Aspect Oriented Design Methods
Instant Messenger client case study
Jeroen Steenbeek
j.steenbeek@student.utwente.nl

ABSTRACT
This paper details a case study concerning 3 design techniques for aspect oriented programming. One for Requirements Engineering, one for Architectural design, and one for detail Design. This study takes a look at the flaws and merits of these design techniques and gives an advice for each method based on that. We do so by applying the design methods to a fictional instant messenger project.

Keywords
Aspect Oriented Programming, Requirements Engineering, Architectural Design, Detail Design

1. INTRODUCTION
A common problem in many software projects are so-called crosscutting concerns. Crosscutting concerns are concerns that exhibit scattering: their implementation is scattered across the program's source. The implementation needs changing, developers are forced to change this implementation throughout the program's source.

When dealing with multiple crosscutting concerns, we encounter a phenomenon known as code tangling. The implementation of various crosscutting concerns gets tangled, making the code both hard to read, and hard to maintain.

Aspect Oriented Software Development (AOSD, see [BD05]) is a technique that deals with this problem by isolating these crosscutting concerns into separate modules called aspects. AOSD usually complements existing programming techniques, such as object-oriented programming, as is the case with the AspectJ language. Aspect Oriented Software Development originated at the programming level of the design process.

At first, Aspect Oriented Programming was used mostly as a “patching mechanism”, to augment an existing program with a crosscutting concern. This is however not a very effective means of using Aspect Oriented Programming. A better alternative is to incorporate the notion of Aspect Orientation into our design process.

But is this early enough? When designing an application, we have to take the application's requirements into account. The designers will have to carefully look at the requirements to see if they can find anything crosscutting, and design the application appropriately. For this to be effective, there needs to be a way to easily find crosscutting concerns within requirements. A better way to deal with this problem is to identify crosscutting concerns while working on the requirements, which is the goal of Aspect Oriented Requirements Engineering (AORE).

There has been a lot of interest in both AORE and AOD, and a lot of research has been done in order to create techniques for performing either. Many techniques exist, for instance Theme/DOC [BC04] and Arcade [RMA03] for Requirements Engineering, and AOSDUC [JN05] and Theme/UML [CB05] for Design. Many techniques however focus only on a particular type of concern (in case of AOD), or focus only on a single type of requirement (in case of AORE).

When starting an AOP-based software project, one can choose from over a dozen techniques of designing an application and for refining his or her requirements, all with their merits and flaws. Choosing the correct technique for a specific situation can be time consuming. There is another way however. Recently the European Network of Excellence on Aspect-Oriented Software Development has been working on a unifying the various methods for both AORE and AOD, which has been published in 3 documents:

- A unified method for Aspect Oriented Requirements Engineering, which relies on natural language processing [CSR06]
- A generic method for Aspect Oriented Architectural Design, GAADA [KT06]
- A unified method for Aspect Oriented Design [JC06]

Each of these documents describes a certain phase of software development. The first document describes the requirements engineering phase. The second document describes the architectural design phase, and the last document describes detail design.

To see how well the 3 methods work on an actual software project, we will apply them to a fictional instant messenger project. The instant messenger application was chosen because this type of application is widely used by the general public, is of relatively small scale, and enables most developers to think along without much effort.

2. PROBLEM STATEMENT
The techniques described in [CSR06], [KT06] and [JC06] are fairly new, having been released only a few months ago. These methods have not yet been tested externally, that is, by people not directly involved in it's development.

A common dilemma in designing things is that a creator is always biased towards his or her creation, and while the authors of the 3 techniques have tested their own methods, it requires someone not directly involved in the creation of these methods to notice flaws a creator would not.

The goal of this study is to take a fictional software project, which will be outlined in paragraph 2.1, and perform each of the 3 methods on this case. The 3 methods have been designed to be used in succession, so we will start by performing [CSR06],
then continue with [KT06] and finish with [JC06].

After applying each of these 3 methods, we will give an advice for improving each of these methods.

2.1 Case

It is hard to imagine the modern-day internet without instant messengers. Programs like AIM and MSN are known worldwide, and used by millions of people. Both of these programs suffer from the same problems: they are both bloated with features, and their security is questionable.

Therefore we want to create a new instant messenger client that offers only basic features, while being very secure.

Each client maintains a list of contacts, and can only send messages to clients that are on their contact list, and online. In order to add a contact, that contact needs to approve being listed in the client's contact list.

A client can open a chat session with another client that is online through their contact list. Once established, additional members can be added to this session, provided they are in the contact list of one of the participating clients.

