Supporting Evolution in Software using Frame Technology and Aspect Orientation

Neil Loughran, Awais Rashid
Computing Department, Lancaster University, Lancaster LA1 4YR, UK
{loughran | awais} @comp.lancs.ac.uk

Abstract

This paper discusses how the problems involved in supporting evolution in software can be resolved by using aspect oriented programming and frame technology. Throughout the lifetime of a software system, new requirements may arise that will require the existing system to be altered or evolved in someway. Evolution is something which is almost impossible to predict at the design stage. Although it is common to anticipate future evolutions and therefore prepare and design our code to accommodate this, there will eventually come a time when a certain feature or scenario appears where this may not be practical.

1. Introduction

Throughout the lifetime of a software system or architecture, new requirements may arise that will require the existing system to be altered or evolved in someway. Therefore an effective mechanism for evolution is an important factor in the creation of software systems. It is estimated that up to 80% of lifetime expenditure on a system will be spent on maintenance and evolution. However, achieving effective evolution across the board with current technologies is difficult because of the complexities involved.

Evolution is something which is almost impossible to predict at the design stage. Although it is common to anticipate future evolutions and therefore prepare and design our code to accommodate this, there will eventually come a time when a certain feature is required or a scenario appears where this may not be practical.

2. Background

2.1 Categories of evolution

Software evolution and maintenance can be divided into the categories shown in Table 1, which are derived from [6].

Table 1. Traditional categorisation of evolution

Category	Description / Example
Corrective	Fixing of bugs
Adaptive	Addition of new features Changing of functionality Support for new platforms
Perfective	Improving system functionality Improving performance
Preventative	Preventing problems before they occur

It should be noted here that any evolution made to a system could fall into one *or more* of the categories shown. For instance perfective evolution where, for example, the performance of a particular component needs to be improved, may also require other components of the system to be evolved thus requiring adaptive and possibly preventative evolution. Evolution of a particular component or feature may require other assets at different stages of the software lifecycle to also be evolved such as testing and documentation. This brings forward cases where evolution effectively crosscuts system structure *and* architecture. From this we can add two sub categories to the aforementioned, namely crosscutting and non-crosscutting evolution.

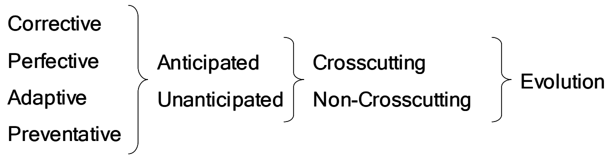

Figure 1. Evolution types

Another important notion is that of anticipated and unanticipated evolution. While anticipated evolutions can be obviously accommodated, unanticipated evolutions are of great concern if the system or architecture is to avoid erosion. Figure 1 illustrates the possible evolutions types.

Aspect orientation is designed to be used with conventional separation of concerns mechanisms, such as object-orientation, and should not be seen as a replacement for these techniques. It should be noted that the notion of aspect orientation now goes far beyond just programming level and is now being used at different levels of the software lifecycle such as the software design [7] [8] and requirements stages [9][10].

2.2 Crosscutting and separation of concerns

One of the principle requirements in software composition is to achieve a good level of separation between the different concerns in the system. By separation of concerns we mean the encapsulation of particular functional or non functional properties of the system which crosscut the system structure. This allows each concern to be viewed in it own space making system comprehensibility and manageability easier to understand thus facilitating reuse and evolution.

2.3 Software erosion

Erosion occurs when software, which has been continually evolved, eventually becomes difficult to understand, maintain and therefore evolve and reuse. When evolving a system we want to lessen the negative effects of the evolution in order to minimise the possibility for erosion. Erosion can occur anywhere from erosion of a particular component to the much larger problem of erosion in software designs and architectures. [1] cites cases where projects have had to be started from scratch as the source had become eroded beyond repair.

3. Approaches

3.1 Frame Based Technologies

Frame technology [2] is a concept that has its roots in the 1970s and was conceived by Paul G. Bassett as a means to providing *adaptive reuse*. By adaptive reuse we mean the process of creating generalised components that can be easily adapted or modified to different reuse contexts. From a simple perspective frame technology is a language independent textual pre-processor that creates software modules by using code templates and a specification from the developer. Variations of the technology inspired by Bassetts work such as XVCL [3] and FPL [4] use the XML language in order to implement the framing syntax. Frame technology works by organising frames into a hierarchy as shown in Figure 2, which depicts a partial view of a simple web browser.

