DISTRIBUTED SOFTWARE INTEGRATION MODEL BASED ON ATTRIBUTE-DRIVEN DESIGN ADD METHOD

Hamid Mcheick[1], Yan Qi[1] and Hani Charara[2]
[1] *Department of Computer Science and Mathematic, University of Quebec at Chicoutimi*
UQAC, 555 Boulevard de l'Université Chicoutimi, G7H2B1, Chicoutimi, Canada
[2] *Department of Computer Science and Mathematic, Lebanese University, Faculty of Science I, Lebanon Hadath, Lebanon*

Keywords: Connector, ADD.

Abstract: Software integration in distributed architecture plays an important role to improve software quality. Engineers often face challenges coming from connectors. Especially, design of connectors in a distributed system encounters more design issues such as: i) how to fulfil the functional and quality requirements of connectors in distributed system; ii) how do multiple technologies combine together to resolve design issues of connectors. In order to design connectors in distributed software, this paper proposes a design model by using ADD methods. And then we give an example of MVC architecture for presenting how to instantiate that model based on ADD design method.

1 INTRODUCTION

Software components and connectors are two important parts of the software architecture. Connectors are regarded as software elements for delivering data and control in a software system (Base et al. 2003).

In software development, there are some simple connectors, such as procedure call, association class, etc. These connectors are widely used and some of them become general programming approaches (procedure call) provided by most of development tools. The connectors are good at linking components together in local environment.

However, in the recent years network technologies and distributed system are rapidly growing. In distributed architecture the components are distributed in separate computers over the network and the traditional approaches (for example, the method-based mechanism) are not directly available for interacting with the distributed components (Qiu, 2005). In order to design the connectors in distributed system, multiple technologies are applied in practice. However, when facing these technologies, developers often meet some problems. For example, messaging system (middleware) is used to deliver messages in network for constructing connectors. But they do not have the ability to build the relationship of components. On the other hand design pattern is most effective for building relationship when combined with aspect-oriented programming (AOP). However, AOP only provides the mechanisms for developer to weave aspects and base code together into a coherent program (Elrad et al. 2001). Thus, AOP is merely applied to handle classes situated in one local application. As a result of this property, AOP cannot be directly used in distributed system. Furthermore, the qualities of connectors may be seldom discussed in the design process of connectors, particularly when these technologies are put together for solving problems.

In our research, we focus on these design issues: in order to fully satisfy both the functionalities and qualities of connector, how to analyze and design connector in distributed architecture by weaving the relevant technologies together (design pattern, aspect-oriented programming, middleware, network protocol etc.)

In the following sections, we firstly present background related to connectors (section 2). Secondly, section 3 develops a general design model and an example scenario for connector in distributed architecture by following ADD method. Thirdly, we draw a conclusion in section 4.

Mcheick H., Qi Y. and Charara H..
DISTRIBUTED SOFTWARE INTEGRATION MODEL BASED ON ATTRIBUTE-DRIVEN DESIGN ADD METHOD.
DOI: 10.5220/0003506500530058
In *Proceedings of the 6th International Conference on Software and Database Technologies* (ICSOFT-2011), pages 53-58
ISBN: 978-989-8425-76-8

2 BACKGROUND

2.1 Middleware in Distributed Systems

Middleware in distributed system is a dedicated software which is designed to deliver messages between components without involving knowledge of network protocol and hardware platform. Distributed system frequently applies two kinds of middleware to interactions of components: object oriented and message oriented middleware.

Object oriented middleware (OO middleware) is applied to component interaction in distributed system. It provides an easy programming model for the interaction of distributed programs by using current programming language features (Pryce, 2000), such as CORBA (OMG, 2004) and Java RMI (Remote Method Invocation). But there are some disadvantages: they are only suitable for client/server architecture; especially, it is hard to apply them to complex distributed systems.

Message oriented middleware (MOM) is also called as messaging system. It is a separate software system that provides messaging capabilities for distributed components (Hohpe et al., 2004). In contrast to OO middleware, MOM implements the interactions of components by formatting them as messages instead of modelling the interaction as procedure calls. MOM provides users with the capability of providing stable communications in an unstable network environment.