When there are exactly two clients in a chat session, a client should be able to send a file to the other client through the server.

All communication between two clients should go through a central server, the IP address of a client should remain hidden to other clients. All data transferred between a client and the server must be encrypted, so that the contents of the transmission remain secure.

The client should work on as many operating systems as possible, and should be easy to use.

Prior to seeing a contact list, the client should authenticate with the server, using a challenge/response type of authentication (no password is ever actually sent across the network). In order to change the password, or make an account to begin with, a website will be used.

2.2 Evaluation procedure

To make a good judgement about each of the discussed methods, we will carry out each method in 3 steps.

The first step is the learning phase, in which we will detail the process of learning a method, problems encountered with this method, and a general overview of how the method works.

Next is the execution phase, in which we will detail the process of applying the discussed method to our case study, what problems we encounter here, and our results.

Finally is the evaluation step, where we will discuss the problems encountered in the previous 2 steps, and give an initial verdict about the method.

3. RESEARCH

This section contains the results of applying the 3 methods to the Instant Messenger case, and details the learning, execution and evaluation of each of these methods.

3.1 Requirements Analysis

In this section we will describe the Requirements Analysis method described by Chitchyan, Sampaio, Rashid and others [CSR06]. We will detail my experience in learning this method, applying it, and my evaluation of the method based on this.

3.1.1 Learning

The requirements analysis method described by Chitchyan, Sampaio, Rashid and others [CSR06] is divided into several steps that must be taken in a specified order, as displayed in figure 1. The first step in this process is the Concern Elicitation phase, which is not visible in this figure.

Figure 1: Unified Aspect Oriented Requirements Analysis

The acronyms Early-AIM and EA-Miner stand for Early Aspect Identification Method and Early Aspect Miner, respectively.

Concern Elicitation

We take the description of the system (in this case the case description) and try to find relevant statements about the system.

Concern Identification

Here we analyse the statements we found in the elicitation phase, and try to structure the properties of our system into concerns.

Concern Representation

Once we have found concerns, we structure them in a logical fashion.

Requirement Composition

For each concern we have identified in the Concern Representation phase, we see which requirements (from the Concern Elicitation phase) apply to which concern.

Trade-Off Resolution

Parallel to the Requirements Composition process we resolve any trade-offs in requirements.

Requirement Refinement

Once we have represented concerns, we see how we can improve our representation, and how this affects the requirements.

Requirement Mapping

In this second to final phase, we map our requirements to an RDL model.

Link RE & Architecture

Not exactly a phase, but this is where the Architectural Design phase more or less starts. [CSR06] does not give any details on how this is achieved, but does mention there is no clearly defined border.

All these steps can be roughly divided into 2 categories, and [CSR06] suggests a different approach for each of these methods. The first of these methods requires the EA-Miner tool, and helps in identifying the concerns in a system description text by using Natural Language Processing.
The second method is the Composition Centric Approach. This approach introduces a specific notation for requirements named RDL (Requirements Description Language). RDL is an XML-based language. The requirements in RDL are grouped by concerns, and several concerns may share requirements. The key feature of the Composition Centric Approach are not the Concern structures, but as the name suggests, the Composition structures.

As we mentioned in the second to last paragraph, we require EA-Miner to carry out the first few phases. This tool is still in development however, and not yet available to the general public for the simple reason that it is not yet ready. The description given in [CSR06] shows it is an impressive tool, but since we are unable to use it for our case study we were forced to look for another way of completing the Concern Identification stage.

Fortunately, [CSR06] suggests several alternatives as input for the second category, among which is Theme/DOC, which is covered in detail in [CB05]. They do however warn that Theme/DOC is only suitable for functional requirements. Another drawback is that Theme/DOC does not aid in the Concern Elicitation phase. To overcome this problem we follow a conventional requirements engineering technique used in Object Oriented Software Development [LL01]. How it works exactly is detailed in the execution phase.

3.1.2 Execution
As we stated in the last section, we need to establish an initial set of requirements before being able to use Theme/DOC to find crosscutting relations. This process proved relatively simple. We analysed the case description, and copied each line that stated something about the system, and incorporated that as a requirement. After this we followed the method described in [LL01], and flagged each requirement as either functional, non-functional, or non-requirement (for instance: a statement about the system that should be decided at the design level, not the requirements level).

There were three non-functional requirements:

1. Ease of use – This requirement primarily affects the design of the user interface, though design decisions about other parts of the system may complicate this.
2. Portability – The impact of this requirement depends on our choice of language. For simplicity's sake, we will assume AspectJ as implementation language since it will ensure a high portability regardless of design.
3. Security – This requirement has a large impact on the system, but also has many functional subrequirements. The presence of this requirement will not affect our ability to use Theme/DOC.

