Asian Journal of Information Technology 6 (6): 681-684, 2007


# Collaborative Software Architecture

[1]Deepak Laxmi Narasimha and [2]M.Y. Sanavullah
[1]Vinayaka Missions University, Periya Seeragapadi, Salem-636308, Tamil Nadu, India
[2]Department of EEE, KSR College of Technology, Trichengode-637209, Tamil Nadu, India

**Abstract:** Modern software architecture becomes more and more complex as the number of properties, features and the requirements are increasing. Software application developer requires to know a number of concerns like functional and non-functional properties of the application during the application development process. Here an attempt is made to develop a software architecture using four different software engineering methodologies namely object, agent, aspect and component on a common platform. Studies show that traditional software engineering methods like Object Oriented Software Engineering (OOSE), Agent Based Software Engineering (ABSE) and Component Based Software Engineering (CBSE) are not efficient enough for the development of large enterprise application. Therefore, a new method i.e., Aspect Oriented Software Engineering (AOSE) combining all the above four software engineering methods has been proposed in the study. AOSE has been found to be useful to remove the concerns associated with the other three software engineering methods. Further aspects in AOSE allows software engineering properties to move from one model to another through weaving. A methodology is proposed to develop an advanced ASPECT-J2EE platform and model based on modification of J2EE and EJB component model. Thus, the combination of these 4 software engineering methods results in collaborative software architecture i.e., Advanced Aspect-Java 2 Enterprise Edition (AD ASPECT-J2EE).

**Key words:** OOSE, ABSE, CBSE, AOSE, EJB, J2EE, design patterns, AD aspect J2EE

## INTRODUCTION

Researchers focus on one method of software development. They try to remove the drawbacks from the individual methods only. Recently researchers realize the application of integrating the software engineering methodologies. Of late there have been many works relating to convergence of the software engineering methods and techniques. The convergence of software engineering is one area where more work needs to be done. The framework evaluation and performance analysis is another area to evaluate the above work. Keeping in view of both the requirements, research has been carried out to combine four existing software engineering methods in a framework. The collaborative environment, model and framework promises to provide the application of all the 4 software engineering methodologies and techniques.

In this study, research is also being carried out to evaluate a J2EE Presentation Tier design patterns for analyzing the software architecture. Aspects help in achieving high modularity, security and manageability through clear separation of concerns in component models. J2EE and EJB component based model are widely acceptable platform for enterprise application, but study shows that there are number of concerns (functional and non-functional properties) which are tangled and mixed up during the coding and execution phase. This gives inefficient results in final software application. Therefore, these concerns are required to be separated. Aspect and agent technique help in removing these concerns and allows to add new properties into the system. Aspect works on divide and rule policy, whereby business logic and technical properties are separated at first and weaved at later stages. Thus, it decomposes the system into modules and later composes to form application. There are large numbers of software are available in the market. The complexity associated with the software is also very high, because each application requires specific code and programmer needs to write the code each time, to develop the application. Component Based Software Development (CBSD) using EJB, Java Beans and CORBA is a solution in this field, but they have also showed their inefficiency. Researchers have found that if the aspects are introduced or weaved into the object, agent or component software methods separately to remove a number of concerns from the application development process.

**Corresponding Author:** Deepak Laxmi Narasimha, Vinayaka Missions University, Periya Seeragapadi, Salem-636308, Tamil Nadu, India

Proposed system specifies how aspect helps in obtaining clear separation of concerns, achieving higher modularity, distribution and easy plug-in and plug-out facility during run time. Aspects allow new agent properties like communication, interaction, autonomy and adaptation to be added into the system. A methodology is proposed to develop an advanced Aspect-J2EE platform and model based on modification of J2EE and EJB component model.

**Concerns in object oriented software development:** Hachani and Bardou (2002), Janzen and Volder (2004) have discussed the problems associated with object oriented software development. They proposed different methods to handle this problems or concerns in programming codes and design patterns. Unfortunately these papers do not provide any concrete solution for effective handling of separation of concerns in object-oriented software development.

**Concerns in agent based software development:** Kolp *et al.* (2001) have examined the various concerns associated with Agent and Multi Agent Systems (MAS). Krutisch *et al.* (2003), Kulesza *et al.* (2004), Garcia (2002) and Trilnik *et al.* (2002) have emphasized on agent-component, aspect-agent and agent-aspect architecture, respectively to deal with these concerns in MAS. Decker *et al.* (1996) and Iglesias *et al.* (1998) describe the failure of MAS to achieve a common goal.

