Volume 3, Issue 5, May 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# A Survey of Methodologies: Component, Aspect and Agent

**Kitty Ahuja***
*Asst. Professor,*
*ABES Engg. College, India.*

**Paritosh Ahuja**
*Project Manager,*
*IBM India*

***Abstract— Modern software systems become more and more large-scale, complex and uneasily controlled, resulting in high development cost, low productivity, unmanageable software quality and high risk to move to new technology. Consequently, there is a growing demand of searching for a new, efficient, and cost-effective software development paradigm. This paper compares Component-Based Software Development, Aspect-Oriented Software Development and Agent Based Software Development, three most advanced software development methods. These different approaches depend almost totally on their application domain but their usability can be equally applied across domains. The purpose of this comparative analysis is to give a succinct and clear review of these three methodologies. Their definitions, characteristics, advantages and disadvantages are considered and a conceptual mind-map is generated that sets out a foundation to assist in the formulation and design of a possible new integrated software development approach. This includes supportive techniques to benefit from the examined methods' potential advantages for cross-fertilization. It is a basis upon which new thinking may be initiated and further research stimulated in the software engineering subject field***

***Keywords— Component, Aspect, Crosscutting Concern, Code tangling and Agent***

## I. INTRODUCTION

Software systems need to advance continuously to meet latest software requirements. Initially, concern became a central concept to cope with software evolution by decomposing a software system into smaller and more comprehensible modules. Component Based Software Development structures a program by separating concerns into reusable components. Aspect Oriented Software Development modularizes crosscutting concerns into aspects, by identifying tangled and scattered code into systems. Agent-Oriented Software Development is being described as a new software development paradigm. Now a days, software agents and multi-agent systems have grown into what is now one of the most active areas of research and development activity in computing. One of the most important reasons for the interest is that the concept of an agent as an autonomous system, capable of interacting with other agents in order to satisfy its design objectives, is a natural one for software designers. Our aim in this paper is to survey and compares the state of the art in Component-Based Software Development, Aspect-Oriented Software Development and Agent Based Software Development.

## II. COMPONENT BASED SOFTWARE DEVELOPMENT

In order to address some of the modern business requirement, such as easily modifiable and reusable functionality, the concept of software component was devised. Component-based software development approach is based on the idea to develop software systems by selecting appropriate off-the-shelf components and then to assemble them with a well-defined software architecture. Component-based software development is an approach to software development that relies on software reuse. It can significantly reduce development cost and time-to-market and improve maintainability, reliability and overall quality of software systems [4].

### A. *Definition and Characteristics of Components*

A component is an independent software unit, creating a system in coordination with other components. It is a discrete piece of software that does some specific, predefined work. The concept was great, because it allows writing a component once and using it everywhere rather than re-inventing the wheel every time. It can be modified, corrected or enhanced, simply by updating or replacing the component. Components are reusable, interoperable and upgradable.
In general, a component has three main features:
1) A component is an independent and replaceable part of a system that fulfils a clear function;
2) A component works in the context of a well defined architecture; and
3) A component communicates with other components by its interfaces [1].
However, some specialists define components from totally different perspectives. Council and Heineman[2] focus on and believe that "a component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard". On the other hand, Szyperski [3] focuses on the: "a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties". So, the priority of CBSE is the regarding the wide range of functionality available throughout a software system.

 

B. *CBSD Terminology*

*Component Code*

This is the actual work component does. Once you set up the component infrastructure, your component need to do something. For example if your component is intended to perform work related to invoices, then you need to create a component to calculate totals, do inventory look up and so on.

*Interface*

An interface allows program to access the functionality of your component. It is a collection of public function definition that your component makes available to your program and other components. It tells rest of the world what your component can do and how to make use of its functionality.

*C. Component Architecture and Interactions*

To ensure that a component-based software system can run properly and effectively, the system architecture is the most important factor. Components architectures are defined by their "requires" (inputs) and "provides" (outputs) interfaces that are related by the function as illustrated in Figure 1. These components are capable of providing a service to other client components, while they themselves could be clients.