To further refine our set of requirements, one of the requirements was simply another way to rephrase another, so the final set of requirements is as follows:

1. Each client has a list of contacts
2. A client can only send messages to clients that are on their contact list
3. A client can only send messages to clients that are online
4. In order to add a contact to a contact list, that contact needs to approve being listed in the client's contact list
5. A client can open a chat session with another client, this starts a session between the two clients
6. Once established, additional members can be added to this session, provided they are in the contact list of one of the participating clients
7. When there are exactly two clients in a chat session, a client should be able to send a file to the other client
8. The IP address of a client should remain hidden to other clients
9. All data transferred must be encrypted
10. Prior to seeing a contact list and being able to send messages, the client must authenticate

These steps are the concern elicitation stage, so we now move to the concern identification phase. For this we employ Theme/DOC. To put it simply: we draw a rounded rectangle around each requirement, and we draw a diamond for each candidate concern that requirement affects (initially: each noun is a candidate concern – EA-Miner seems to function in a similar way). A rounded rectangle is used to denote a requirement, and a diamond is used to denote a concern.

After this we draw lines to indicate relationships between concerns and requirements. The resulting diagram will likely be in need of further refining.

We have now basically done the Concern Representation as well, so now we take a look at what we made and try to make improvements. The first thing we notice is that requirements 2 and 3 both impose limitations on the process of sending messages. It is therefore logical to group these requirements together under a single message restriction requirement, with the 2 restrictions as subrequirements. We also notice that much of the functionality depends on encrypted communication, so the encryption concern crosscuts all communication. Another crosscutting concern is Authentication, since all useful actions depend on the user being authenticated.

Both Authentication and Encryption can be seen as specializations of the concern Security.

Theme/DOC distinguishes between Themes and Requirements, and they can almost directly be mapped to RDL. A Theme becomes a concern, and encapsulates any requirement it has a relation with. Some requirements can not be expressed in this way however, and must be expressed using a Composition. A good example is authenticating prior to anything else.

3.1.3 Evaluation
The Unified Aspect Oriented Requirements Analysis method has a number of flaws and merits, which we will briefly discuss here.

<table>
<thead>
<tr>
<th>Merits</th>
<th>Flaws</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA-Miner seems to be an excellent tool to automate the otherwise tedious task of generating an initial set of requirements</td>
<td>EA-Miner is not available to the public yet</td>
</tr>
<tr>
<td>The Requirements Description Language (RDL) requires only minor adaptations to be used for concepts such as Themes rather than Concerns.</td>
<td>Learning Curve: To those unfamiliar with Natural Language processing this approach is hard to master. Lack of EA-Miner leads to</td>
</tr>
</tbody>
</table>
### Merits

<table>
<thead>
<tr>
<th>Flaws</th>
</tr>
</thead>
<tbody>
<tr>
<td>drastic increase in the required amount of work.</td>
</tr>
</tbody>
</table>

The RDL contains a lot of information regarding the interpretation of certain words, which would normally not be present in a set of requirements. This can help in avoiding misinterpretations.

Clarity: XML isn’t exactly easy to read, and the RDL contains a lot of semantic information that clouds the actual requirements phrase. Unless you’re versed in RDL, you have no idea what it says.

Verbosity: XML is verbose by nature, but has the added advantage of being easily parsable.

Due to EA-Miner not being available yet, our verdict is limited to the general process and the RDL. Our verdict is as follows: The RDL is an XML-based technique, and therefore parsable by a large set of tools. The information included in the RDL is both a flaw and a merit; it makes it difficult to comprehend for someone not versed in the RDL. On the other hand, one can write tools to overcome this problem. At the same time, the RDL is very clear about the semantics of requirements.

The process in general works rather well, but needs to be better documented. Especially a description of the various phases would be helpful.

### 3.2 Architectural Design

In this section we will describe the Architectural Design method named GAADA, as described by Krechetov, Tekinerdogan and others [KT06]. GAADA stands for General-purpose aspect-oriented architecture design approach.

#### 3.2.1 Learning

The Architectural Design method was probably the easiest method to learn. The method was clearly described and had a very useful case example that helped illustrate the method.

[Figure 2: Aspect Oriented Architectural Design Method]

Figure 6 of [KT06], copied here for reference as figure 2, details the architectural design method. This process is based on a method called ASAAM. There are 3 main processes, namely a conventional process, which includes activities typical to Object Oriented Software Engineering such as Domain analysis, requirements specification and a conventional architectural design process. The result of the conventional process should be an Architecture, as specified by the figure.

