Dynamic Framed Aspects for Dynamic Software Evolution

Philip Greenwood, Neil Loughran, Lynne Blair, Awais Rashid
Computing Department, Lancaster University, Lancaster LA1 4YR, UK
greenwop|loughran|lb|awais@comp.lancs.ac.uk

Abstract. Software evolution is an inevitable process when developing a system of any notable size and is the most costly stage in the life cycle of a system. Automating parts of this process will reduce the resources required to carry out this stage of development. We aim to develop a framework that achieves this automated evolution by using Dynamic AOP to encapsulate these evolutionary changes and allow them to be applied dynamically at runtime. However, a problem with this is being able to reuse these aspects in different systems and scenarios. We propose the use of framed aspects to parameterise the aspects to generalise them so they can then be customised for a specific use.

1. Introduction

Software evolution is an inevitable stage when developing any type of software. Evolution is defined in [16] as "the process of changing a system to maintain its ability to survive". There are three types of maintenance that can be applied to a system:

- Corrective Maintenance
- Adaptive Maintenance
- Perfective Maintenance

Large amounts of time and money are spent on software evolution and so it is desirable to reduce the amount of effort required to perform this stage of development. In this paper we will outline a framework that will allow a system to evolve certain parts of itself automatically and so reduce the effort to maintain the system. The framework will concentrate on performing perfective maintenance which is generally considered to be maintenance that implements new functional or non-functional requirements.

The use of dynamic Aspect-Orient Programming (AOP [10]) has been proposed to develop an autonomic system in [6]. Autonomic systems [7] are those which are able to perform certain levels of maintenance upon themselves and so can evolve dynamically. The properties that dynamic AOP possesses will allow these changes to be well encapsulated and applied at run-time without needing the system to be taken off-line.

Problems in this domain, regarding the reuse of the aspects and creating the aspects to be applicable in a variety of different scenarios will arise. Framed aspects will allow the parameterisation of each aspect and allow it to be customised to a variety of systems and scenarios.

Framed aspects is a new approach that allows for the easy parameterisation of aspects. The use of framed aspects has been proposed in previous work [11]. The context set out in [11] was to allow the easy development of software product lines. The framed aspects were used to extract common cross-cutting concerns from a particular system family and were then parameterised to allow the easy customisation of the aspect for a particular version of the system. This process improved the reuse of the aspect and reduced development time for later versions. This paper proposes a different application of framed aspects which will require extensions being made to their behaviour.

The aim of this paper is to describe the use of Dynamic Framed Aspects – framed aspects that can be created and applied dynamically to a running system. The paper will illustrate how such a framework could be implemented and will highlight certain key issues.

The structure of the paper is as follows. Section 2 describes in more detail the implementation and use of framed aspects. Section 3 then looks at the use of dynamic AOP to implement a dynamically evolvable system and highlights some of the problems that can arise. Section 4 examines in more detail how dynamic framed aspects can be used to overcome the problems encountered. Section 5 will then describe other related work and how this work will differ from them. Finally, section 6 concludes this report and summarises it.

2. Framed Aspects

Framed aspects are the amalgamation of frame technology [3] with AOP. Frame technology, which has its origins in the late 1970s, provides a mechanism for creating reusable components by way of meta-variables, code templates, conditional compilation, parameterisation, generation and a specification from a developer. Generalising code and assets in this manner allows them to be reused in different contexts making frames ideal for use in the generation of code libraries and software product lines. [18] presents a language independent XML based implementation of frame technology and has been used in the creation of product lines as diverse as city guide systems [19] and UML documents [8]. Typical examples of commands in frames are <*set*> (sets a variable), <*select*> (selects an option), <*adapt*> (refines a module with new functionality) and <*while*> (creates a loop around repeating code).