The two kinds of middleware work towards a common goal: sending/receiving data over network. However they cannot be used to describe the relationship of components and they have no idea about the message construction.

2.2 Design Pattern

Design patterns are often used to describe relationships between components (classes or objects). The design patterns in Gamma et al.'s book (1995) are descriptions of communication of objects and classes that are customized to deal with general design problems. The communication of object and class should be loosely coupled without becoming entangled in each other's data models and methods (Cooper, 2000). Gamma et al.'s design patterns particularly deal with problems at the level of software design, especially object-oriented software design. They can be classified by criterion scope which is used to specify whether the pattern is mainly used to classes or objects. Thus, the design patterns have two kinds: class design pattern and object design pattern (Gamma et al., 1995). Object patterns are applied to object relationships, which can be modified at run-time and are more dynamic, such as Proxy Pattern, Observer Pattern, Publish-Subscribe and so forth.

Design pattern is specially used to design the dependency of components. It cannot be directly applied to distributed architecture, if it is does not have a proper transport mechanism.

2.3 Aspect-oriented Programming

Aspect-oriented programming (AOP) is combined with object-oriented programming (OOP) for achieving a basic objective (Gradecki et al., 2003): describe and divide the concerns by crosscutting components. The core mechanism of AOP uses an aspect weaver that is a compiler-like tool. This tool compiles the whole program by combining the crosscutting concern (aspect) with other classes and the compiling process is defined as weaving in AOP (Laddad, 2003). Therefore, the working mode of AOP technology is static. The whole program cannot be dynamically changed after weaving them.

According to the properties of AOP, it has the capability to build relationships (following a design pattern) between separate components without modifying much of the existing source code of components. As a result of this point, AOP enables the developers to easily design, implement, extend and maintain software system.

Despite this, AOP has a major limitation. It is merely applied to classes situated in one local program. Thus, AOP cannot be directly used in distributed system and the use of AOP across network is still a challenge for developers.

2.4 Attribute-Driven Design (ADD)

In software engineering some design methods are defined to model software engineering activities. For example, both the attribute-driven design method and object-oriented (OO) methods belong to software engineering design methods. OO method merely focuses on the functionalities of software system.

Attribute-Driven Design (ADD) method is developed by the Carnegie Mellon Software Engineering Institute (SEI). In contrast to OO method, ADD is an approach to designing a software architecture in which the high level design process relies on the software's quality attribute (Wojcik et al. 2006). ADD method follows a recursive design process: 8 steps (Wojcik et al. 2006) that are iterated

until all architecturally important requirements are satisfied (Bass et al. 2003).

The input of ADD involves functional requirement, design constraint and quality attribute requirement; the output of ADD is the high levels design of some views of architecture. The views of architecture include module decomposition, concurrency, and deployment. Quality attribute requirements show the different properties which a system must exhibit; functional requirements indicate what functions a system must be provided for stakeholder's needs when the software works under some conditions; design constraints are decisions about a system's design that must be involved into final design (Wojcik et al. 2006).

Architectural drivers are the combination of functional requirement and quality requirements which are used to form the architectures and modules (Bass et al. 2003). Architectural patterns and tactics are applied to satisfy the quality attributes which are used to define types of elements and interactions by following the step4 of ADD, while functionality requirements are chosen to instantiate the module types (in step 5 of ADD).

Bass et al.'s research (2003) addressed the system quality attributes of software, such as availability, modifiability, performance, security, testability, usability, and so forth. In order to achieve quality attributes of connector in distributed system, these qualities should be considered when designing, implementing connector.

3 DESIGN MODEL FOR DESIGNING CONNECTOR

In this section we provide a general design model for designing connector by using ADD method in distributed architecture. And then, we give an example scenario about design of connector in distributed MVC (Model, View and Controller) architecture.

3.1 Model for Designing Connector

Connectors which are situated in distributed system must be designed to satisfy the demands: i) the transport of information; ii) description of relationship; iii) message construction. According to the properties of the distributed connectors, we propose a general model for designing connector using ADD design method. The design process following the standard of SEI's ADD design method is described below.

