International Journal of Computer Science and Engineering (IJCSE)
ISSN(P): 2278-9960; ISSN(E): 2278-9979
Vol. 3, Issue 1, Jan 2014, 105-112
© IASET

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

EXPANDING ASPECT-ORIENTED PROGRAMMING TOWARD ADVANCED DYNAMIC NETWORK LEVEL ASPECTS IN C++

RAWAN KASASBEH, SUFYAN AL-MAJALI & ARAFAT AWAJAN
Department of Computer Science, Princess Sumaya University for Technology, Amman, Jordan

ABSTRACT

Programming models are continuously improving to achieve two principals, more modularity and less crosscutting concerns. The final improvement was the emergence of Aspect Oriented programming (AOP), AOP emerged to improve previous programming models rather than replace their work. The main improvement AOP has caused was the increase in program modularity, reduction of code redundancy and less code scattering. This is done through gathering the scattered code in one separate class called an aspect, weaving this aspect where needed according to point cuts, this way the redundant code is limited which results in better code modularization.

Our approach proposes the use of AOP along with networking, through allowing the introduction of a new aspect at runtime not only on a single computer, but at a network level. Therefore, there are two dimensions to our work, first allowing the aspect to be introduced at runtime gives better dynamicity and availability to the application, the program is dynamic since it is applied at runtime of the program and does not need to be introduced at runtime, it is available since there is no need to shutdown or restart the program in order to introduce the new code. The second dimension is adding this work at a network level.

Having a client server like network and having the option to add an aspect to the specified locations on some or all the client computers, weaving or unweaving the aspect at runtime. As mentioned this work increases dynamicity, flexibility and availability on the expense of system performance. Although the user should prioritize the most important requirements before using ADAC++ whether it is more dynamicity and availably or better performance.

KEYWORDS: Aspect Oriented Programming, Advanced Dynamic Weaving, Runtime Weaving, Weaving at Network Level Dynamic Weaving

Subject Classification: Computer Science Programming

INTRODUCTION

Previously in applications using AOP, the developer needed to introduce the aspect at compile time in order for the program to run properly. In this paper our work aims at allowing the introduction of the aspect at runtime instead of compile time. This contribution is intended to be done at a network level. With a client server network, each computer on this network has its own agent to facilitate communication between other clients and the server, to receive the aspect and load it into the application code at runtime. The aspect is first introduced and loaded to the server; afterwards it is distributed to the clients on the network. Our work is done on two different dimensions; the first is related to AOP. Through introducing the aspect to the application code at runtime. The second dimension is to apply this work on a network level rather than a single computer. The newly introduced aspect library is first introduced to the server computer by the administrator, the server distributes the aspect to the required clients on the network. The agents for each client gets the aspect and loads, weaves the aspect depending on the point cut existing in the program.

The computers on the network might all require the same feature to be added into their code through an aspect, whether they have the same applications or different applications. For example, if all clients require a security feature to protect the contents of an application, rather than going through the existing code for all the application classes of each computer on the network. It is more practical and flexible to introduce the code to the server computer through an aspect, distribute the aspect to the client computers, weave them according to their point cut at runtime. Following the user has the option of unweaving the aspect. Again, what is meant by dynamic in the title is to introduce an aspect dynamically while the application is already running, weaving at the network level where new aspects can be introduced, woven and unwoven at runtime. This work adds new properties such as more modularity, less crosscutting concerns and more availability for the application.

ADVANCED DYNAMIC AOP SYSTEM RUNNING ON C++ (ADAC++)

Advanced Dynamic AOP System Running on C++ (ADAC++) Approach

As mentioned, previous versions of AOP didn't allow the introduction of aspects at runtime. An aspect must be introduced at compile time in order for the application to run correctly. To improve this limitation, ADAC++ allows the introduction of aspects dynamically at runtime. This task is achieved by using DLL and polymorphism concepts. The aspect is first created as a DLL, following through the concept of polymorphism used in the virtual advice function, the aspects will be allowed to run. The result would be different each time depending on the function in DLL.

ADAC++ work is done by creating an aspect as DLL, this aspect is woven and loaded into the application code at runtime. It does not need to be previously defined, it is only introduced while the program is running. The unweaving process is the same, the aspect is unwoven while running the program. The improvement made by our approach, is adding the concept of polymorphism in the advice virtual function, where many DLLs perform the same function but with different results each time, depending on the function of the DLL.

Advanced Dynamic AOP System Running on C++ (ADAC++) Architecture

The following figure explains the ADAC++ architecture, where a customer requests adding a new concern, this request is sent to the developer which creates an aspect and compiles it as a DLL. This aspect is then deployed to the administrator, compiled into the server computer on the network. Following, an admin user will request to weave the aspect. Once this request is received by main computer, the aspect library is sent to the client machines. It will be received by each client agent. Applications installed on each client machine will synchronize with the agent and load received aspects into their memory space, weave them according to point cut expression, once they notice new aspects arrival at the agent.