**Concerns in component based software development:** Concerns in component based software development are specified in numerous studies. Mehta *et al.* (2000) and Aoyama (1998) have emphasized on a common taxonomy and definition for all the components and the component life cycle. Aspects in CBSD based system are studied by Chavez *et al.* (2004). They have used MDA with reference to CORBA to combine application architecture, platform and framework.

**Concerns separation using aspect oriented software engineering:** Grundy has specified life cycle and phases of component and specified the adjustment to be made in component to accommodate aspects. AOSD and AspectJ have discussed the AspectJ programming to develop enterprise applications. However, this study again unsuccessful to bring out the best of aspect and component in application development context. These study do not address many concerns.

**Inference drawn from literature survey:** The detailed study clearly demonstrate that there is a lack of one software architecture supporting and incorporating all the four software engineering methodologies.

## MATERIALS AND METHODS

The aim of the study is to combine all the four software engineering methodologies in a common platform as shown in Fig. 1. Following steps are involved.

- Design for agent-component
- Design for agent-aspect
- Design for agent-object
- Design for aspect-component
- Common CBSD (J2EE and EJB based application)

Figure 1 illustrates the proposed method and principle.

**Method:** Collaborative usage of the four software engineering methodologies as shown in Fig. 1 and Table 1.

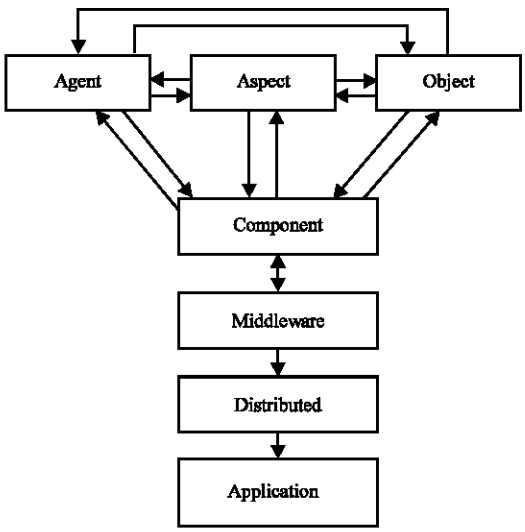


Fig. 1: Collaborative software architecture

Table 1: Properties and application features that are expected to obtain in the final platform

| Aspect | Agent | Object | Component |
|---|---|---|---|
| Cross-cutting | Adaptability | Modularity | Distributed services |
| Metadata | Mobility | Overloading | Reusable |
| Introduce | Communication | Polymorphism | Prefabricate component |
| Point cuts | Interaction | Inheritance | Highly adaptable |
| Join point | | Encapsulation | |
| Advice | | Abstraction | |

Object+agent+aspect+component => AD Aspect J2EE Aspect+J2EE=>Aspect-J2EE Programming language

Table 2: J2EE presentation Tier design pattern implementation using aspectJ2EE

| | Presentation tier | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Coupling | | Cohesion | | Code size | | Manageability | |
| Pattern | Before (%) | After (%) | Before (%) | After (%) | Before (%) | After (%) | Before (%) | After (%) |
| Intercepting filter | 52 | 50 | 60 | 65 | 38 | 29 | 43 | 51 (18) |
| Context object | 53 | 51 | 60 | 64 | 42 | 37 | 46 | 41 (-9) |
| Front controller | 34 | 34 | 41 | 50 | 46 | 34 | 35 | 41 (20) |
| Application controller | 54 | 50 | 43 | 49 | 55 | 45 | 54 | 46 (-17) |
| View helper | 45 | 39 | 47 | 40 | 34 | 35 | 32 | 23 (-35) |
| Composite view | 34 | 34 | 47 | 50 | 35 | 30 | 37 | 45 (20) |
| Dispatcher view | 61 | 55 | 35 | 46 | 49 | 35 | 38 | 24 (-30) |
| Service to worker | 50 | 46 | 35 | 45 | 47 | 40 | 42 | 52 (16) |

J2EE presentation tier design pattern implementation and analysis using aspectJ2EE

## RESULTS AND DISCUSSION

Table 2 illustrates the application of using collaborative architecture, framework and model i.e. AD AspectJ2EE as compared to J2EE. This is demonstrated by implementation of J2EE Presentation Tier Design Patterns using AspectJ2EE Language. The J2EE design patterns are examined using quantitative and qualitative tools for coupling, cohesion, code size and manageability.