The second process is the Identification process, which uses Aspect Oriented Domain Analysis and Requirements Engineering to identify candidate aspects. This is basically the method of [CSR06], which we’ve done in the previous method.

The third process is the Refactoring phase. Here we take the information about candidate aspects and use it as a basis to introduce aspects into our architecture.

[Figure 3: Architectural Design]

The Refactoring phase is meant to be carried out simultaneously with the later steps of the Conventional phase. For each component of the architecture we design, we consider our candidate aspects and how they apply to that component, and change the architecture to incorporate that aspect if needed.

In addition to this process, [KT06] also describes a modelling language (referred to as the Integrated Modelling Language), and an XML-based description language called ADL. These 2 languages are semantically compatible, and can be easily mapped to one another.
3.2.2 Execution

In our requirements phase we found out our primary crosscutting concerns were Authentication and Encryption, both of which can roughly be called Security. We have decided that both of these concepts should be implemented as aspects.

The easiest of these two is Encryption. This aspect can simply intercept any call to a method that sends outgoing messages and receives incoming messages, and encrypt their input or output, respectively.

Authentication is a bit harder though. Our requirements specify a multitude of situations in which a user should be authenticated. These situations translate to a multitude of positions within the design, and they do not share the same name, as is the case with Encryption. Instead, we are now forced to name each specific method affected by this aspect individually.

The architectural design we created can be seen in figure 3. The circles in the diagram indicate crosscutting interfaces, as explained in [KT06]. The arrows indicate all components crosscutted by the aspects. The two boxes at the bottom detail the methods within the components that are affected by the aspects, which can be directly translated as pointcut expressions.

As can be seen, this design is very global. Aside from the fact that an architectural design shouldn't include too much detail, this is simply a case study, so the design need not be exhaustive.

3.2.3 Evaluation

We will briefly discuss the flaws and merits of the Unified Aspect Oriented Architectural Design method.

<table>
<thead>
<tr>
<th>Merits</th>
<th>Flaws</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Curve: The key difficulty in this method is figuring out which parts of the system include aspects. GAADA solves this by using the ASAAM method. ASAAM makes excellent use of work done in the previous phase. Other than that, most of the things one needs to learn to use this method are simple additions to what one already knows.</td>
<td></td>
</tr>
<tr>
<td>Splitting the process into conventional/aspect oriented seems counter-intuitive at first glance</td>
<td></td>
</tr>
<tr>
<td>Clarity: The Integrated Modelling language is similar to UML, and easy to understand. ADL is very concise when compared to RDL, and understandable by anyone familiar with the terminology.</td>
<td></td>
</tr>
</tbody>
</table>

With this information we have reached the following verdict: GAADA is an excellent method for Aspect Oriented Architectural Design. It makes excellent use of information gathered in previous phases, and is very well documented.

3.3 Detail Design

In this section we will describe the detail design method created by Jackson & Clarke, as described in [JC06].

3.3.1 Learning

The first impression when reading [JC06] is the thoroughness of the article. Every facet of the development phase is thoroughly documented. The goal of the document is threefold: to provide a process, language and semantics for Aspect Oriented Design. The latter of these is not fully discussed in the article, though the authors did detail which steps are needed to achieve a unified semantics. So instead we focus on the process and language of the design method.

![Figure 4: Aspect Oriented Design Process](image)

The Aspect Oriented Design process is displayed in figure 4, and is divided into 7 phases. This figure is a copy of figure 1 from [JC06].

**Concern Identification & Classification**

In this step the inputs from previous phases are analyzed, and improved if necessary.

**Design Test(s)**

In this step we perform tests to see if our design adheres to our requirements. As is mentioned in [JC06], there is still some work to be done before this technique is fully supported for AOD.

**Design by Reuse**

In this step we look for existing design patterns that we can use in our design.

**Design Concern Module(s)**

In this step we use UML to design one or more Concern Modules.

**Design Composition Specification**

In this step we describe how our concerns are to be composed to
form a functioning piece of software.

Verify

In this step we will want to verify whether or not our design has the properties we desire, and lacks those we don't want, such as a deadlock.

Refine

In this step we verify certain quality properties of our model, and refine the model if necessary.

For a more detailed explanation of the process, please refer to [JC06]. The language used to model our design is based primarily on Theme/UML, though it incorporates best practice techniques of 23 other techniques, each of which is reviewed in [JC06]. The merged technique can best be called Merge. Simply stated it is Theme/UML with added modelling techniques.