The weaving command is a simple line of code written in the following method:

Weave (m): An unweaving request is done by a request form a customer to the administrator to unweave a specific aspect form the application code, afterwards the admin sends a command to the server computer to unweave an aspect from all the client computers or some of them, through a line of code such as the following:

Unweave (m): To edit an existing aspect, the intended plan is to first unweave the aspect in the same method mentioned previously, change on the content of the aspect, following inserting the aspect through weaving it to the code as shown in the following figure.

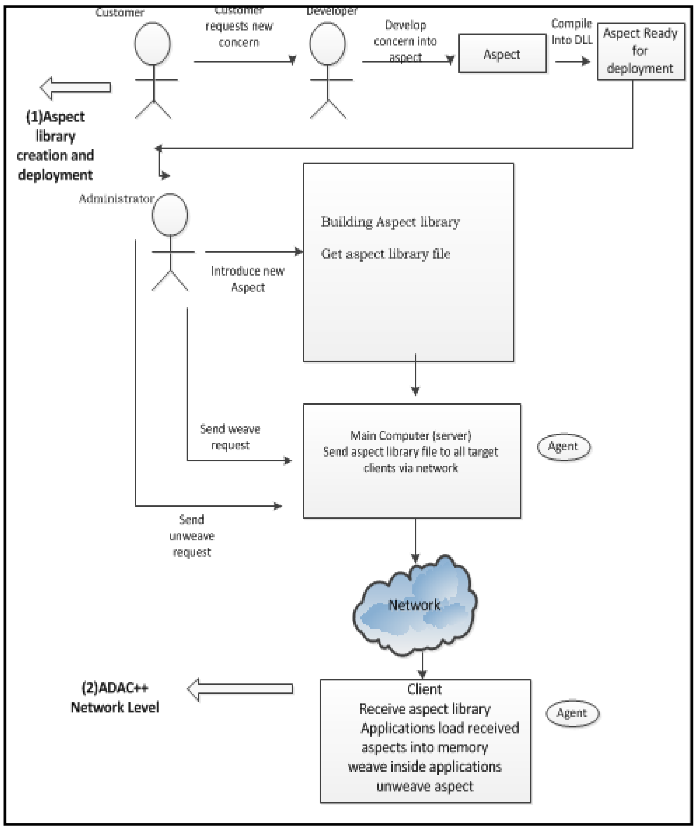

Figure 1: ADAC++ Framework Stages

The components of ADAC++ frameworks are as following:

- **Dynamic Aspect Generator:** This component of ADAC++ is used to apply the following. First depending on the end user requirements, the developer will build the aspect through a DLL, compile aspects and prepare them to be deployed dynamically within the network by the administrator.
- **Dynamic Aspects Library Store:** After the first step is done, the aspect is created and ready to be deployed. DLL Store represents the area in which all the created aspects are stored. This gives the user the ability to request weaving these aspects at any point in the application code. Any aspect can be woven if it is only available within store.
- **Management Station:** The main functions accomplished by the management station is to hold a database of information about the network. To communicate with its agents installed on the network hosts.
- **Weaver:** Weaving is the process of composing core functionality modules with aspects. Aspect languages have defined several mechanisms for weaving, including static weaving through inserting aspects at system load and weaving at runtime, or dynamic weaving through introducing aspect at compile time and weaving aspects at system runtime.
- **ADAC++ Network Protocol:** A network protocol defines rules and conventions for communication between network devices. Network protocols include mechanisms for devices to identify and make connections with each other, as well as formatting rules that specify how data is packaged into messages sent and received. Some protocols also support message acknowledgement and data compression designed for reliable and/or high-performance network communication. This component allows the Management station to communicate commands of weaving and unweaving within network.

- **Server Agent and Client Agent:** The agent is software that will be installed on every computer in the network including the server computer and all the client computers, every host agent stays in contact with MS using the network protocol in order to receive the weaving, unweaving aspect requests.

As an example that can benefit from our approach, is the logging concern. If the end user requested the addition of a log concern, a user should enter a username and password in order to enter the application. Giving each user certain restrictions, access limitations based on the job rank, this is done by adding a log concern. The new concern request will first be sent to the developer; the developer creates a logging aspect as a DLL, and sends it to the administrator in order to be inserted into the application code.

