Available online at www.sciencedirect.com

ScienceDirect

AASRI Procedia 5 (2013) 133 – 139

AASRI Procedia

www.elsevier.com/locate/procedia

2013 AASRI Conference on Parallel and Distributed Computing and Systems

# Design Reusability and Adaptability for Concurrent Software

Paniti Netinant[a,b] *

[a]*Information Technology Department, Bangkok University, Bangkok, Thailand*
[b]*Concurrent Programming Research Group, Illinois Institute of Technology, Chicago, IL, USA*

**Abstract**

Developing reusable and adaptable concurrent software are very difficult. Reusability and adaptability in concurrent software become minimal. These collaborations not only perimeter reusability, but also make modification invasive changes to meet new requirements in the design of the concurrent software. The concurrent system has to reengineer, in order to meet the future requirements. A misunderstanding, that a concurrent object-oriented programming generally endorses reusability and adaptability, as none of these problems is imposed. Software developers have to specifically deliberate software making reusability and adaptability. We present a concurrent aspect framework which better supporting reusability and adaptability. In the framework, functional components and system assets are relatively separated. This technique makes concurrent software developing better reusability and adaptability. A framework enables manageable reusability and adaptability for building of concurrent software. Our research focuses on decomposition of contact in concurrent software development and our goal is to demonstrate a better model of concurrent software design.



*Keywords:* Aspect Orientation; Adaptability; Reusability; Framework; Concurrent Software.

* Corresponding author. Tel.: +66-8185-1116-; fax: +66-2965-0366.
*E-mail address:* paniti.n@bu.ac.th.


doi:10.1016/j.aasri.2013.10.069

## 1. Introduction

One of the key appearances in concurrent software design is able to deal with a reckless adaptability and reusability. Software continuously changes when either a new hardware or requirement is needed. Adaptability of concurrent software is definitely demanded. Without reengineering of the whole concurrent software, reusability in the design of concurrent software is a critical aspect of concurrent software development. The more components in the existing software can reuse, the faster and better software design are. Most changes in software make it defects and producing errors [1]. When we need new features or adding new hardware to the system, modification of the software is inevitable. Modifications of the concurrent software are not a relaxed process. Variation of the concurrent software is invasive evolutions and changes to meet needs. Certain alterations can be applied to traditional functional decomposition and object-oriented analysis and design. However, modifications must not be too complicate. Concurrent software developers essentially keep simplicity of concurrent software development. Ease in the concurrent software development is a noteworthy specific of a good software development. Classical benchmarks in concurrent software design have used to implement and demonstrate how practical of adaptability and reusability of the concurrent software development, like readers-writers and producers-consumers problems.

The commercialization of concurrent software, e.g. operating systems, has ensued in their design gaining more importance. System software such as operating systems is constantly extended for improvements as well as to support new features and hardware. During the design of concurrent system software, certain decisions such as reusability and adaptability are essential [2].

The necessity in the principle of separation of concerns deals with one issue at a time, originally addressed by [3, 4]. The principle of separation of concerns is the major of concurrent software development. Separation of concerns leads many benefits. Although reusability and adaptability recognized, nowadays there is none commonly acknowledged approach tips programmers and developers how preeminent achieve reusability and adaptability. Developers and programmers have to deliberate how separation of concerns can address and how a quantity of aspects in the system can take [5] not only in the design, but also in an implementation of the system software. The better separation of concerns is, the better reusability and adaptability of both functional components and aspects in the system is.

Concurrent system software consists of multiple concerns spreading across many components of the system. Systems are disreputable of one concern spreading across many components. We mean this situation is not only depriving reusability, but also gradual adaptability. System asset is defined as one concern spreading across multiple components in the system. Example system assets include system monitoring, error handling, synchronization, and scheduling.

At present, traditional functional decomposition techniques have been used in most current concurrent software design. Separation of concerns can achieve by using functional decomposition technique with excessive limitation of reusability and adaptability. The concurrent object-oriented design and programming provide higher abstraction. However, reusability and adaptability have to manually achieve. It depends on the ability of developers and programmers. Concurrent object-oriented design and programming provide only one facet which developers can separate concerns. However, tangling concerns still exist in the system if developers are unaware of it. Concurrent object-oriented design and programming solutions help to localize the effect of adding or changing to data, but it can also result in tangling and scattering of system assets across multiple objects [6]. Tangling system assets lead to a lack of higher level of abstraction in concurrent software increasing dependencies between the functional components, known as tightly couple. Tightly couple makes it difficult to reuse and adapt. As a result, adapting or reusing of existing components has to produce massive changes in the concurrent system. This affects software quality of adaptability and reusability for concurrent system. Thus, reengineering the whole system to meet the new requirements cannot avoid.