Step 1: collect the architectural drivers: specific quality scenarios functional requirements and design constrain (Table 1).

Step 2: choose the module to decompose. We consider the whole connector as the primary element.

Step 3: identify chosen architectural drivers. For the design model, the quality attribute: Availability and Modifiability are high priorities.

Step 4: choose the patterns and tactics to satisfy the architectural drivers (Table 2). In light of the architectural drivers, we propose a general layered model in which each layer corresponds to different modules for meeting the functional requirements and achieving quality attribute of connectors. The layered model consists mainly of Transport Layer, Dependence Layer and Presentation layer.

Table 1: Architecture drivers.

Quality Attribute	Scenarios
Availability	Connectors can also deliver the messages when the network is in bad situation or even the component is shutdown.
Modifiability & Scalability	When designing and implementing a new connector, the existing design and implementation of components should not be modified too much.
Performance	Connectors can provide efficient service for the delivery of messages.
Security	The connector can protect the messages and data transferred against the attack by an unauthorized attempt.
Testability	Most parts of design and implementation of the connector can be independently tested and verified. The internal state and inputs of the connector can be easily controlled and then the outputs of connector should be also observed.
Functionality	
Connectors must link components and transfer data and control by using some formatted information according to the dependence (relationship) between components	
Constraints	
Components which are linked by the connector can be situated in network or one application. The platform in which the components are situated must be Windows and Linux compatible.	

Table 2: Architectural patterns and Tactics.

Quality Attribute	Architectural Patterns & Tactics
Availability	Messaging systems
Modifiability	Aspect (AOP)
Scalability	Messaging systems; Design Patterns
Performance	Socket; Reduce overhead
Security	Secure Sockets Layer; Encryption
Testability	Separate interface; Indenpent module

Transport layer (module) situated in the base of the connector is responsible for basic transceiver in network. It is used to "carry" the dependence or relationship between components by linking them together. Normally, it is implemented by network protocols (such as TCP/IP, HTTP protocol stacks) or MOMs (WebSphere MQ/ Open Message Queue etc.). MOM can satisfy the Availability and Scalability, because that MOM has the ability to provide the reliable communication and support high scalability for a large number of clients. However, there is a side effect for MOM: poor Performance. Thus, if the Performance is very important to the system, the socket API (TCP or UDP) should be chosen. Particularly, secure socket layer can provides Security for transfer.

Dependence layer (module) describes the relationship between components in detail. In order to build the relationship of components, we can use object design pattern or architecture pattern to implement it. AOP and design pattern are good at building the relationship without modifying much of existing components. Therefore, they can satisfy the Modifiability.

Presentation layer (module) is responsible for constructing the transferred information depending on the type, format and amount of the information. We can use Markup language such as XML or a customized format defined by developers according to the type and amount of the information. Secure encryption of data and messages are always required in this layer. Therefore, it can satisfy the Security.

3.2 Example Scenario: Design of Connector

In this subsection, we design architecture of connector for an example scenario: a distributed MVC software system by using our general design model. And then, decompose the general architecture into concrete modules by using ADD method.

The MVC architecture (Figure 1) shows: the component Controller can send messages to all Models and Views; the Model A can send messages to View 1 and View 2; Model B can send messages to View 2 and View 3. Based on this structure, the MVC structure can provide active working model, because when the data related to Model has been changed, the Model can update the registered Views without notifying Controller.

According to the standard of SEI's ADD and requirement of connectors, the steps 1- 4 have been completed in the design process of the general design model described in subsection 3.1. Then, we start to discuss the design process from Step 5 of ADD.

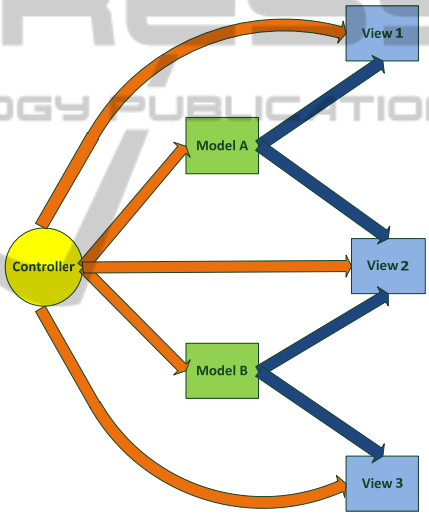

Figure 1: Example Scenario: MVC architecture.