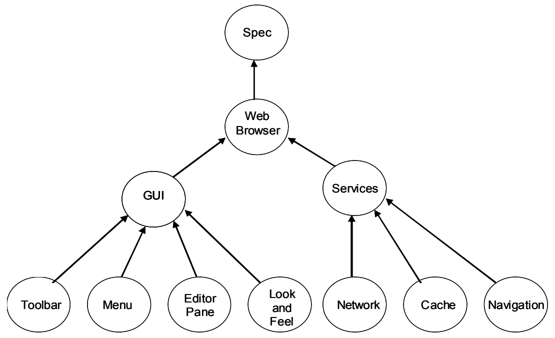

Figure 2. Example of frame technology hierarchy

Frames allow points of interest in the code, such as variation points, code repetition, configuration routines, optionality etc…, to be explicitly marked in place with metadata tags or moved to a *child* frame. By allowing these points of interest to be marked or modularised the developer can quickly create highly customisable systems. The basic granularity for a frame is the separation of a particular concern, class, method or related attributes with the hierarchy of frames serving to isolate content into separate layers, allowing the localisation of the effects of change and easing evolution. Usually the lower order frames are the most reusable as they contain less context sensitive information such as IO routines, library functions etc...

3.2 Aspect Oriented Programming

Aspect oriented programming (AOP) [5] technologies are now gaining popularity as a means for supporting the separation of concerns for features and constructs that would otherwise cause unmanageable code tangled across multiple classes in traditional object-oriented systems (Figure 3).

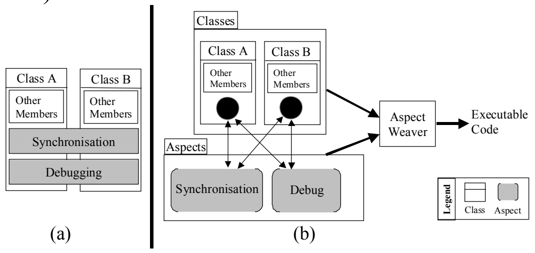

Figure 3. (a) Crosscutting concerns in OO languages (b) Separation of crosscutting concerns with AOP

Examples of the type of concerns that can cause this fragmentation of context are logging, profiling and tracing. Having all of the code for each particular concern modularised has the benefit of making system code easier to evolve, maintain and be reused hence increasing productivity, flexibility and reducing costs thus making them conducive for use within the software product line context.

There are numerous aspect oriented programming approaches available for use with the most well known being *AspectJ* [11], *Hyper/J* [12], and *composition filters* [13]. There also AOP approaches to *run time* evolution of programs such as *Java Aspect Components* [16] (JAC) and *JMangler* [17]. Run time evolution promises the facility for programs to be modified while they are executing. This facility will be of great importance to systems where stopping the system and evolving the code thus rendering the system from functioning is an undesirable characteristic from economic and safety perspectives. Examples of these systems could be 24/7 banking facilities, online commerce and air traffic control systems.

3.3 Other approaches

There are other approaches which seek to solve the problems associated with software product line issues notably Gen Voca [14] and work from the SEI [15]. However for the purpose of this paper we will only concentrate on frame based and aspect oriented approaches.

4. Supporting evolution

4.1 Evolution with frames

In section 2 we mentioned the notion of crosscutting and non crosscutting evolution. Non crosscutting evolutions are generally easy to solve with frame technology as their implementations are localised, the main problem being where the evolution might be spread out over many child frames spawned from the parent frame. In this sense the framing process can suffer from fragmentation of context.

Crosscutting evolution however, is not very effective with framing alone as there is no separation of concern mechanism beyond class and frame boundaries. For this reason aspect oriented technologies can play an important role in improving the evolution of systems which impart crosscutting behaviour.

4.2 Evolution with aspect orientation

Aspect orientation has been created with separation of crosscutting concerns in mind and thus would seem to be an ideal candidate for supporting the crosscutting evolutions that is difficult to achieve by framing.

However, while it is possible to use aspect oriented technologies alone to perform some form of evolution, it is constrained by the lack of configurability, generalisation and optionality that framing allows.

4.3 Hybrid approach

We have previously made a case where neither framing nor aspect orientation can support various evolutionary scenarios effectively in isolation. With this in mind it makes sense to combine the two technologies to improve on current techniques. Table 2 shows a comparison of the two techniques with their associated merits and demerits.

Table 2. Comparing frames and aspect orientation

Capability	Framing	AOP
Configuration Mechanism	Very comprehensive configuration possible	Not supported natively, dependent on IDE
Separation of Concern	Only non crosscutting concerns supported	Addresses problems of crosscutting concerns
Templates	Allows code to be generalised to aid reuse in different contexts	Not supported
Code Generation	Allows autogeneration of code and refactoring.	Not supported
Language Independence	Supports any textual document and therefore any language	Constrained to implementation language although this will change as AOP gains wider acceptance
Use on Legacy Systems	Not supported	Supports evolution of legacy systems at source and byte code level
Dynamic Runtime Evolution	Not supported	Possible in JAC and JMangler. Future versions of AspectJ will have support.

