Computers in Human Behavior xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Aspect oriented design for team learning management system

Ghada Al-Hudhud

College of Computer and Information Sciences, Department of Information Technology, King Saud University, Riyadh, Saudi Arabia

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Multi agent system (MAS)
Aspect-oriented software development (AOSD)
Team learning activities
Learning styles for group learning activities
Learning management systems

ABSTRACT

A multi agent system (MAS) is a complex system composed of heterogeneous agents each has a number of concern that are cross-cutting such as mobility, learning, collaboration, adaptation, interaction and autonomy. MASs are currently designed to be superimposed on object oriented designs so that it can be possible to separate these concerns in order to improve reusability and maintainability. Hence, aspect-oriented software development (AOSD) exists to cope with complexity of software development for the purpose of separating functionality that are not handled by other software development. Following this line of thought, AOSD is considered for developing aspects for team learning management system; that allows recognizing learner's learning preferences and associated learning style in the learning environment. This paper presents an approach to move from object oriented eTutor to agent oriented eTutor through aspect oriented software development. This transition is being deployed through the implementation level.

1. Introduction

Object oriented (OO) software engineering provides standard modularity mechanisms, but they are unable to modularize all concerns of complex systems (Reis, Reis, Schlebbe, & Nunes, 2002). Agent oriented approaches provide unique and advanced program structuring and modularization techniques that explicitly capture the crosscutting system structure. Agent oriented software engineering enhance improving the modularity of software by focus on separation of concerns SOC into aspects and structuring system during system development (O'Riordan, 2001). Agent oriented software engineering approach is used to develop high complex system with highly modularity of all concerns. Currently existing OO approaches do not support early stage handling of the SOC (Garcia, Kulesza, Chavez, & De Lucena, 2006; Garcia, Kulesza, & Lucena, 2005). Moreover, the current OO software development approaches handle generally a limited number of agent types which are encapsulating the agent behavior and data an object. This results in lack of support for dealing with the interactive and overlapping nature of concerns.

Another approach is current well known; multi agent systems (MAS). MAS applications is composed of multiple types of agents, each of them having different concerns, properties and roles that might overlap and interact with each other accordingly, this requires scrutinizing a structured way for composition to be considered during an early stage of design that supports and handle the separation of concerns in terms of each agent functionalities and role to support maintainability and reusability. Software engineering of MASs involves a number of concerns (Brichau, Chitchyan, Rashid, & D'Hondt, 2008; Cunha, Sobral, & Monteiro, 2006). There is some challenges appear in modeling, designing, and development of these concerns (aspects). This is because they are inherently crosscutting as the system complexity increases. Unfortunately, the existing modeling languages, design and implementation approaches are not able to provide explicit support for the separation of crosscutting MAS-related concerns.

For the purpose of overcoming the above mentioned problems, aspect oriented software development (AOSD) is considered. AOSD supports the developer in cleanly separating components (functionality) and aspects (concerns) from each other, by providing mechanisms that make it possible to abstract and compose them in complex MAS (Reis et al., 2002). ASOD has been proposed as a technique for improving SOC in software construction and support improved reusability and ease of evolution ate the implementation stage (Cunha et al., 2006). The aspect oriented techniques deal with crosscutting concerns that separate and compose the system (Garcia, Kulesza, Lucena, 2005).

This paper investigates how to integrate agents neatly involving the redesign and refactoring of an existing object oriented learning management system while maintain reusability. The following sections in the paper present AOSD which allows supporting SOC of MAS. The goals of our proposal is to achieve an improved adaptability in a learning management system to support the SOC and accordingly to improve reusability.

E-mail address: galhudhud@ksu.edu.sa

http://dx.doi.org/10.1016/j.chb.2015.01.032
0747-5632/

In the background section, a description and review for objet oriented paradigm relating it to other software development paradigms MAS, AOSD. In Section 3, description of our case study.

2. Background

2.1. Object-oriented software development

The term of OO is to model a system using a classes that have a set of attribute to define and set of operations to perform and these classes after run time create an object to store data. To model the system, OO use UML and RUP to modularize these classes and perform function. UML is a unified modeling language and RUP is Rational Unified Process used to model OO Analysis and design. We can improve the reusability of OO by using design pattern or AOSD (Sommerville, 2010). But today most systems are tend to be complex so this is a main problem of using OO so we use MAS instead to deal with complexity and apply AOSD to enhance reusability and maintainability (Sommerville, 2010).