This aspect will be compiled into the server computer on the network. A request from the administrator will be sent to weave the logging aspect, once this request is received by main computer, the aspect library is sent to the client machines, will be received by each client agent. Applications installed on each client machine will synchronize with the agent, and load received aspects into their memory space. Weave them according to point cut expression, once they notice new aspects arrival at the agent. Again the logging aspect will be added to the application code while it is still running; it is woven depending on the point cuts and the advice function. It runs before or after a certain point cut in the code as specified in the main of the program.

Aspect Oriented Programming at a Network Level

In this part we explain the network dimension of our work in ADAC++. Computer networks and network applications are growing and changing continuously. New network applications with complicated network and software requirements are emerging. For instance, wireless networks both local area networks and wide area networks are becoming common. The applications associated with these network types require great dynamicity from the underlying network and software technology used to build them. Two main factors impact the network's flexibility toward deploying new policies to an existing network: the software design, and the network protocols used for deploying new software changes [3].

In ADAC++ the network is a client server like network, with both clients and servers having agents in order to communicate with each other, and receiver new commands, such as weaving and unweaving aspect , modifying existing aspects, etc. The computers on the network might all require the same feature to be added into their code through an aspect, whether they have the same applications or different applications.

For example, if all clients require a security feature to protect the contents of an application, rather than going through the existing code for all the application classes of each computer on the network. It is more practical and flexible to introduce the code to the server computer through an aspect, distribute the aspect to the client computers, weave them according to their point cut at runtime. Following the user has the option of unweaving the aspect. Again, what is meant by dynamic in the title is to introduce an aspect dynamically while the application is already running, weaving at the network level where new aspects can be introduced, woven and unwoven at runtime. This work adds new properties such as more modularity, less crosscutting concerns and more availability for the application since the weaving process does not require restarting or shutdown.

An example is given in Figure(3), where 4 different computers are in a certain network, through ADAC++ aspects can be woven to all computers, one computer or a part of them. Also unweaving of a certain aspect can be done to all computers , one computer or some of them. This is done at runtime by giving commands to the agents that are given for each computer.

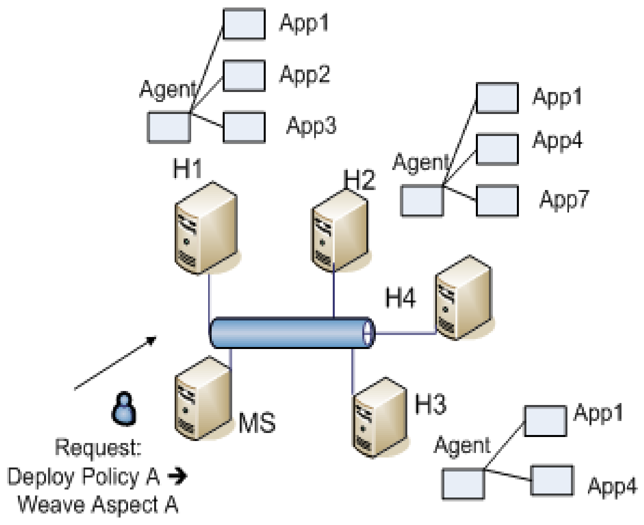

Figure 2: Networking in ADAC++

EVALUATION OF ADVANCED DYNAMIC AOP SYSTEM RUNNING ON C++ (ADAC++)

The evaluations aim is to view the advantages of ADAC++ over previous AOP versions. As mentioned previously ADAC++ has benefits, of flexibility for the developer to add new aspects to the application code while running or modify existing aspects to add a certain function or delete or modify one. On the other hand it is worth mentioning that AOP in general has limitations, the main two are the following:

The limitation which is concerned for ADAC++.While ADAC++ improves flexibility and dynamicity through allowing the introduction of the aspect containing the new code at runtime, without the need to restart or cause system shutdown. An important downfall to this is lower system performance for the program using ADAC++, again although dynamicity and flexibility issues are solved in ADAC++, its worth mentioning this is done at the expense of performance, since the program will function slower than without using ADAC++ In the evaluation of ADAC++ was shown that this approach increased flexibility and dynamicity through introducing a new concept of introducing the aspect at runtime of the program at the network level, but this work lowers the performance and increases the complexity of the code at the same time. While ADAC++ solves some problems, before using this approach the end user should prioritize his concerns, whether it is system dynamicity and flexibility or better performance and less complexity. To prove this work a comparison between static, dynamic and advanced dynamic weaving was held in Table 1.

