procedure on the discharge of a patient as shown in Listing 1; here a discharge method is called with discharge details passed as parameters. To incorporate this information into a HL7 compatible message we create an "ADT_A03" message for Discharge/End Visit. An A03 event signals the end of a patient's stay in a healthcare facility.

```
public class Administration {
....
dischargepatient("218", "Smith", "Danny",
      "197508270000", "MainSt,NewTown,NewCounty,Ireland",
       "Nugent", "200901011000", "200901011800");
....
}
```

Healthcare Application Discharge

We create an aspect, Listing 2, which identifies the points in the healthcare application we want this message to be created i.e., when the discharge method is called. The pointcut is defined to capture the parameters of the method so that they can be used to populate the HL7 ADT_A03 message. The advice defines the behaviour that is triggered on the matching of the joinpoint defined in the pointcut. Here the discharge details from the application are passed to a class that constructs the HL7 message in XML format.

```
public aspect Hl7DischargeNotification {
....
```

```
        pointcut writeDischarge(String patientID, String
            surname, String firstName, String DOB,
            String address, String Doctor, String
            admitDate, String dischargeDate) : call(
            void dischargepatient(String, String,
            String , String , String , String , String,
            String)) && args(patientID, surname,
            firstName, DOB, address, Doctor, admitDate,
            dischargeDate);

        after(String patientID, String surname, String
            firstName, String DOB, String address,
            String Doctor, String admitDate, String
            dischargeDate): writeDischarge(patientID,
            surname, firstName, DOB, address, Doctor,
            admitDate, dischargeDate) {
                ....
                createHl7DischargeNotification();
        }

        public void createHl7DischargeNotification(){
                XMLCreator xmlCreator = XMLCreator.
                    getInstance();
                xmlCreator.buildXML(patientID, surname,
                    firstName, DOB, address, Doctor,
                    admitDate, dischargeDate);
        }
....
}
```

ADT_A03 Creation Aspect

The ADT_A03 thats is created is illustrated in Listing 3. The XML file created is compatible with HL7 version 2.4 as used by the Irish Health Service. This HL7 file can then be stored or dispatched as appropriate.

```
<?xml version="1.0" encoding="UTF-8"?>
<ADT_A03>
  <MSH> ....
     <!--Timestamp 1/1/09 6pm-->
        <MSH.7> <TS.1>200901011800</TS.1> </MSH.7>
           <!--Type of message and event e.g. ADT A03
               Discharge Notification-->
            <MSH.9>
               <MSG.1>ADT</MSG.1>
               <MSG.2>A03</MSG.2>
            </MSH.9>
  .... </MSH>
        ....
        <!--Patient -->
        <PID> <!--Patient Identifier List-->
          <PID.3> <!--Patient Id from Hospital System-->
            <CX.1>218</CX.1>
               <CX.4> <!--~Issuer of ID-->
                  <HD.1>Mater Hospital </HD.1>
                       <HD.2>908</HD.2>
                       <HD.3>HIPE</HD.3>
               </CX.4>
        .... </PID.3>
           <PID.5> <!--Patient Name-->
               <XPN.1> <FN.1>Smith</FN.1> </XPN.1>
               <XPN.2>Danny</XPN.2>
      .... </PID.5>
..... </ADT_A03>
```

ADT_A03

6. Evaluation

To evaluate how the proposed AOP approach addressed modularity, we have conducted a study to measure the effect of the approach on the modularity of applications. Modularisation can be defined as the separation of concerns in an application into smaller, more independent elements known as modules. Modular code reduces the complexity of applications and enables the modules to be developed in isolation as each concentrates and addresses a separate concern. We use the software engineering benefits identified by Parnas [8] as indicators of modularity namely; manageability, maintainability and comprehensibility. These "ilities" describe the ability of the software to be developed in components and composed easily, modified and extended easily and understood easily.

We measure the effects on the software using the AOP-Metric suite [1]. This suite provides aspect-oriented extensions to traditional object-oriented metrics.We use measures of coupling dependencies, instability, size and complexity to measure the proposed approach as a means of supporting healthcare information sharing and exchange in modular software.

We use an open-source product, "HL7 Browser" [2], to evaluate our approach. This existing application makes use of HL7 functionality. This functionality has been developed using standard object-oriented in Java. In this implementation the HL7 functionality is scattered throughout the application. We have refactored the HL7 functionality into an aspect using AspectJ and compare the two implementations to quantify the use of AOP in the modularisation of HL7 data management.

7. Results

The module most heavily affected by the refactoring of the HL7Browser application was reduced in size by 70% as shown by LOCC (Lines of Class Code). This illustrates the vast amount of HL7 related code that was entwined with the modules base functionality. Removing the HL7 concern positively affects encapsulation and cohesion. Using AOP, the module becomes more cohesive by addressing fewer concerns and hence is easier to understand by the developer.