2.1 Problem with frames

Frames by themselves cannot encapsulate crosscutting concerns effectively, thus future evolutions can lead to changes across frames, effectively limiting the longevity of systems and components and giving rise to architectural erosion [17]. Frame technology requires that variation points and compositions are done explicitly in the code; this can lead to hard to read code. Utilising frame technology with a suitable AO language minimises this disruption by allowing system features, which are often crosscutting, to be encapsulated within a single module. This encapsulation will ease the evolution of the crosscutting concerns, and combining this with the configuration and generalisation

properties that frames provide will ease the evolution process further still.

2.2 Using framed aspects

The Lancaster Frame Processor, a simplified adaptation of frame technology which is strongly influenced by XVCL, has been designed to be used with AO from the ground up and allows features to be composed together in a non invasive manner. [11] describes how a simple cache component can be generalised using frames in order to make it reusable.

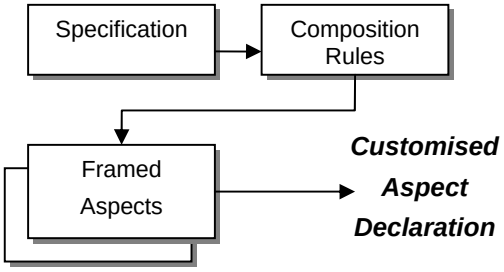

Figure 1. Framed aspect composition

A framed aspect composition (figure 1) consists of three distinct parts:

- Framed Aspect – generalised aspect code (via conditional compilation, parameterisation etc.).
- Composition Rules - contains possible legal aspect feature compositions, combinations, constraints. The rules are also responsible for controlling how these are bound together.
- Specification - **c**ontains the developer's customisation specifications. Frame commands will consist of setting of meta variables and selecting various options. The developer will usually take an incomplete template specification and fill in the options and variables s/he wishes to set.

In the framed aspect approach requirements elicited from the analysis phase are modelled into a feature graph based on FODA [9]. From the feature graph it is possible to delineate frame boundaries by following rules based on mandatory, alternative and optional features of a system or component and from this we can create the aspect code. Variation points or 'hotspots' are identified and the aspectual code is generalised with a suitable frame construct. Constraints and valid/invalid combinations of features are modelled in the composition rules module while the specification module supplies a custom specification from the developer. [12] explains the framed aspect process and methodology in more detail.

3. Dynamic Software Evolution and Dynamic AOP

This section will outline the use of dynamic AOP to implement a system capable of dynamic evolution, describing the properties this type of system needs to posses and why dynamic AOP is suitable for their implementation. First, we will describe how dynamic software evolution can be implemented and the benefits it will bring.

3.1 Dynamic Software Evolution

As already stated, software evolution is an inevitable task that has to be performed when developing a system of any notable size. The aim of the proposed framework is to ease some of the burden that this task requires.

In order to do this we aim to automate some parts of the evolution process by allowing the system itself to decide which available evolution steps are required to keep the system operating as intended. As mentioned previously these evolution steps will be encapsulated using dynamic AOP, with framing techniques used to parameterise them.

Based on information collected at run-time (collected by monitoring modules regarding the current environmental conditions) the system will be able to make decisions about which evolution changes, if any, are required. The information gathered will also be used to customise the aspect to suit the current system and conditions.

Automating this process will provide many benefits to system operators, from improved system availability to lowered running costs. Since such systems can maintain themselves dynamically, they do not have to be taken off-line for re-configurations to be applied. Also, as human contact is reduced, system availability is improved as human error is the most common cause of system failure [4]. Furthermore, as fewer maintenance staff are required, the running costs are lower.

Obviously there are some limitations as to what the proposed framework will be able to achieve. For example, the evolutionary changes must be already be thought of and pre-programmed but the key point is that it will be the system that decides when and where the changes should be applied. If the correct selection of evolutionary changes exists in the framework they should be applicable to a number of different systems and scenarios. This will reduce development time and improve reuse.

Large scale changes, such as when entire business goals change will have to be managed and applied manually as this is beyond the scope of our framework. We will be focussing on small scale perfective maintenance such as switching between algorithms to maintain optimum performance and introducing various functional and non-functional concerns as they are required.