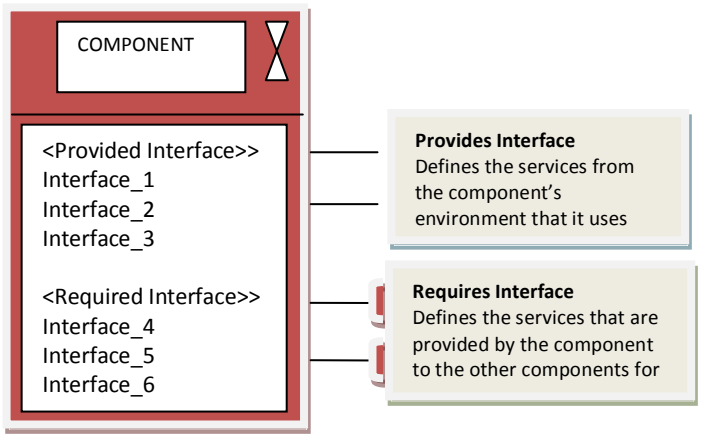


Figure 1 Component Architecture with Inputs and Outputs

The system architecture of component-based software systems should be a layered and modular architecture as in Figure 2. CBSD emphasizes modular architecture which helps to partially develop a system and incrementally enhance the functions by adding and/or replacing components. It enables efficient and error-free software development.

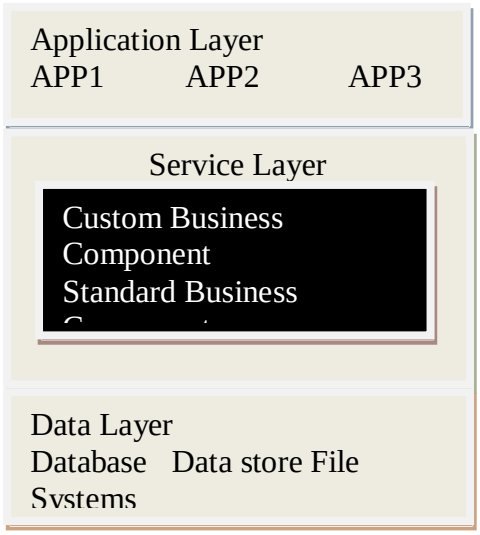


Figure 2 Layered and Modular Architecture of Component Based Systems

First layer is the application layer supporting a business. The second layer consists of components engaged in a specific business or application domain. It contains the business logic as well as business rules, unifying processes that span multiple applications, products and lines of business. Finally, the lowest layer of system software components includes data layer providing a uniform method to access information and perform transactions.

 

This separation of business processes and rules from the underlying application logic delivers the ability to change processes with minimal impact to systems infrastructures, helping organizations to increase agility and responsiveness to business needs. In addition, it provides a framework for extensive customization based on user needs.

*D. Component Based Technologies*

The idea is to build reusable, manageable and exchangeable software units through clearly defined interfaces. Different manufactures offer platforms like OMG's CORBA, Microsoft's Component Object Model (COM) and Distributed COM (DCOM), .NET Remoting and Sun Microsystems's JavaBeans and Enterprise JavaBeans.

## III. ASPECT ORIENTED SOFTWARE DEVELOPMENT

Aspect-oriented software development (AOSD) is a software development technology that provides improved separation of concerns. The implementation of software applications using aspect oriented software development techniques results in a better implementation structure which has an impact on many important software qualities such as enhanced reusability and reduced complexity. In turn, these software qualities lead to an improved software development lifecycle and, hence, to better software [13]. AOSD allows concerns to be addressed separately and automatically unified into working systems.

*A. Definition of Aspect*

A programming methodology is called Aspect-Oriented provides a solution to separate concerns that would otherwise be cross-cutting. The goal of aspect oriented software development is to provide a modularization technique to separate the core functionality of a software system from system-wide concerns that cut across the implementation of this core functionality. AOSD introduces a special abstractions mechanism known as aspects to separate crosscutting concerns throughout the software life cycle. An aspect is a special kind of module that represents a crosscutting concern.

*B. Aspect Oriented Terminology*

*Separation of Concerns*

The principle of separation of concerns states that each concern that is relevant to the software application is best treated separately from the other concerns. A concern of an application can be related to its functionality or domain and developers need to identify and implement these different concerns. At the software implementation level, this means that each concern is implemented in its own module. Separation of concerns thereby reduces the complexity of each individual module concerns promoting good design and reusable implementations. In other words, Concerns that are contained within a single module are relatively easy to reuse, while concerns that are contained in modules together with other concerns are not.

*Cross-cutting Concerns*

In object-orientation, separation of concerns is achieved by decomposing an application into individual objects. Object-orientation provides significant support to achieve this through encapsulation, polymorphism, inheritance and delegation. However, these may not be sufficient for separating some special concerns found in most complex systems. These concerns are called cross-cutting concerns since they cross cut the other concern. The concerns can be such as security, auditing, logging, consistency, error handling, user interface and so on. Two main problems caused by crosscutting concerns are, code tangling and scattering. These are two difficulties that occur in this methodology which affect software design and development and results in lower productivity and less code reuse.

