Comparing Stability of Implementation Techniques for Multi-Agent System Product Lines

Camila Nunes[1], Uirá Kulesza[2], Cláudio Sant'Anna[3],
Ingrid Nunes[1], Alessandro Garcia[4], Carlos Lucena[1]

[1]*Computer Science Department, Pontifical Catholic University of Rio de Janeiro, Brazil*
[2]*Federal University of Rio Grande do Norte – UFRN, Brazil,*
[3]*Federal University of Bahia – UFBA, Brazil*
[4]*Computing Department, Lancaster University, United Kingdom*
{cnunes,ioliveira,lucena}@inf.puc-rio.br, uira@dimap.ufrn.br
santanna@dcc.ufba.br, garciaa@comp.lancs.ac.uk

Abstract

Multi-agent systems (MAS) are increasingly being exploited to support autonomous recommendation of products and information to contemporary application users. Multi-agent system product lines (MAS-PL) promote large-scale reuse of common and variable agency features across multiple MAS applications. The development of MAS-PLs can be achieved through alternative MAS-specific frameworks (JADE and Jadex), and general-purpose implementation techniques, such as aspect-oriented programming (AOP). However, there is not much evidence on how these techniques provide better modularity, allowing the conception of stable MAS-PL designs. This paper reports an empirical study that assesses the modularity of a MAS-PL through a systematic analysis of its releases. The study consists of a comparison among three distinct versions of this MAS-PL, each one implemented with a different technique: (i) Jadex platform and configuration files; (ii) JADE platform and configuration files; and (iii) JADE platform enriched with AOP mechanisms. Our analysis was driven by fundamental modularity attributes.

1. Introduction

Software agents bring unique advantages to software development as it supports the (semi-) automation of tasks, such as autonomous recommendation of products and information to users [9]. With the agent technology being applied to a wide range of application domains, the design of stable and modular software product-line (SPL) architectures [1] becomes a key challenge to multi-agent system engineers. Multi-agent system product lines (MAS-PLs) have been recently proposed [3] as a systematic way to address a set of common and variable agency features in a family of MAS products. However, the stable and modular design of such variabilities in a MAS-PL is particularly intriguing as they tend to exert a global crosscutting effect over each other and the agent architecture decompositions [2].

There is a growing number of MAS-specific and general-purpose technologies emerging as candidates to support stable MAS-PL designs. For instance, different MAS platforms, such as JADE [13] and Jadex [12], offer classical OO variability mechanisms, such as inheritance and polymorphism. These mechanisms can be complemented with the use of configuration files and dependency injection from Spring Framework [14] and aspect-oriented programming (AOP) mechanisms [4, 5]. However, there is no work that assesses the (dis)advantages and complementarities of these different implementation strategies for improving MAS-PL design longevity.

In this context, this paper briefly presents a quantitative assessment of alternative techniques for implementing and evolving a typical Web-based MAS-PL, called OLIS (*OnLine Intelligent Services*). Our evaluation was based on conventional metrics suites for modularity analysis [6]. Our investigation compared OLIS MAS-PL implementations based on the use of: (i) JADE platform and Spring configuration files; (ii) Jadex platform and Spring configuration files; and (iii) JADE and AOP with the AspectJ language [7].

This paper is organized as follows. Section 2 presents a brief description about the OLIS MAS-PL and the study results. Section 3 presents a short discussion. Finally, Section 4 presents final remarks.

2. Study Results

The OLIS (OnLine Intelligent Services) is a MAS-PL for web applications that provide several personal services to the users. The core of the OLIS architecture is composed mainly by two services: (i) Events Announcement; and (ii) Calendar Services. The Events Announcement service allows users to announce events to other system users through an events board. The Calendar service lets users to schedule events in their calendar. The OLIS was designed in such way that the system can be evolved to incorporate new services without interfering on the existing ones. The system has different flavors according to the type of event that it manages: generic events, academic events and travel events. The OLIS MAS-PL was structured according to the Layer architectural pattern [16]. The layers that compose the architecture are: GUI, Business, and Data [15]. During the development and evolution of the MAS-PL, we applied a series of change scenarios, adding mandatory, optional and alternative features in the MAS-PL architecture. During the evolution of the OLIS MAS-PL, eight releases [15] were generated.