3.2 Dynamic AOP

AOP aims to improve the areas where Object-Oriented Programming (OOP) fails by allowing concerns that would normally crosscut a number of objects to be cleanly encapsulated in a single element. AOP introduces three concepts: aspects, advice and joinpoints. Advice is used to implement the crosscutting concern, joinpoints specify the points in the base-code where the advice should be applied and aspects are used to encapsulate the advice and joinpoints.

Dynamic AOP techniques allow aspects to be woven while the system is being executed (either at class-load time or run-time). This provides a variety of useful features such as being able to introduce entirely new aspects and removing aspects already

woven. However, certain problems can also arise such as lower performance, compatibility issues and security issues.

The majority of concerns that will require evolution will tend to be crosscutting and so dynamic AOP will be suitable for this kind of implementation. The following properties of dynamic AOP have been identified as being fundamental for implementing a system able to evolve automatically and dynamically, detail of these can be found in [6]:

- Apply adaptations dynamically
- Easily remove adaptations
- Encapsulate adaptations
- Implement fine-grained changes
- Apply adaptation to various points in the system

There are currently numerous dynamic AOP techniques such as AspectWerkz [2], JAC [13] and Prose [15]. The majority of these techniques support all of the above properties but to varying degrees. Whichever technique we choose, we will inevitably have to extend it in order to overcome a number of issues; see [6] for a comprehensive list of these problems. The issues that this paper aims to address are:

- Customising aspect behaviour – developing a mechanism for customising an aspect to suit the current needs of the system. The run-time and environment conditions of the system will vary greatly over time; the aspect should be adaptable to meet these needs.
- Re-use – as well as being customisable to suit the current conditions, the aspects should be applicable to a variety of systems. Each system may have methods which perform similar actions but the differing method and fields names may prevent an aspect which is suitable to alter the actions of these methods from being applied. The use of framed aspects will overcome these problems.

4. Dynamic Framed Aspects

From the earlier description of framed aspects it is clear that they allow the customisation of the behaviour of the aspects and also improve the reuse of the aspects they implement.

However, the framed aspect specification is currently statically defined and the concrete aspects are created from the frames before the aspects are woven to the system, normally during the compilation phase. This process prevents changes being made to the framed aspects dynamically.

In order to implement dynamic framed aspects we propose the use of a dynamic AOP language to create the aspects and a mechanism to dynamically create/update the aspects depending on the current environmental conditions.

For our initial prototype we propose to use AspectWerkz as the dynamic AOP technique. The reason why AspectWerkz is chosen here is because it is the most familiar to us at this time. Prose was discounted due its limitations in the types of aspects it can create and its overall flexibility.

Parameterising aspects in AspectWerkz should not pose any problems as the structure they use is the same as standard Java classes, and Java classes have been successfully parameterised in the past using framed aspects.

The architecture we envisage will be implemented is shown in figure 2.

AspectWerkz Runtime
The AspectWerkz runtime is responsible for weaving and unweaving the aspects with the application base-code. The AspectWerkz runtime must be able to receive aspects sent to it from the framed aspect server.

Dynamic Application
The dynamic application is that which is being controlled by the AspectWerkz runtime and the application monitor. Using the proposed architecture any application should be able to be made dynamic without requiring much modification to the source code.

Application Monitor
This module is responsible for monitoring the current context of the running application and then translating this into an aspect request that will be sent to the framed aspect server. Initially only physical attributes of the system will be monitored such as memory usage, network usage, hard-disk space and processor usage. More detailed context information can be gathered later, such as parameter values passed to methods, field values, etc.

Framed Aspect Server
The framed aspect server will receive aspect requests from the application monitor. From these requests the server will retrieve the framed aspect from a repository and then customise it to suit the request received. The concrete aspect will then be sent to the AspectWerkz server to be woven to the dynamic application.