The complexity of this module was also positively affected by the use of AOP. The WOM (Weighted Operations in Module) metric measures the number of operations in a given module. The AOP approach reduced the number of operations in the module by 27%, thus reducing complexity. Empirical studies show a good correlation between complexity and comprehensibility. Code that is easier to understand is therefore easier to maintain and update, benefiting both developers and the healthcare professionals using the software.

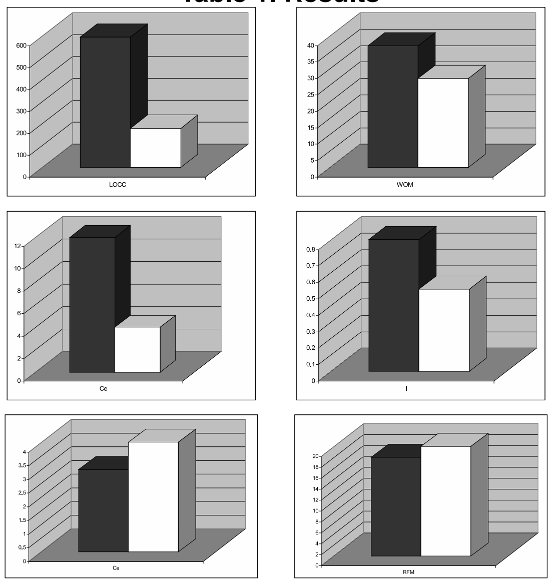
The package which contained the modules which were most significantly affected by the refactoring of the HL7

[1]http://aopmetrics.tigris.org/metrics.htm

[2]http://nule.org

functionality was also evaluated using package level metrics. Coupling in the package was reduced by 67%, as illustrated by Ce (Efferent Coupling). This reduction in coupling in the AOP implementation is due to the reduction in outgoing dependencies as the base application no longer makes calls related to the HL7 functionality. This reduction in coupling positively affects the maintainability of the software as changes do not incur a ”waterfall effect” i.e., a change in one module will require knock on changes in other modules. The stability of the package was also increased by the use of AOP by 37.5%, as shown by the decrease in the I (Instability) metric. Instability indicates the packages resilience to change. The package is more stable using AOP due to the removal of package dependencies on other modules for HL7 functionality.

Table 1. Results



The use of AOP does produce some negative results. While using AOP, pointcuts refer to syntactical references in the base code as indicators of where crosscutting behaviour is to be injected. These references introduce dependencies inwards on the base application which did not exist before the use of AOP. This is evident in this study by the increase in RFM (Response for a Module) coupling metric in a module by 11%. This measure indicates the number of potential advices that could be executed within the particular module. The dependency inward introduction of AOP can also be seen in the increase of Ca (Afferent Coupling) by 33%. Ca is a coupling measurement to quantify the number of modules outside a package that depend on modules within the package, and so is increased by the use of base application references when using AOP.

8. Conclusions

HL7 standards are a requirement in the facilitation of the sharing and exchange of healthcare information. The incorporation of these standards prove problematic due to the modification and intrusion on existing application code. Using object-oriented techniques, the introduction of such functionality reduces the modularity of software, reducing the quality and stability of the system. AOP enables the creation, parsing and exchange of HL7 data to be injected into applications without the requirement for refactoring. This eliminates the potential damage that may occur when modifying a fragile codebase. Additionally, AOP has the ability to introduce such functionality in a modular method, upholding the software engineering principle of ”separation of concerns”. We evaluate an object-oriented HL7 system against an AOP version. Results show that using AOP reduces coupling, reduces complexity and increases stability. These results produced code that is easier to maintain, manage and comprehend.

9. Related Work

A domain-specific language for pervasive healthcare that makes use of AOP for modularity has proved to be beneficial for the introduction of concerns including mobility, context awareness and healthcare concerns [6]. However, this work does not include an in-depth quantitaive analysis of the use of AOP in HL7 applications. Microsoft have implemented a software factory for HL7 [9]. This factory provides a model-driven approach to the development of HL7 components. This approach exclusively targets HL7 version 3, which many health systems have not adopted including the United States and Ireland.

References

[1] Hapi. http://hl7api.sourceforge.net/.
[2] Healthlink. http://www.healthlink.ie/.
[3] Hl7 java sig project api. http://aurora.regenstrief.org/javasig.
[4] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of aspectj. In *ECOOP ’01: Proceedings of the 15th European Conference on Object-Oriented Programming*, 2001.
[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. marc Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP*, 1997.
[6] J. Munnelly and S. Clarke. A domain-specific language for ubiquitous healthcare. volume 2, Oct. 2008.
[7] J. Murphy. International perspectives and initiatives. *Health Information and Libraries Journal*, 2007.
[8] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12), 1972.
[9] M. Regio and J. Greenfield. A software factory approach to hl7 version 3 solutions. Microsoft, White Paper.