*Code Scattering*

This situation occurs when a concern exists in many places or scattered throughout the entire application. It is particularly a state where one concern exists in many modules and making it difficult to evolve.

*Code Tangling*

The code tangling breaks the principle of separation of concerns because a single module contains the implementation of more than one concern. [2] It is particularly a state where many concerns exist in one module. This complicates the code and reduces the reusability of individual concerns.

*Without proper means for separation and modularization, crosscutting concerns tend to be scattered and tangled up with other concerns.*

*Join Points*

A join point is a specification of when the aspect code should be invoked in the main program. Aspects can only be invoked at some well defined and principled points in the program's execution. These points are referred to as join points. In standard AOSD terminology, the term join point is not limited to program execution but is extended to the entire software development lifecycle [13].

*C. Aspect Oriented System Architecture*

Aspect-oriented architectures support the division of software into modules that represent the concerns-of-interest to the system. This includes concerns that are feature-based or, in particular, concerns that have an impact on (or "crosscut") many different parts of the system.

 

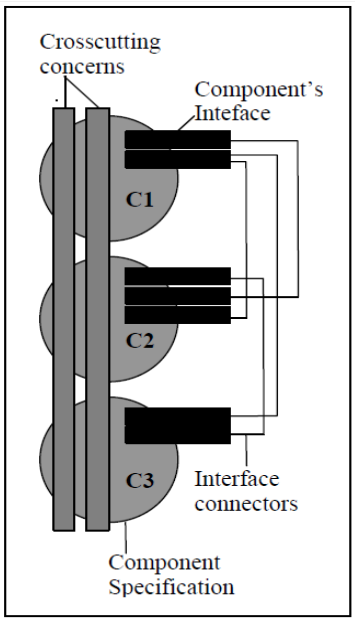


Figure 3a
Current Specification of Software Architecture

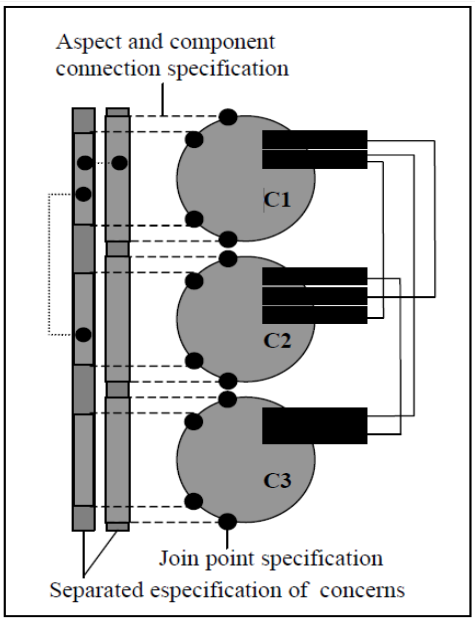


Figure 3b
Separated Specification of Crosscutting Concerns at the Software Architecture Level

Figures 3a and 3b specify the software architecture of the same system. Figure 3a shows how software architectures are currently specified. Figure 3b specifies the software architecture can be used for AOSD. The specification of aspects has been separated and the join points are specified in the components [23].

*D. Languages and Framework*

Aspect oriented programming (AOP) can be understood as the desire to make quantified statements about the behaviour of programs, and to have these quantifications hold over programs written by programmers [11]. AOP makes statements of the form: In program P, whenever condition C arises, perform action A over a coded program P [20]. Several aspect-oriented programming languages and frameworks are available for use in development, which includes

■ AspectJ (www.eclipse.org/aspectj) is an aspect-oriented extension to Java [19], which is an aspect-oriented extension to Java.

■ JBossAOP (jboss.com/products/aop) provides an aspect framework for Java that includes prepackaged aspects for caching, asynchronous communication, transactions, and security.

■ AspectC++ (www.aspectc.org) offers aspect-oriented support for C++.

■ Aspect# (www.castleproject.org/index.php/AspectSharp) is an aspect-oriented framework for the Microsoft Common Language Infrastructure [20].

## IV. AGENT BASED SOFTWARE DEVELOPMENT

As the need to integrate several heterogeneous environments has increased, this introduces new levels of complexity of computing systems. This complexity appears to be approaching the limits of human capability which makes some issues to be dealt with at runtime. As the systems become massive, there is no way to make timely decisions to rapidly changing and conflicting demands. The only option left is agent based software systems – systems that can manage themselves given high level objectives [29].

Agent based software development is rapidly emerging technology for the development of complex software systems, combining contributions from many different research areas, including artificial intelligence, software engineering, robotics and distributed computing [30]. This approach enhances our ability to model, design and build complex distributed systems [26]. The applications developed using this paradigm exhibit proactive and intelligent behavior [30].