Recently, an aspect-oriented programming (AOP) proposes a better separation of concern technique. It keeps advantages of object-oriented programming. Its goal is to achieve central decomposition [7]. Nowadays, there are many software development methodologies. There is one used by developers in order to produce high-quality software such as aspect-oriented software development [10, 11, and 12]. The aspect-oriented approach aims to provide explicit defining for modularizing design decomposing concerns of functional components in the system. Spreading concerns divide, contain, and collaborate in separated components called aspects. Aspect components and functional components are woven into the whole system at predefined joinpoint. Aspect-oriented approach is a general software model which is noticed to extract concerns spreading in components. Aspect-oriented approach can apply to all phases in the software development life cycle. In reality, aspect-oriented approach encourages decomposition of spreading concerns in separated components at the early beginning of software development. Current aspect-oriented approaches interpret entities into components and aspects. Finally, the system will automatically combine aspects, where the joinpoints have been defined, and functional components to create the system. A system asset is a concern which spread multiple functional components. Every system asset can be splatted into independently from functional components.

## 2. Architecture of Framework

Using a fork for plotting plants is a misplaced as using a spoon to dig out an excavation for a dinosaur bone. Concurrent software development turns out to be more and more complex. High quality concurrent software development is become more expensive. Concurrent software is never completed finish. In the future, new requirements and changing hardware require to be modified and improved the software. For these reasons, it is practical to look for adequate methods that allow complexity to be mastered, or make invasive modifications by better supporting reusability and adaptability.

We believe that aspect-oriented approach to software design can assist developers and programmer in the concurrent system. It could cleanly support separating concerns from multiple components, and captured into only one component, called an aspect component. Aspect-oriented approach could deliver a technique making it possible to a higher abstraction. Then the weaver component composes functional components and system assets together [8]. The framework view differs from the languages view, in the sense the framework architecture are usually more flexible than the language part. The framework is a design of a cluster of cooperative objects which handles aspects. The framework consists of three parts. First, the moderator takes the place of the weaver. The collaboration of functional components and the aspect components has been defined by jopinpoint in moderator components. Second, the proxy intercepts accesses to functional components. The proxy can have more than one in order to increasing the speed of intercepting accesses. A proxy object uses a design pattern such as factory or proxy pattern. Third, an aspect-factory component creates asset components. The framework is illustrated in Fig.1.

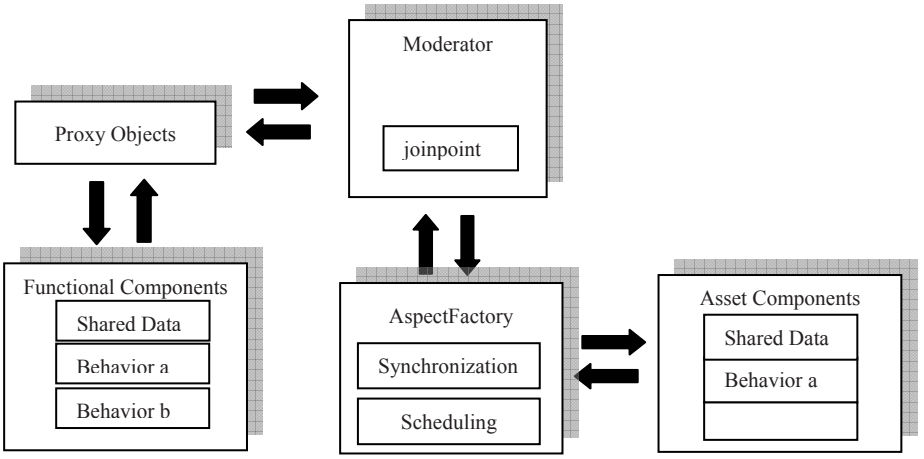


Fig 1. Concurrent Aspect Framework.