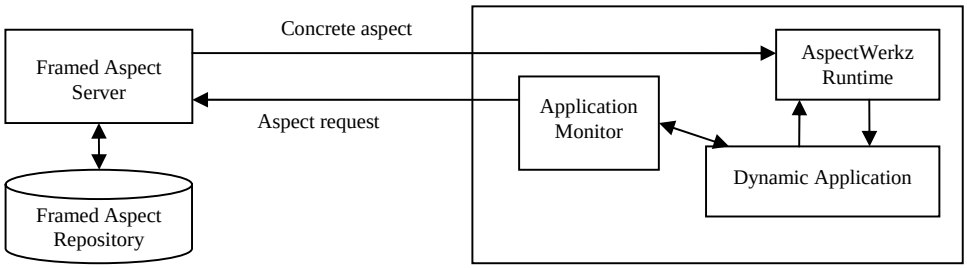

Figure 2. The proposed architecture.

Framed Aspect Repository
This will be used to store all the framed aspects. It is proposed that this will be implemented as a database structure to support querying and fast retrieval. It should also be possible to add new framed aspects to introduce new behaviour to the dynamic applications that was not available when the system was first started.

Aspect Request
The aspect request sent to the framed aspect server will contain a set of parameters detailing the current conditions of the system and the running environment. This data will be used to select an appropriate framed aspect and to create a concrete version of it.

4.1 Problems
This section will briefly introduce some of the initial problems that we anticipate when developing such a framework.

Applying Framed Aspects to the Base-System
One of the initial problems that we need to overcome is finding a way to mark where each aspect can be applied in the system. Each framed aspect implements a different crosscutting concern and these concerns may not be applicable at all points in the system. For example, a caching aspect does not need to be applied to a method that only prints a message to the user.

The solution we propose for this problem is to allow the programmer to specify a configuration file which will allow them to specify all the potential aspects that are relevant to the system and all the places where they could be applied.

However, this could result in complications and a long configuration file being specified. To reduce this problem, aspects could be grouped according to the types of changes they apply. This will allow the programmer to list types of aspect rather than each individual one. Also a GUI would aid the user in the creation of this file.

Framed Aspects Structure
For this solution to work successfully, each framed aspect will have to be created to follow certain conventions so that they can be applied to a variety of different systems and to make their structure predictable. Currently, framed aspects are largely unstructured and can be used to parameterise any part of the aspect. A more constrained structure is required so that the dynamic application can be created to accommodate all the potential aspects that may be applied to it. If the system has already been created then it may need refactoring to be made suitable for the aspects to be used.

Monitoring
The way the framed aspects have been parameterised will affect the information that is to be collected. We have to be sure that the information collected from the application monitor will be able to 'fill-in the gaps' of the framed aspects. From the information passed to the framed aspect repository a decision will have be made about which is the best framed aspect to use. It is not vital for this decision to be correct as the monitor will be able to use the data it has collected to check the aspect has had the desired effect; a new aspect can be requested if it has not. The above-mentioned configuration file, used to specify where the aspects need to be applied, will also be used to determine where and what needs monitoring.

4.2 Example
Suppose a client-server application is having performance problems. The monitoring module on the client detects a particular method that is experiencing network congestion when communicating with the server. From the configuration file, the system knows that introducing a cache may solve the performance problems. The monitor sends a request for a caching aspect, this specification includes what needs to be cached and the size the cache should be (this depends on the amount of resources available). This information is sent so that an aspect can be created with the correct data types and so that it does not use too many resources caching data. The aspect server will then create and compile the aspect and send it back to the client where it will be woven with the base-system.

This example, although simple, demonstrates how such a system would operate and how it could evolve its behaviour depending on the current system properties. Obviously in reality the aspects would have more than two parameters that need to be collected and sent to the server, but the principles are the same as in this simple example.

5. Related Work
Hot Swap [1] is an autonomic system developed by IBM. A monitoring component is used to examine the environment of the Hot Swap system and then requests adaptations to be made. The limitation of Hot Swap is only whole components can be replaced; AOP will allow us to make much more fine-grained changes. Additionally the replacement components used in Hot Swap need to be entirely pre-programmed, whereas the aspects in our solution can be dynamically re-programmed to suit the current environmental needs of the system.