A. *Definition and Characteristics of Agent*

In agent based software development the key abstraction that is used is "agent". Now what an agent is. An agent is characterized by unique identity, proactivity, persistence, autonomy, sociability [30]. An agent is an autonomous software entity which interacts with its environment and with other agents proactively to achieve its goals. An agent inherits its unique identity simply by being an object [27]. Some of the examples of software agents are: computer viruses, artificial players in computer games, web spiders which collect data to build indexes to be used by a search engine, i.e. Google. [25]. In most cases, single agent is not sufficient; multiple agents are the norm to represent:

- Natural decentralization

 

- Multiple loci of control
- Multiple perspectives
- Competing interests [26]

B. *Agent Based Software Development Terminology Proactive*

To be proactive, agents should not simply act in response to their environment; they should be able to exhibit opportunistic, goal-directed behavior and take the initiative where ever appropriate [31].

*Persistence*

Persistence makes it worthwhile for agents to learn about and model each other.

*Autonomy*

Agent autonomy enables an agent to choose its own actions.

*Sociability*

Enabling an agent to converse with other agents achieves sociability. The conversations, normally conducted by sending and receiving messages, provide opportunities for agents to coordinate their activities and cooperate. The Figure 4 shows how an agent interacts with its environment [27].

C. *Agent Architecture and Interactions*

To ensure that an agent-based software system can run properly and effectively, the system architecture and its interactions are the most important aspect.

Agent based systems architecture is defined by "inputs" an agent senses from the environment, action is taken by effectors based on the "condition and action" rules and provides "outputs" to the environment. These interactions are depicted in Figure 4.

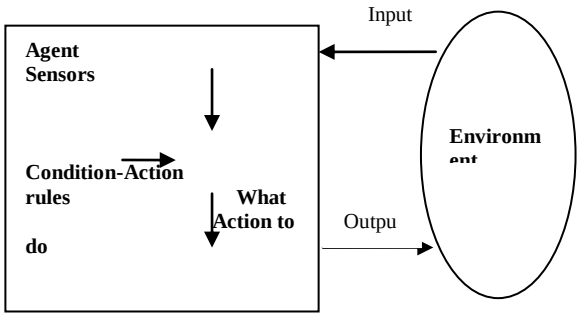


Figure 4 Interactions in Agent Environment

D. *Languages and Framework*

Agent-oriented programming is a programming paradigm where the construction of the software is centered on software agents. There are multiple 'frameworks' that implement this programming paradigm like Jason, Jadex and Jade. JADE (Java Agent Development Framework) for the Java-platform one of the frameworks which is mainly used for agent oriented programming. JADE is a middleware that facilitates the development of multi-agent systems. It includes

- A runtime environment where JADE agents can "live" and that must be active on a given host before one or more agents can be executed on that host.
- A library of classes that programmers have to/can use (directly or by specializing them) to develop their agents.
- A suite of graphical tools that allows administrating and monitoring the activity of running agents [32].

## V. CONCLUSION

This paper has sought to give a survey of three software development paradigms, Component Based Software Development, Aspect Oriented Software Developer and Agent Based Software Development. Component-based software development builds new software systems by integrating existing components. Identifying and properly managing the interdependencies among components are the main concern in this paradigm. Aspect oriented software development provides a modularization technique to separate the core functionality of a software system from system-wide concerns that cut across the implementation of the core functionality. Agent based software development promises to increase the flexibility and power of software systems to provide intelligent web services and intelligent systems which are "self-documenting", "self-maintaining" and thus "self-managing" of the entire software life-cycle.

With the advent of agent based software development, "Autonomic Computing Systems" are going to become more real and visible in present-day computing world, thus creating a context-aware-ubiquitous computing environment. Autonomic Systems have the capability of configuring, healing, optimizing and protecting themselves. These Self-managed systems are going to become a part of large enterprises, exhibiting autonomic behavior and tendency for self-adaptability. Studying these "Autonomic Systems" and "Autonomic Computing" paradigm is an exciting research challenge for future R & D work.

 

**REFERENCES**


[1] A.W.Brown, K.C. Wallnau, "The Current State of CBSE," IEEE Software , Volume: 15 5, Sept.-Oct. 1998, pp. 37 – 46.

[2] Councill B. Heineman, G. T. Definition of a Software Component and Its Elements Component-Based Software Engineering: Putting the Pieces Together, ed Boston:Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 5-19.

[3] Szyperski, C. Component Software: Beyond Object-Oriented Programming: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] G.Pour, M. Griss, J. Favaro, "Making the Transition to Component-Based Enterprise Software Development: Overcoming the Obstacles – Patterns for Success," Proceedings of Technology of Object-Oriented Languages and systems, 1999, pp.419 – 419.