Step 5: instantiate architectural modules and define responsibilities. In this step, we instantiate the modules analysed in step 4.

In the example, there are two kinds of connectors: i) connector between Controller and other components (Models and Views); ii) connector between Model and View. The relationship between Controller and others components is the same as the relationship between Model and Views. They both follow the Publish-Subscribe pattern. For example, Controller publishes messages/command to Models and Views; Views and Models can receive the message they subscribe to.

i) In presentation layer, module writer and parser are utilized to process messages by adopting the message protocol of access (Figure 2).

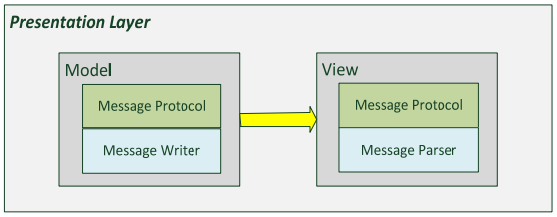

Figure 2: Presentation layer of connector.

Message protocol is the meaning or logic of the message. For example, if one component wants to access a database server, it may use a SQL code as the logic of messages. Message writer/parser handles the format of message, such as the package/un-package data, encryption/ decryption, compression/ decompression, etc.

ii) In dependence layer, we apply an Aspect of AOP and design pattern to build the relationship of components. In figure 1, it shows an active mode MVC. When the data maintained in Model is changed, the Model must notify the Views of the changes. Thus, Publish-Subscribe pattern is suitable to build the relationship between Model and View. For the detailed design, we use Observer design pattern to implement the Publish-Subscribe and apply AOP to implement the Observer design pattern. The reason why we prefer utilizing AOP is analysed below: the component Model should concentrate on the data processing of application and all the functionalities of Model should be separated from all things related to display and user interfaces that should be done by component View. In other words, the developers of Model do not need to know when to notify Views and what data should be delivered to Views. Fortunately, the AOP enables developers to build the relationship between two components (Model and View) by crosscutting the two components without modifying much of the logic of both components. Particularly, when planning to update one MVC architecture by adding a new connector between components (for example, a new View wants to get some information from one existing Model.), the developers can get maximum benefit from the usage of AOP.

We show a diagram about the design of the dependence layer (Figure 3).

Figure 3 shows that a new component Proxy is added. The component Proxy and Model are situated in one program that runs in a same computer. That means that they share the same memory space and CPU. One purpose of Proxy is to represent the

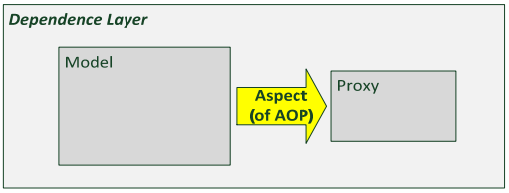

Figure 3: Dependence layer of connector.

component View in dependence layer. The other purpose of Proxy will be described in transport layer. In figure 3, we also note that another special module is instantiated. It is named as an Aspect which is used to crosscut the two components (Model and View). In term of the Observer design pattern, we make the Model work as Subject and make Proxy work as Observer. In term of the architecture of Figure 1, multiple Views can be supported as Subject. Then the dependence (relationship) is built between the two components (Model and View).

iii) In transport layer, in light of the quality of network and the requirements of whole system, developers choose the network protocols or MOM to implement the task of transport in network.

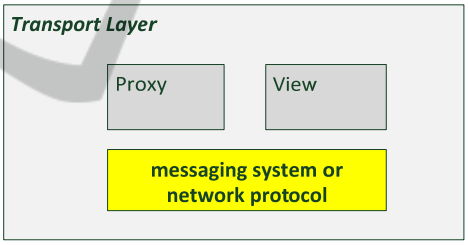

Figure 4: Transport layer of connector.