2.2. Agent-oriented software engineering AOSE

AOSE is a complementary for OOSD (Garcia, Kulesza, Lucena, 2005). AOSE is defined as an approach in software development that have methodologies and modeling method for integrating heterogeneous agent into software system (Juneidi, 2004). To solve complex problems, these agents must work cooperatively with other agents in a heterogeneous environment (Juneidi, 2004). This is the area of multi agent systems (MAS). MAS is an organization of autonomous agents that have particular objective to achieve and interact with each other (Ahuja & Ahuja, 2013).

2.3. Aspect-oriented software development (AOSD)

AOSD is an evolving software design paradigm to modularize concerns by providing support to separating the functionality for its cross-cutting concerns. Existing approaches, including Structured Programming, Object-Oriented Programming (OOP) and Agent-Oriented programming (AOP) not handle these concerns well. So AOSD complements these approaches, but does not replace them (Garcia, Kulesza, Lucena, 2005; Sabatucci, Garcia, Cacho, Cossentino, & Gaglio, 2008). The main goal of AOSD is to enhance modularization of crosscutting concerns in a system, to manage the relationship between represented concerns. The concerns can be intermixed throughout the design and implementation; common examples include error handling, logging, and security (Brichau et al., 2008). Concerns can relate to functional or nonfunctional requirements. The problem of crosscutting concerns that are make systems difficult to maintain, increase the complexity of the system and reduce the reusability (Kulesza et al., 2006). By applying AOSD techniques, these concerns can be put into separate modules called aspects, untangling them from each other and make program easier to maintain and reuse (O'Riordan, 2001; Sabatucci et al., 2008). Aspect is an abstraction that modularizes these concerns (Garcia, Kulesza, Lucena, 2005).

2.4. Object, agent and aspect comparison

A comparison between object, agent and aspect is presented in Table 1. Nowadays aspect-oriented software development is becoming very important because it is promoting to improve separation of concerns, so they make software systems easier to maintain and reuse. It is important to verify that emerging development paradigms support the improvement of MASs modularization of the crosscutting concerns relative. Also, It is important to understand the relationships between aspect and agent-oriented.

AOSD is an evolution of current software development technologies. Its purpose is complementing current paradigms in order to improve the separation of crosscutting concerns in software development, providing better understandability, maintainability and reusability to the artifacts generated during software development lifecycle. The aspect oriented paradigm is recent and, therefore, there is no consensus about the concepts and practices it encloses.

Fig. 1 shows that the classical concerns that agent systems software engineers tackle in OO software engineering are different from concerns that related to agents, such as: agent autonomy, interaction, adaptation, collaboration, learning, and mobility (see Table 2 and 3).

2.5. Separation of concern

A concern can be thought of as a part of software that represents a single concept (Garcia, Kulesza, Lucena, 2005). Encapsulating and manipulating those parts of software that are relevant to a particular concern is commonly known as separation of concerns (SOC) (Brichau, Chitchyan, Rashid, & D'Hondt, 2008; Garcia et al., 2003). Separating these concerns from components requires a mechanism for composing them later. The process of composing is called a join points. Join points are well-defined points in the dynamic execution of the program. Examples of join points are method calls and method executions (Reis, Reis, Schlebbe, & Nunes).

Table 1
Comparison between Object, agent, and aspect.

	Definition	Component	Properties
Object	1. Abstraction 2. Represents an entity 3. Has an identity Encapsulates a state and behavior	1. Attribute 2. Operation	1. Interaction 2. Learning
Agent	1. Encapsulated computer system 2. Situated in some environment, 3. Capable of flexible, autonomous action in that environment 4. meet its design objectives, (Garcia, Kuleska, Sant'Anna, Chavez, & Lucena, 2005)	1. Goals 2. Believes 3. Plans 4. Capabilities	1. Interaction 2. Adaptation 3. Autonomy 4. Learning 5. Mobility 6. Collaboration
Aspect	1. Abstraction 2. Separate the concern Deals with cross-cutting (require at several component) and Encapsulated into modules (Garcia, Kuleska, Sant'Anna, Chavez, & Lucena, 2005)	1. Joint point 2. Advice-the code to implement the cross-cutting concern. 3. Poitcut- a statement that defines where the aspect will be woven into the program	1. Interaction 2. Adaptation 3. Autonomy 4. Learning 5. Mobility 6. Collaboration

Please cite this article in press as: Al-Hudhud, G. Aspect oriented design for team learning management system. *Computers in Human Behavior* (2015), http://dx.doi.org/10.1016/j.chb.2015.01.032