[5] Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes Xia Cai, Michael R. Lyu, Kam-Fai Wong Roy Ko The Chinese University of Hong Kong Hong Kong Productivity Council

[6] New Age of Software Development:How Component-Based Software Engineering Changes the Way of Software Development ? Mikio Aoyama,Department of Information and Electronics Engineering,Niigata Institute of Technology

[7] The artifacts of component-based development, M. Rizwan Jameel Qureshi, Shaukat Ali Hayat, Dept. of Computer Science, COMSATS Institute of Information Technology

[8] M. L. Griss, "Software Reuse Architecture, Process, and Organization for Business Success," Proceedings of the Eighth Israeli Conference on Computer Systems and Software Engineering, 1997, pp. 86-98.

[9] P.Herzum, O.Slims, "Business Component Factory – A Comprehensive Overview of Component-Based Development for the Enterprise," OMG Press, 2000.

[10] HongKong Productivity Council, http://www.hkpc.org/itd/servic11.htm, April, 2000.

[11] IBM:http://www4.ibm.com/software/ad/sanfranci

[12] Proceedings of the Third International ERCIM Symposium on Software Evolution (Software Evolution 2007) Evolutionary Problems in Aspect-Oriented Software Development1 Kim Mens and Tom Tourw´e

[13] Aspect-Oriented Software Development (AOSD)- An Introduction, Johan Brichau and Theo D'Hondt.

[14] Aspect-Oriented Software Development in Practice: Tales from AOSD, A. Rashid, T. Cottenier, P. Greenwood, R. Chitchyan, R. Meunier, R. Coelho, M. Sudholt, W. Joosen, Lancaster Univ., Lancaster, UK 19FEBRUARY 2010 Computing Practices Published by the IEEE Computer Society © 2010.

[15] Aspect-Oriented and Component-Based Software Engineering Dr Awais Rashid IEE EProc.-Softw, Vol. 148. NO. 3, June 2001.

[16] A comparative study of agile, component-based, aspect-oriented and mashup software development methods, Ahmed Patel, Ali Seyfi, Mona Taghavi, Christopher Wills, Liu Na, Rodziah Latih, Sanjay Misra

[17] "AOSD Ontology 1.0: Public Ontology of Aspect Orientation", Report of the EU Networkof Excellence on AOSD by Klaas van de Berg, Jose-Maria Conejero, Ruzanna Chitchyan (editors), 2005.

[18] A component-based and aspect-oriented model for software evolution by Nicolas Pessemier, Lionel Seinturier* and Laurence Duchien "International Journal of Computer Applications in Technology 31, 1-2 (2008) 94-105"

[19] AspectJ, Aspect-oriented Programming in Java, http://www.aspectj.org.

[20] Aspect-Oriented Programming: Gail Murphy, University of British Columbia Christa Schwanninger, Siemens IEEE 2006.

[21] Sommerville, I. Software Engineering, 8 ed, 2007.

[22] Aspect-oriented software development versus other development methods : vahdat abdelzad, fereidoon shams aliee: Journal of Theoretical and Applied Information Technology September 2011. Vol. 31 No.2

[23] Aspect Oriented Software Architecture: a Structural Perspective: A. Navasa, M.A. Pérez, J.M. Murillo, J. Hernández

[24] Using aspect oriented software architecture for enterprise systems development: IEEE: Digital Information Management (ICDIM), 2010 Fifth International Conference on July 2010, Verma, Pawan Kumar Dept. of Computer Sc.

[25] A survey of Agent-Oriented Software Engineering, Amund Tveit, www.csgsc.org, May 2001.

[26] Agent oriented software Engineering, Nick Jennings, http://www.ecs.soton.ac.uk/~nrjk/

[27] Inside an Agent José M.Vidal , Paul A. Buhler, Michael N. Huhns , February 2001, IEEE Internet computing

[28] Agent Based Software Development Michael Luck, Ronald Ashri and Mark d'Inverno Chapter 4: Methodologies and Modeling Languages

[29] The vision of Autonomic Computing , Jeffrey O.Kephart David M.Chess IBM Thomas J. Watson Research Center , IEEE January 2003

[30] Agent Oriented Software Engineering for Internet Applications: Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, Michael Wooldridge, Coordination of Internet Agents, Springer 2000.

[31] Agent-Oriented Software Engineering: The State of the Art, MichaelWooldridge and Paolo Ciancarini

[32] Agent Oriented Programming Fabiano Dalpiaz :Agent-Oriented Software Engineering (AOSE) : 2010/2011


 