Composing of aspects using the moderator presents components of aspect factory. Functional components need to gather required aspectual components. Then functional components decide about the weaving policies. The framework offers an open architecture that is very suitable for adaptability and reusability of both functional components and aspect components knowing of system assets. The architecture is conceptually very simple as it divides aspect and functional components, and the control over the weaving aspects. The aspects are centralised in the aspect factory component.

System assets in concurrent software should not be seen and be treated as a single monolithic component. The framework consists of components, assets, and layers. Functional components are the basic functionality of the concurrent system software. System assets are concerns extracted from functional components. Each asset component is a concern like fault tolerance, synchronization, scheduling, naming, and logging. Layers are a number of more manageable sub problems decomposed into functional components and system assets. Each functionality component will be isolated from system assets. System assets of each functional component will be captured into asset components. Asset components are defined asset abstraction and asset objects. This design outcomes a better clarification of interactions, reduces a concern tangling by increasing understanding of system assets in each component. It makes the concurrent advanced level of abstraction, and better to easily understand.

Design patterns played roles in the concurrent aspect framework. We use abstract a factory pattern and a bridge pattern to create aspect components. A factory pattern uses to isolate between an asset abstraction and the asset objects. The obligation and the process of creating asset objects have been responded by a factory pattern. The asset object appears only once in a functional component instantiated. When an asset object has been modified, the framework promotes consistency. A bridge pattern helps to prevent a permanent binding between an asset abstraction and an asset implementation. Using a bridge pattern, asset abstractions and asset implementations can freely association and extend. Changing in the implementation of an asset object or an asset abstraction does not result any effects on functional components. A proxy pattern uses to control access asset components and allow organization responsibilities when an asset component is retrieved.

## 3. Adaptability of the Framework

Most developers think an object-oriented model naturally endorses reusability and adaptability. We argue that those concerns are not naturally enforced. Concurrent system software must be carefully specified and

designed in order to make the concurrent software reusable and adaptable. One of many important software qualities is an adaptability factor in software design. A minimal degree of adaptability can lead to reengineering a whole system. Adaptability of software quality has explicitly plotted into a concurrent software design stressed in concurrent software research [2]. Changing requirement or adding new hardware without invasive modifications is an increasing degree of adaptation. Traditional object-oriented software design supports minimal degree of adaptation. Adaptability can achieve in two ways: - a vertical adaptability or a horizontal adaptability. An inheritance component supports a vertical adaptability. A composition, encapsulation, and message passing support a horizontal adaptability. Concurrent object-oriented software design does insufficiently support adaptability of concurrent software. System assets are not only tangled in functional components, but also prevented concurrent software to adapt or change later. The concurrent aspect framework does not require any structure to represent and support system assets. A few classes for system assets must be added into the concurrent software while concurrent software is designing and implementing. Using the concurrent aspect framework, previous system assets can be easily removed from the system. The moderator objects response to dynamically activate or deactivate system assets during runtime (dynamic adaptability can achieve).

The semantic of collaboration between system assets and functional components can cleanly define in the moderator through joinpoints. Joinpoints in the moderator represent a semantic of interaction between system assets and functional components as well as the ordering of execution system assets. Joinpoints can dynamically alter during the runtime. An automatic weaver technologies does not support a dynamically adaptability. The concurrent aspect framework designs functional components and aspect components independently. When developing a system asset, developers and designers do not have to concern functional components. On the other hands, development of asset components does not know functional components in advance (before runtime). The asset components only know functional components. An asset component has been created and registered by the moderator. Joinpoints of the moderator take care of weaving functional and asset components. Adding or changing new asset or functional components needs not to reengineer the system. A new collaboration between asset and function components has to define joinpoints in the moderator. Only the moderator has to be recompiled. There is no need to make any modification either functional or asset components. This is an invasive change. The concurrent aspect framework seems to promise a better degree of adaptation in the design of concurrent software with naturally adaptable.

## 4. Reusability of the Framework

The concurrent aspect framework does not only support adaptability, but also reusability. The functional components and asset components are independently thought and designed. The framework can achieve both a vertical and horizontal reusability of both functional components and asset components. The moderator uses design patterns that weave functional components and system assets together. Adaptability can achieve by using design pattern [2]. The pros of using the moderator are when a new system asset added to the system. There is no need to make any modification of the moderator. Reusability can achieve in two ways: - a vertical adaptability or a horizontal reusability.