In the Figure 4, the component Proxy is described in the design of dependence layer (shown in Figure 3). Here, we talk about another purpose of the Proxy. It is used to send message to Views by using an instantiated module (messaging system or network protocol). The component Proxy plays an important role in our solution. It works as an Observer to “monitor” the Model component. When the Model changes its data which interests the Proxy, the Proxy will be updated automatically in dependence layer. In addition, it is responsible for transferring the updated data to Views relying on the module messaging system or network protocol.

In this Step, we instantiate the modules of connector based on the MVC structure by taking connector between Model and View for example. Another type of connector between Controller and other components can be designed in the same way.

After Step 5, the general model for connector in

the MVC example is instantiated. According to ADD methods, the Step 6 (define interfaces) and Step 7 (refine requirements) should be done in an actual project by meeting different requirements.

3.3 Analysis of the Design Model

This design model presents a clearly layered structure of connector which fulfils the architectural drivers by using ADD methods. Each layer is responsible for dedicated requirements. The developers can be guided through the process of design of connector by using the design model.

Compared to other approaches of connector, this model covers every aspect of connector in distributed system. For example, in some research works, network protocols (TCP/IP, P2P etc…) are regarded as connectors. This misunderstand can lead developers to only concentrate on the functionality of transport, neglecting to study and design the relationship of components.

Relationship is the core of the model. When using this model, developers must put a lot of effort into the relationship of the components. To simplify the analysis of relationship, the pattern approach (design pattern by using AOP) is applied to the design of connector in distributed system. Messaging system and other transport mechanisms are designed to ensure that the messages can be successfully delivered. In our opinion, the messaging system should be situated in transport layer of this model, even though some of messaging systems have the function of description of independency of components.

4 CONCLUSIONS

This paper discusses a design model of connectors in distributed architecture. We mainly concentrate on the design problems of connectors: how to gain the benefit from the combination of multiple technologies (design pattern, AOP, MOM and network protocol) for satisfying the functional requirements and achieving quality attribute. To resolve the problems, we propose a general design model for design of connector in distributed architecture based on ADD method. And then we give an example of distributed MVC architecture for presenting how to instantiate that model based on ADD. In that example, our design model enables us to easily design and develop the connectors for coherently linking the Views and Modes together and fulfilling the functional and quality requirements of connectors.

The presented approach is at an early stage of development and would certainly merit being more detailed especially for generalize our model to facilitate the maintenance and reduce the cost of distributed applications.

ACKNOWLEDGEMENTS

This work was sponsored by the University of Quebec at Chicoutimi (Quebec), and by NSERC (Canada).

REFERENCES

Bass, L., Clements, P. & Kazman, R., 2003. *Software Architecture in Practice*, Addison-Wesley, Second Edition.

Qiu, X., 2005. *Message-based MVC Architecture for Distributed and Desktop Applications.* Syracuse University PhD.

Elrad, T., Filman, R. E. & Bader, A., 2001. *Aspect-oriented programming: Introduction.* Magazine Communications of the ACM Volume 44 Issue 10, Oct. 2001 ACM New York, NY, USA.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson P., Nord R. and Wood, B., 2006. *Attribute-Driven Design (ADD),* Version 2.0. TECHNICAL REPORT, CMU/SEI-2006-TR-023, ESC-TR-2006-023.

OMG (Object Management Group), 2004. *The Common Object Request Broker: Architecture and Specification*, Version 3.0.3.

Pryce, N. G., 2000. *Component Interaction in Distributed Systems.* PhD Thesis. Imperial College of Science, Technology and Medicine.

Hohpe, G. & Woolf, B., 2004. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.

Gamma, E., Johnson, R., Vlissides, J. & Helm, R., 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Cooper, J. W., 2000. *Java Design Patterns: A Tutorial*. Addison-Wesley.

Gradecki, J.D. & Lesiecki, N., 2003. *Mastering AspectJ: Aspect-Oriented Programming in Java*. Wiley.

Laddad, R., 2003. *Aspect J in Action: Practical Aspect-oriented Programming*. Manning.