An alternative way system behaviour could be modified is to use a technique called adaptive code. Adaptive code can be implemented in a variety of ways, from using parameters that are modified at run-time to alter the behaviour, to implementing a number of alternative algorithms to perform a certain task and the most appropriate can be selected at run-time (see [5]). The biggest drawback of using adaptive code is that it is not possible to insert new code or new monitors at run-time.

In the literature, the implementation of a system which possessed a certain degree of dynamic behaviour has also been achieved through the use of Prose [15]. MIDAS [14] was created to allow the distribution of aspects in a mobile environment. Whenever a node entered a particular location, aspects were distributed to it so it would behave correctly for that particular location. This is a limited solution in that the system is only location aware; we aim to create a framework that can be used in a variety of scenarios.

6. Conclusions
This paper has proposed a framework which will allow a system to be developed that will be able to evolve using a combination of framed aspects and dynamic AOP. When combined, the properties these technologies posses will allow aspects to be developed with a high degree of flexibility and they will be able to be applied

dynamically to the system. In previous work the problem of reuse and customising the aspects to suit a particular scenario was highlighted as an issue to be resolved; the use of framed aspects will achieve this. This work is still in the early stages of development but the benefits of using framed aspects in this way are clear and will ease the development and implementation of dynamically evolvable systems.

References

[1] Appavoo, K. et al, "Enabling Autonomic Behaviour in Systems Software with Hot Swapping", IBM Systems Journal Vol. 42 No. 1, 2003.

[2] AspectWerkz, "AspectWerkz Dynamic AOP for Java Overview", http://aspectwerkz.codehaus.org/, 2004.

[3] Basset, P. "Framing Software Reuse – Lessons from Real World", Yourdon Press Prentice Hall, 1997.

[4] Ganek, A.G., T. A. Corbi, "The Dawning of the Autonomic Computing Era", IBM Systems Journal Vol. 42 No. 1, 2003.

[5] Glass, G., Cao, P., "Adaptive Page Replacement Based on Memory Reference Behaviour", Measurement and Modelling of Computer Systems pp. 115-126, 1997.

[6] Greenwood, P., Blair, L., "Using Dynamic AOP to Implement an Autonomic System", Dynamic Aspects Workshop AOSD 04, 2004.

[7] Horn, P., "Autonomic Computing: IBM's Perspective on the State of Information Technology", 2003.

[8] Jarzabek, S. and Zhang, H. "XML-based Method and Tool for Handling Variant Requirements in Domain Models", 5th IEEE International Symposium on Requirements Engineering pp.166-173, 2001.

[9] Kang, K. C. et al, "Feature Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21 Software Engineering Institute Carnegie Mellon University, 1990.

[10] Kiczales, G., et al, "Aspect Oriented Programming", Proceedings ECOOP '97, 1997.

[11] Loughran, N. et al, "Supporting Product Lines Evolution with Framed Aspects", ACP4IS Workshop AOSD 04, 2004.

[12] Loughran, N., Rashid, A., "Framed Aspects: Supporting Configurability and Variability for AOP", ICSR-8, 2004.

[13] Pawlak, R. et al, "JAC: A Flexible Solution for Aspect-Oriented Programming in Java", Reflection 2001, 2001.

[14] Popovici, A., Frei, A., Alonso, G., "A Dynamic Middleware Platform for Mobile Computing", Middleware 2003, 2003.

[15] Popovici, A., Gross, T., Alonso, G., "Dynamic Weaving for Aspect-Oriented Programming", AOSD 2002, 2002.

[16] Sommerville, I., "Software Engineering 5th Edition", Addison Wesley, 1997.

[17] Van Gurp, J., Bosch, J., "Design Erosion: Problems and Causes", Journal of Systems and Software Vol. 61 Issue 2, 2002.

[18] Wong, T.W. et al, "XML Implementation of Frame Processor", Symposium on Software Reusability SSR 01 pp. 164-172, 2001.

[19] Zhang, W., et al, "Reengineering a PC-based System into the Mobile Device Product Line", International Workshop on Principles of Software Evolution (IWPSE),2003.