3.3.2 Execution

Rather than creating an exhaustive amount of models for every facet of the system, we will limit our design to the crosscutting concerns. The first step is of course the analysis of our previous designs.

![Diagram of Theme/UML model for Encryption]

The design of the encryption module we presented in the previous phase is sufficient for simply encoding and decoding of data. The design does not specify how this is achieved. While encryption and decryption are similar, they have 1 fundamental difference – we encrypt data before we send it, and we decrypt data after we receive it. GAADA does not allow us to specify this difference (as it is a detail not normally found in architectural design), but Theme/UML does provide a mechanism for specifying this behavior. The result is a Theme with an embedded class, and 2 sequence diagrams explaining the application of the Encryption aspect.

I will give some further explanation of figure 5. A Theme contains all the information about a concern, and any classes part of that concern, in this case Connection. The Theme also contains a sequence diagram detailing how the aspect is applied to this class. Whenever a call to the method send is made, the parameters should be encrypted first. Similarly, whenever a call to receive is made, it's result should be decrypted. What the sequence diagram in figure 5 shows is the augmented method. The original (non-enhanced) method send is now renamed _send(). The call to encrypt is the advice code being executed as part of the aspect functionality. Decrypt functions in a similar way.

Similarly, we can specify the behavior needed for the Authentication aspect, as can be seen in figure 6.

In case of startSession and populate, the methods should only be executed if isAuth returns true, the model in figure 6 displays this behavior.

3.3.3 Evaluation

As with the previous 2 methods, we will list the flaws and merits of this method, and give an initial verdict of the method.

<table>
<thead>
<tr>
<th>Tabel 3: Merits and flaws of [JC06]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Merits</strong></td>
</tr>
<tr>
<td>Verbosity: As is the case with GAADA, Theme/UML does not differ from standard UML that much and has minimal overhead.</td>
</tr>
<tr>
<td>Theme/UML, combined with the various methods discussed in [JC06], has great expressive capabilities.</td>
</tr>
</tbody>
</table>

The primary problem with [JC06] is the thoroughness of the documentation, which is also it's strength. It is almost impossible to learn to use their technique if [JC06] is our only source. On the other hand, it is not an instruction manual, it is a research paper.

Secondly, the instant messenger case is too small in scope to cover all aspects of this method.

The advice I have for this method is to recommend [CB05] as
literature regarding the subject, prior to attempting to learn the integrated method by [JC06].

3.4 General Evaluation
Each of the methods we have used has been evaluated, and has had its flaws and merits discussed. We can now draw an overall conclusion about the three methods.

The three methods have a lot of potential, and some very strong merits. Opposed to this are a number of minor flaws. The methods are of good quality. When using these methods together, the question of interleaving often surfaces. It is not always clear how the methods connect to one another.

Also, the documentation ([CSR06], [KT06] and [JC06]), provide adequate descriptions of their respective methods, but they were not intended to be user guides. Once the authors of the methods feel their method is finished, efforts should be made to create documentation in order to learn the three methods.

4. CONCLUSION
The 3-part process of learning, executing and evaluating a method has proven very useful in putting these methods to the test.

The result of this research is a basic architectural design and a detail design for the crosscutting concerns. This is however not enough to proceed to implementing an instant messenger. The remaining components from the architectural design need to be further described, and several more iterations of the detail design phase are necessary to provide a design of adequate detail for an implementor.

The results of this research are of good quality. Based on our experience with the three methods, we were able to make suggestions for improving the three methods. These improvements are small steps however, since the three methods proved to be of very good quality.

4.1 Future research
This study was relatively small in scale, both due to timing constraints and the case chosen. As such, it does not cover all facets of the three methods. A larger case study should be carried out to better judge the methods. This is especially the case with [JC06], as we mentioned in 3.3.3.

In 3.4 we mentioned that in some cases it is not clear how methods interconnect with one another. With the exception of [KT06], the methods do not give much detail about interconnecting.

The EA-Miner tool is not yet available to the public, so we were unable to involve this in our case study. Therefore, when the tool is finished, there should be a case study that does involve EA-Miner.

ACKNOWLEDGEMENT
I would like to thank Ivan Krechetov for his help in understanding GAADA, and for evaluating this paper.

REFERENCES

[KT06] Ivan Krechetov, Bedir Tekinerdogan, Mónica Pinto Alarcón, Lidia Fuentes, Initial Version of Aspect-Oriented Architecture Design Approach, University of Twente, Twente, AOSD-Europe Deliverable D37, AOSD-Europe-UT-D37, 01 March 2006 http://www.aosd-europe.net/deliverables/d37.pdf