By combining the two approaches we gain increased flexibility which will allow aspects to handle the crosscutting concerns and framing to impart configuration, optionality and generalisation of those aspects where required. Figure 4 demonstrates how a generalised aspect can be used to perform a crosscutting evolution on a system or architecture

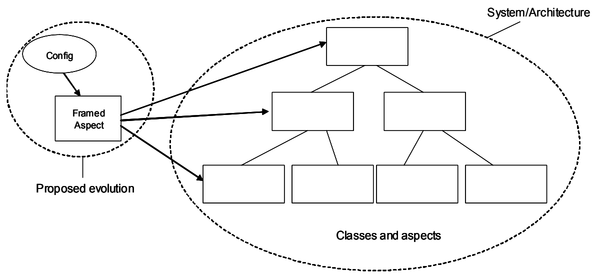

Figure 4. Evolution with framed aspects

It should be noted here that the framed aspect could work on the architecture even if the architecture itself was framed or not, thus allowing frames in some sense to work on legacy systems. Using these approaches brings forward exciting possibilities for the following:-

- Generalised reusable components which solve crosscutting problems.
- Refactorisation of aspectual code
- Configurable dynamic run time aspects
- Configurable legacy aspects
- Configurable development aspects (tracing, profiling etc)

These could be used to perform various kinds of tasks and evolutions that previously would have been difficult to realise in a particular technology alone.

5. Conclusion

We have seen that neither frame technology nor aspect oriented technologies alone can solve all the problems that evolution brings. There is clearly a need for configurable aspects for crosscutting evolution. By combining aspect orientation with a variant configuration mechanism such as frame technology we get the best of what both have to offer in terms of flexibility and evolvability. Generalisation of aspects allows them to be used in different situations thus making them ideal candidates for use within software product lines. By utilising aspect orientation and allowing crosscutting concerns to be localised we improve our understanding of system comprehensibility and thus lessen the risks of architectural erosion.

6. Acknowledgements

The authors would like to thank Dr Stan Jarzabek and Dr Zhang Weishan of the National University of Singapore with regards to queries on framing technologies.

References

[1] van Gurp J. and Bosch J., "Design Erosion: Problems & Causes", *Journal of Systems & Software*, volume 61, issue 2, 2002.

[2] Bassett, P. 1997. Framing software reuse - lessons from real world, Yourdon Press, Prentice Hall.

[3] Wong, T.W., Jarzabek, S., Soe, M.S., Shen, R. and Zhang, H.Y. "XML Implementation of Frame Processor," *Symposium on Software Reusability, SSR'01,* Toronto, Canada, May 2001, pp. 164-172.

[4] Sauer, F. "Metadata driven multi-artifact code generation using Frame Oriented Programming", OOPSLA 2002.

[5] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., Irwin, J., "Aspect Oriented Programming," *Proc. of the European Conference on Object-Oriented Programming (ECOOP),* 1997.

[6] Lientz, B., Swanson, E., and Tompkins, G., "Characteristics of Application Software Maintenance," CACM 21, No. 6 June 1978

[7] Clarke, S., Walker, R. J., "Composition Patterns: An Approach to Designing Reusable Aspects" *proceedings of the 23rd International Conference on Software Engineering* (ICSE), Toronto, Canada, May 2001.

[8] P. Tarr, H. Ossher, W. Harrison and S.M. Sutton, Jr. "N Degrees of Separation:Multi-Dimensional Separation of Concerns". *Proceedings of the International Conference on Software Engineering* (ICSE'99), May, 1999.

[9] Rashid, A., Sawyer, P. et al., "Early Aspects: A Model for Aspect-Oriented Requirements Engineering", *IEEE Joint International Requirements Engineering Conference*, IEEE Computer Society Press, 2002.

[10] Grundy, J., "Aspect-Oriented Requirements Engineering for Component-based Software Systems". *4th IEEE International Sympsium on RE*, IEEE Computer Society Press, 1999.

[11] Xerox PARC, USA, AspectJ Home Page, http://aspectj.org/

[12] IBM Research, Hyperspaces, http://www.research.ibm.com/hyperspace/

[13] Aksit, M., Bergmans, L. & Vural, S., "An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach", *ECOOP '92*, LNCS 615, pp 372-395, Springer-Verlag, 1992.

[14] Batory, D., Chen, G., Robertson, E. and Wang, T. "Design Wizards and Visual Programming Environments for GenVoca Generators," *IEEE Trans. on Software Engineering*, Vol. 26, No.5, May 2000, pp. 441-452

[15] Carnegie Mellon, Software Engineering Institute, homepage http://www.sei.cmu.edu

[16] Pawlak, R., Martelli, L. and Seinturier, L. The JAC project home page. http://jac.aopsys.com

[17] Kniesel, G., Costanza, P. and Austermann, M. JMangler home page, http://javalab.cs.uni-bonn.de/research/jmangler/