This section presents results of the measurement process based on the set of metrics defined in [6]. The interest was to observe the stability of such modularity attributes for each MAS-PL alternative implementation. Due to space limitations, we do not present all the data and graphics of the absolute values for each of the metrics.

2.1. Separation of Agency Features

We analyzed three features of the OLIS MAS-PL using the concern metrics. This analysis encompassed one mandatory, one alternative and one optional feature. We considered the absolute values for their corresponding concern metrics.

Figure 1 illustrates the results for the feature that imports events, a mandatory feature added in the release 2 (R2). The results show that the AO solution presents lower values and superior stability in terms of tangling (CDLOC – Concern Diffusion over Lines of Code) and scattering (CDC – Concern Diffusion over Components). As it can be observed, the AO modularization is more stable. The results show that both implementations (Jadex and JADE) presented the same values because agents were not included in this release yet. This indicates that the AO implementation was more effective to modularize this feature according to the software history. This feature in the OO implementations presents superior values of tangling with other features (CDLOC).

Figure 2 shows the results for the Academic Event alternative feature included in R3. The values were the same for all the implementations in terms of CDC and CDO metrics. This occurred because the use OO solutions in combination with the configuration files technique required the creation of classes that extend existing classes in the MAS-PL core. This is important to allow the configuration of the product and a superior feature modularization. For example, in order to allow the management of academic data, the `EventDataManager` class had to be extended by the `AcademicEventDataManager` class in JADE and Jadex solutions [15]. In the AO implementation [15], we created an aspect that affects the `EventDataManager` class. Thus, for all implementations, it was necessary to create an additional component. As a consequence, the implementation of this alternative feature presented similar results regarding the separation of concerns metrics.

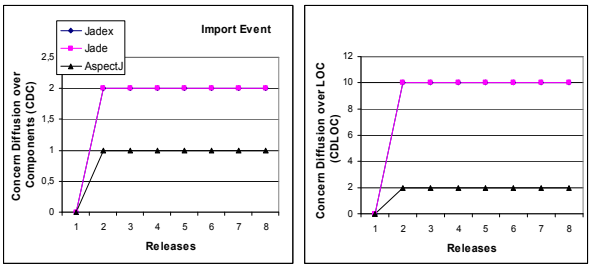

Figure 1. Concern metrics for import events mandatory feature.

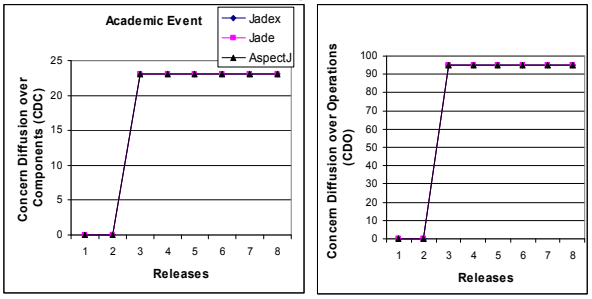

Figure 2. Concern metric for Academic events alternative feature.

Figure 3 shows the results for the Event Suggestion optional feature included in R4. The results show that the Jadex implementation exhibited more components (CDC) than other implementations. From this release, the entire agent infrastructure is included. Consequently, many components (agent classes and/or respective associated Jadex classes and configuration files) were added to allow agents communicate among them to implement agency features. The Jadex implementation required more components because it introduced many new plans (Java classes) and XML

files to manage these plans in order to provide interaction among these plans. On the other hand, these extra components did not translate in a high number of operations (CDO – Concern Diffusion over Operations). As mentioned, the interaction among the plans is specified in XML files. The number of operations (CDO) was the same for both JADE and AO implementations. The AO solution demanded fewer components to implement this feature. However, the CDO was the same of the JADE implementation. Besides new class methods, aspects include methods and advices to intercept existing classes and implement this feature.