Table 1: Comparison between Static, Dynamic, Advanced Dynamic Weaving

	Static Weaving	Dynamic Weaving	Advanced Dynamic Weaving
Aspect introduction time	Aspect introduced at compile time	Aspect introduced at compile time	Aspect introduced at run time
Aspect Weaving	Aspect is woven at compile time	Aspect is woven at compile time	Aspect is woven at runtime
Point cut definition	Needs multiple point cuts for multiple aspects	Needs multiple point cuts for multiple aspects	Needs one point cut for multiple aspects
Aspect unweave	Not applicable	Can be unwoven	Can be unwoven
Frameworks	Aspect J, Aspect C++	JBOSS,PROSE, DAC++	Our Framework
Application Maintenance effort	Large	Medium	Minimum
Dynamicity	Not Dynamic	Moderate	High
Flexibility	Not Flexible	Moderate	High

SUMMARY AND CONCLUSIONS

AOP is a relatively new programming paradigm, striving to help the developer separate concerns to overcome the problems with crosscutting concerns, which improves modularity, to simplify the code and ease the development and maintenance. What OOP has done for object encapsulation and inheritance, AOP does for crosscutting concerns.

AOP helps overcoming the problems caused by code tangling and code scattering, which lead to implications of lower productivity, hard reuse of code and evolving the system. Also it is also easy to add newer functionality, by creating new aspects.

In this paper, the contribution was adding a new aspect to the application code dynamically at runtime, where the aspect is woven and unwoven at runtime, unlike previous AOP versions, where the aspect should be introduced previously to the application for it to work properly. Another importance of this thesis is it works in a rigid language which is C++, attempting to make it more dynamic.

REFERENCES

1. Alam, F., Evermann, J., Fiech, A.: Modeling for dynamic aspect-oriented development, Proceedings of the 2nd Canadian Conference on Compurer Science and Software Engineering, (2010).
2. Almajali, S., Elrad, T.: Coupling Availability and Efficiency for Aspect Oriented Runtime Weaving Systems, Dynamic Aspects Workshop, pp. 47-55, (2006).
3. Almajali, S. , Elrad, T.: A dynamic aspect oriented system using C++ programming using MOP, Dynamic Aspects Workshop, pp. 1-8 Key, (2004).
4. Almajali, S., Elrad, T.: Dynamic Network Policies Using Aspect Oriente Network Framework, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, (2006).
5. Ansaloni,D., Binder, W., Moret, PH, Villazon, A.: Dynamic Aspect-oriented programming in java, Faculty of Informatrics, University of Lugano, Switzerland, (2012).
6. Apel, S., Batory, D,: How Aspect J is Used An Analysis of Eleven Aspect J Programs. Department of Informatics and Mathematics University of Passau, Germany, (2008).
7. Assaf, A., Noye, J.: Dynamic Aspect J, DLS '08 Proceedings of the 2008 symposium on Dynamic languages, (2008).
8. An Oracle White Paper: Comparing Oracle Glass Fish Server and J Boss: Which Application Server Is Right for You, (2010).
9. Chiba, S.: A Study of Dynamic Weaving for Aspect-Oriented Programming, A Dissertation Submitted to Department of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, (2005).
10. Dyer, R., Rajan, H.: Supporting dynamic aspect-oriented features, Iowa State University, Ames, (2010).
11. García, M., Llewellyn-Jones, D., Ortin, F., Merabti, M.: Applying dynamic separation of aspects to distributed system security, IET Software, Volume 6, Issue 3, (2012).

12. Kienzle, J., AlAbed, W., Fleurey, F., Jezequel, JM., Klein, J.: Aspect-Oriented Design with Reusable Aspect Models, INRIA, Centre Rennes - Bretagne Atlantique, Rennes, France.CRP Gabriel Lippmann and University of Luxembourg, Luxembourg, (2010).

13. Katic, M., Fertalj, K.: Model for Dynamic Evolution of Aspect-Oriented Software, Software Maintenance and Reengineering (CSMR), European Conference, (2011).

14. Laddad, R: Aspect J In Action. Second Edition, Manning Publications, chapter ten,(2010).

15. Ortiz, G., Bordbar, B., Hernández, J.: Evaluating the Use of AOP and MDA in Web Service Development, Internet and Web Applications and Services, third International Conference, (2008).

16. Ostermann, K., Mezini, M.: Object-Oriented Composition Untangled. Published in OOPSLA '01 Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Pages 283- 299, (2001).

17. Schroder-Preikschat, W., Lohmann, D., Scheler, F., Gilani, W., Spinczyk, O.: Static and dynamic weaving in system software with aspect++, System Sciences, Proceedings of the 39th Annual Hawaii International Conference, Volume.9, (2006).

18. Tartler, R. and Lohmann, D. and Scheler, F. and Spinczyk, O.: Aspect C++ An integrated approach for static and dynamic adaptation of system software, riedrich-Alexander University, Erlangen-Nuremberg, Germany, (2010).

19. Tartler, R., Lohmann, D., Schroder-Preikschat, W, spinczyk, O.: Dynamic Aspect C++ Generic Advice at Any Time, Proceedings of the 2009 conference on New Trends in Software Methodologies, (2009)