Results shows that some J2EE design patterns are highly beneficial, other are lesser and least beneficial, while remaining show no change and negative impact of the software architecture used maximum (51-70%): composite view. Minimum (1-49%): Intercepting filter, composite view and service to worker. No change (0% or -ve result): Application controller, context object, view helper and dispatcher view.

**J2EE presentation tier design pattern analysis using aspectJ2EE:** Table 2 shows the behavior of various J2EE design patterns in collaborative AD Aspect J2EE environment. The table shows the comparisons before and after the implementation of J2EE design patterns using Aspect J2EE Language.

Aspect oriented software engineering is still in the primary stage. New properties and features are being developed and added with every new version. The software requirements are very high and demanding. Software industry can not wait for the new versions and updates. Hence, a new collaborative architecture is needed which provide the application of all the modern software engineering methods, properties and features. It must address all necessary requirements of a software application and must not be complex at the same time. Further, there are many applications where Agents are required for specific intelligence and for knowledge, interaction, adaptability and autonomy. These requirements are not fulfilled by J2EE or AspectJ software, because this software is not developed to handle such applications. Their usages are limited. While AD Aspect-J2EE is developed to accommodate the above requirements. Designs principles use heterogeneous design patterns and style to adopt the various components into the system.

## CONCLUSION

- EJB based component platform are highly beneficial as a result of introduction of Aspect and Agent. The heterogeneous application development using this platform help programmers to develop the enterprise application in areas beyond the traditional object oriented and component based software development.
- This Framework allows large number of Agent-Component (AC) to easy plug in into the EJB and J2EE based platform.
- The research being carried out to study J2EE based design patterns at presentation Tier level.
- Results show that higher maintainability, reusability, software evolution and modularity among others are achievable using this techniques and framework.

## REFERENCES

Aoyama, M., 1998. New age of software development: How component-based software engineering changes the way of software development. International Workshop on Component-Based Software Engineering. Kyoto, Japan.

AspectJHome,http://aspectj.org/

Aspect-Oriented Software Development, http://aosd.net.

Chavez, C., 2004. A Model-Driven Approach to AspectOriented Design. PhD .Thesis, PUC-Rio.

Decker, K., K. Sycara and M. Williamson, 1996. Matchmaking and Brokering, International Conference on Multi-Agent Systems (ICMAS).

Hachani, O. and D. Bardou, 2002. Using aspect-oriented programming for design patterns implementation. Proceeding Workshop Reuse in Object-Oriented Information Systems Design.

Iglesias, C.A., M. Garijo, J.C. Gonzalez and J.R. Velasco, 1998. Analysis and Design of Multi-Agent Systems Using MAS-CommonKADS, M.P Singh, A.Rao, M.J. Wooldridge, (Ed.), Intelligent Agent IV(ATAL'97), LNAI 1365, Springer-Verlag, Berlin, Germany, pp: 314-327.

Janzen, D. and K.D. Volder, 2004. Programming With Crosscutting Effective Views. In Proc. 18th Eur. Conf. Object-Oriented Programming (ECOOP), pp: 195-218.

Kolp, M., P. Giorgini and J. Mylopoulos, 2001. A Goal-Based Organizational Perspective on Multi-Agents Architectures, (ATAL), Seattle, USA.

Krutisch, R., P. Meier and M. Wirsing, 2003. The Agent Component Approach, Combining Agents and Components.

Kulesza, U., A. Garcia and C. Lucena, 2004. Generating Aspect Oriented Agent Architectures. Proceedings of the 3rd Workshop on Early Aspects, 3rd International Conference on Aspect-Oriented Software Development, Lancaster, UK.

Mehta N.R., N. Medvidovic and S. Phadke, 2000. Towards a Taxonomy of Software Connectors. Twenty two International Conference on Software Engineering (ICSE), Limerick, Ireland.

Sun-Java 2 Platform, Enterprise Edition and JavaBeans Architecture, http://java.sun.com/j2ee/.

Sun Microsystems: Enterprise JavaBeans Specification., www.sun.com.

Trilnik, F., A. Diaz and Marcelo Campo, 2002. Smartweaver: An agent-based approach for aspect-oriented development. ICSE., pp: 716.