First, a vertical reusability can achieved. Asset components on the higher layer can use abstractions of asset components on the lower layers to implement. To develop rapid concurrent software, this promotes not only reusability, but also cost and time reduction. Asset object in the upper layer can specify and reuse asset object in the lower layer that provide a generally specified asset. Channing of specification of asset object in the lower layer will have affected asset objects in the upper layer. This would make asset objects in the concurrent system consistent by separately relatively designed. Second, a horizontal reusability can achieved. To develop various asset objects in different domain in the same layer, asset interface of that layer can

implement as many as asset objects to meet specified requirements. For example, a system asset of logging in the concurrent software is defined in the lower layer to record activity of the system every 2 seconds. This logging asset in the concurrent system can reuse and modify at the upper layer. The asset interface in the lower layer provides the abstraction of the lower logging asset. The concurrent aspect framework seems to promise a better degree of reusability in the design of concurrent software with naturally reusable.

## 5. Conclusion

This paper presents our approach for improving the concurrent software system using the aspect-oriented framework while maintaining the advantages of better reusability and adaptability. We propose an alternative technique for designing concurrent software system that present a higher level abstraction to the designer and the programmer. Instead of requiring the designer and the programmer to deal with the complex components of concurrent software system, we allow designers and programmers to modify and understand components simply. Concurrent software system, developed using this framework, is referred to as reusability and adaptability models.

We believe that aspect-oriented software development can assist developers and programmers to design the adaptable and reusable concurrent software. Aspect-oriented approach does not limit to any language either. A general feeling, that concurrent object-oriented design and development naturally helps reusability and adaptability, is a misconception. Concurrent object-oriented design does not naturally enforce reusability and adaptability. Relatively, developers of concurrent software have to carefully specify and design for reusability and adaptability supporting. The high level abstraction of relatively separated concerns better allows for reusability and adaptability. The framework promises on decomposition of concurrent object-oriented system. The framework can achieve and improved a better separation of concerns in the development of concurrent software system.

We achieve extraction both functional components and spreading concerns through a single component, called an asset component. A moderator component can coordinate the interaction of functional components and aspect components via joinpoints. System assets can reuse and adapt without interfering any functional components. Functional components can reuse and adapt without interfering any system assets as well. The concurrent aspect framework not only promises reusability and adaptability, but also lets developers find any proper language for the system. The framework can address various asset and functional components. This technique can reduce time and cost to develop concurrent software. A weakness of this approach is an increasing number of classes.

## References

[1] M. E. Fayad and A. Altman. An introduction to software stability, Communications of ACM, Vol.44, 2001; 9: 95-98.
[2] M.E. Fayad, M. Cline. Aspect of Software Adaptability. Communications of ACM, Vol. 39, 1996; 10:58-59.
[3] Dijkstra E. W. A Discipline of Programming. England-wood Cliff, New Jercy: Prentice-Hall; 1976.
[4] Parnas D. On the Criteria to be Used in Decomposing Systems into Modules. Communications of ACM, Vol. 15, 1972:12:1053-1058.
[5] Lopes C. V. and W. L. Hursch. Separation of Concerns. College of Computer Science, Northeastern University, Boston; 1995.
[6] R.E. Filman et al. Aspect-Oriented Software Development, Addison-Wesley; 2004.
[7] H. Ossher and P. Tarr. MultiDimensional Separation of Concerns in Hyperspace. Workshop on Aspect-

Oriented Programming in International European Conference on Object-Oriented Programmming; 1999.
[8] P. Netinant. Component + Aspect = an Extensible and Adaptable System Software. Proceedings of the International Conference on Software Engineering Research and Practices (SERP 2005), USA; 2005.
[9] Fernando Sanchez et al. Run-Time Adaptability of Synchronization Policies in Concurrent Object-Oriented Languages. Proceeding ofEuropean Object-Oriented Programming'98. Workshop on Aspect-Oriented Programming ; 1998.
[10] V. Abdelzad and F. S. Aliee. Aspect-Oriented Software Development Versus Other Development Methods. Journal of Theoretical and Applied Information Technology, Vol.30, 2011 ; 2 :147- 152.
[11] Johan Brichau et al. A Model Curriculum for Aspect-Oriented Software Development. IEEE Software, Vol. 23, 2006; 6:53 – 61.
[12] A. Rashid, T. Conttenier et al. Aspect-Oriented Software Development in Practice : Tales from AOSD-Europe. IEEE Computer, 2010; 19-26.