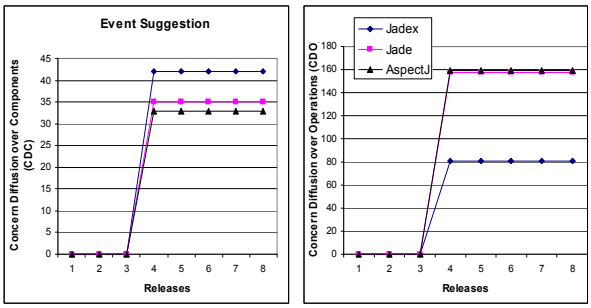

Figure 3. Concern metric for Event Suggestion optional feature.

2.2. Coupling

Figure 4 illustrates the values collected to the coupling metric of the OLIS MAS-PL. The graphic shows that the CBC (Coupling between Components) metric in the AO implementation was better through the releases as most aspects affect homogeneously fewer components; they add code with the goal to improve the modularization of the feature. One example of coupling increase in the JADE implementation is the OO Role pattern implementation [10]. While in the OO JADE implementation, each role class accesses methods of several classes, in the AO implementation each role was modularized as aspects, thereby contributing to reduce coupling metrics.

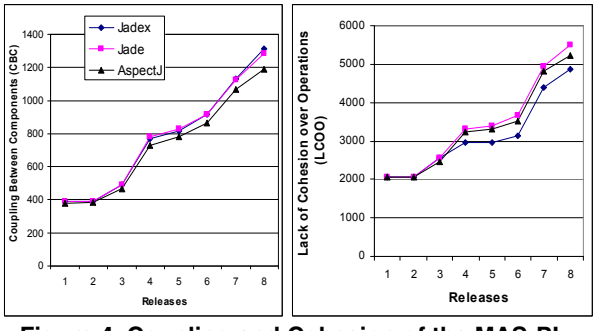

Figure 4. Coupling and Cohesion of the MAS-PL.

Another metric illustrated in Figure 4 is the LCOO (Lack of Cohesion over Operations) metric. It shows that the Jadex solution is comprised of highly cohesive classes throughout the releases. This happens because in the Jadex implementation, the interaction among plans and attributes declaration are defined in XML files. Additionally, the Jade implementation also indicates lack of cohesion. One reason that contributed for this cohesion impairment was the implementation of the Observer pattern [8], which contributes to reduce the cohesion of each class that participates of the pattern implementation.

2.3. Concern Interaction

This subsection presents the results for the User *Agents* concern. The *User Agents* concern includes the following roles: Suggestion, Reminder, Scheduler, Client and Participant. This metric aims at quantifying the interaction between concerns, which are agent roles in this case. Figure 5 shows the results of the metric quantifying the degree of interlacing between concerns in the SPL components (CIBC) [5]. The `UserAgentCore` component in the Jade implementation is responsible for initializing agent roles. This entails the so-called concern interlacing: roles are implemented by different set of methods, attributes and statements in the classes shared by the agent roles. In this case, there is no common element implementing both (i.e. concern overlap). In the JADE platform, role concepts are not present and, consequently, roles are specified by a set of classes: `UserAgentCore`, `UserAgent`, and `UserAgentRole` [15].

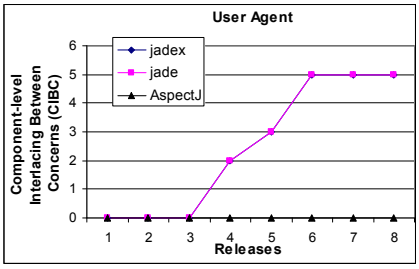

Figure 5. CIBC metric.

Specification of roles in Jadex is made via XML files based on the capability concept, and they are also declared in the main XML file called User.agent.xml. Concern interlaces also manifest in Jadex implementations, but through intertwined tags at the XML files. Despite of the different implementations, it was observed the same interlacing degree in both implementations (JADE and Jadex). On the other hand, all roles are completely modularized in different aspects in the AO releases. Therefore, the

`UserAgentCore` component is not interlaced with role semantics. In this case, the AO solution was more effective to allow the segregation of agency concerns.

3. Discussions

This section discusses some benefits and drawbacks of using different platforms, such as JADE and Jadex with different implementation techniques to implement agency features in MAS-PLs.

Roles Implementation. In general, the inclusion of new roles in the OLIS MAS-PL releases presents better results in the Jadex implementation in terms of CDO, CDC, and a number of other attributes, considering the absolute values. The roles implementation in the Jadex platform has shown that a good OO framework structure can provide key abstractions to developers, which also offers great results in terms of MAS-PL feature modularization. The OO framework structure can also be enriched to provide a complete separation of roles using the AOP abstractions [11], as demonstrated in our AO version of the OLIS MAS-PL. It contributes to avoid the drawbacks related to the feature interlacing discussed previously (Section 2.2).

Modularization of Alternative Features. The inclusion of alternative features in the OLIS MAS-PL occurred in R3 and R8. In R3, as showed in Section 2.1, the AO solution presented better results. However, the inclusion of the feature for supporting travel events presented different values for the implementations. Considering the concern metrics, the JADE was better for the CDC metric; the Jadex was better for the CDO and number of attributes metric; the AO was better for the CDLOC metrics. This result emphasizes that there is no clear advantage to use a specific solution to implement agency alternative features. However, for the alternative features considered in our study, we believe the Jadex and JADE OO implementations are more adequate solutions. Additionally, the use of inheritance and polymorphism in the implementation of the alternative events types represents a natural way to modularize this kind of feature, which is already well understood and widely adopted by the development community.

4. Final Remarks

In this paper, we presented a quantitative study of the incremental development of a multi-agent system product line (MAS-PL). In our study, we compared three different versions of the OLIS MAS-PL implemented using traditional OO technologies, such OO frameworks and configurations files with dependency injection, and aspect-oriented alternative implementations. We developed a traditional web-based system, named OLIS, and after that we evolved it to incorporate a series of change scenarios (agency features). We concluded that the three implementations did not present relevant differences regarding the degree of modularity and maintainability along the different releases.

5. References

[1] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, USA, 2002.

[2] Garcia, A., Lucena, C. Taming Heterogeneous Agent Architectures with Aspects. Communications of the ACM, May 2008.

[3] Pena, J., et al. Managing the Evolution of an Enterprise Architecture Using a MAS-Product-Line Approach. *Software Engineering Research and Practice*, pp. 995–1001. CSREA Press, 2006.

[4] Alves, V., et al. Extracting and evolving mobile games product lines. In Proceedings of the 9th SPLC 2005, 70–81.

[5] Figueiredo, E., et al. Evolving software product lines with aspects: An empirical study on design stability. In *Proceedings of the 30th ICSE'08*, pp. 261-270.

[6] Sant'Anna, C., et al. (2003). On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In *Proceedings XVII of SBES'03*, pp. 19–34.

[7] The AspectJ Project. http://eclipse.org/aspectj/.

[8] Gamma, E., et al.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley (1995)

[9] Holz, H., et al. Personalization Techniques and Recommender Systems, V(70). World Scientific Publishing.

[10] Bumer, D.,et al: The Role Object Pattern. http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/riehle.pdf (PLoP) 97.

[11] Kendall. E.: Role Model Designs and Implementations with Aspect-oriented Programming. OOPSLA 1999: 353-369.

[12] Jadex BDI Agent System. http://vsis-www.informatik.uni-hamburg.de/projects/jadex/.

[13] Jade. http://jade.tilab.com/

[14] Spring Framework. http://www.springframework.org/.

[15] Nunes, C., et al. Comparing Stability of Implementation Techniques for Multi-Agent System Product Lines. Technical report, PUC-Rio, February, 2009.

[16] Fowler, M., Patterns of Enterprise Application *Architecture*. Addison-Wesley Professional, 2002.