The Journal of Systems and Software 83 (2010) 711–722

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Stability assessment of aspect-oriented software architectures: A quantitative study ☆

Ambra Molesini [a,*], Alessandro Garcia [b], Christina von Flach Garcia Chavez [c], Thais Vasconcelos Batista [d]

[a] Alma Mater Studiorum, Università di Bologna, Italy
[b] Pontificia Universidade Católica do Rio de Janeiro (PUC-Rio), Brazil
[c] Universidade Federal da Bahia, Brazil
[d] Universidade Federal do Rio Grande do Norte, Brazil

ARTICLE INFO

Article history:
Received 22 October 2008
Received in revised form 30 April 2009
Accepted 5 May 2009
Available online 15 May 2009

Keywords:
Aspect-oriented software architectures
Crosscutting concerns
Pointcuts
Style semantic composition
Architectural metrics

ABSTRACT

Design of stable software architectures has increasingly been a deep challenge to software developers due to the high volatility of their concerns and respective design decisions. Architecture stability is the ability of the high-level design units to sustain their modularity properties and not succumb to modifications. Architectural aspects are new modularity units aimed at improving design stability through the modularization of otherwise crosscutting concerns. However, there is no empirical knowledge about the positive and negative influences of aspectual decompositions on architecture stability. This paper presents an exploratory analysis of the influence exerted by aspect-oriented composition mechanisms in the stability of architectural modules addressing typical crosscutting concerns, such as error handling and security. Our investigation encompassed a comparative analysis of aspectual and non-aspectual decompositions based on different architectural styles applied to an evolving multi-agent software architecture. In particular, we assessed various facets of components' and compositions' stability through such alternative designs of the same multi-agent system using conventional quantitative indicators. We have also investigated the key characteristics of aspectual decompositions that led to (in)stabilities being observed in the target architectural options. The evaluation focused upon a number of architecturally-relevant changes that are typically performed through real-life maintenance tasks.

1. Introduction

Given the increasing volatility of contemporary software requirements, many organizations are pursuing the achievement of stable software architectures as a way to control their software development and maintenance costs (Jazayeri, 2002; Bahsoon and Emmerich, 2004). Hence, stability is often seen as a primary criterion to assess the value of constantly-changing software architectures (Bahsoon and Emmerich, 2004). Design stability is directly dependent on the ability of a software architecture to sustain its modularity properties and not succumb to modifications (Yau et al., 1985; Baldwin and Clark, 1999; Casais, 1995). It is well known that the achievement of stable architectures is largely dependent on the used composition mechanisms (Yau et al., 1985; Casais, 1995; Greenwood, 2007). In addition, recent empirical studies have also pointed out that design instabilities are directly proportional to the degree of crosscutting concerns (Greenwood, 2007; Figueiredo et al., 2008).

As a consequence, many researchers have claimed that aspect-oriented software development (AOSD) (et al., 2005) lead to architectural decompositions that cope better with changes to crosscutting concerns. Architecting aspect-oriented software designs basically relies on using new composition mechanisms that allow for quantifying otherwise crosscutting concerns. In fact, the core of existing aspect-oriented extensions (Garcia et al., 2006) to existing architecture description languages (ADLs) resides on the explicit support for expressing quantifications of architecturally-relevant crosscutting behaviors (Batista et al., 2006; Garcia et al., 2008). Aspect-oriented ADLs (Garcia et al., 2006; Navasa et al., 2005; Pinto et al., 2005; Quintero et al., 2005) foster fine-grained composition specifications by means of pointcut specifications. Architectural *pointcuts* select points – the *join points* – in of a behavioral architecture description where aspect composition is supposed to take place. Pointcuts allow for the flexible binding of "aspectual components" to the selected join points, a characteristic not supported by conventional ADLs (Garcia et al., 2006; Quintero et al., 2005).

☆ This paper is an extension of Molesini et al. (2008).

* Corresponding author. Tel.: +39 051 2093541; fax: +39 051 2093073.
E-mail addresses: ambra.molesini@unibo.it (A. Molesini), afgarcia@inf.puc-rio.br (A. Garcia), flach@ufba.br (C. von Flach Garcia Chavez), thais@ufrnet.br (T.V. Batista).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.05.022

However, there is no empirical understanding of the positive and negative effects of aspect-oriented (AO) software architectures in promoting architectural design stability. In particular, there is a pressing need for analyzing the extent of the benefits and drawbacks of the richer AO composition capabilities in the presence of architecturally-relevant changes. However, it has been empirically observed that flexible composition mechanisms might be detrimental to design stability (Yau et al., 1985; Casais, 1995; Greenwood, 2007). For instance, certain studies have detected that the versatility of multiple inheritance is one of the main causes of ripple effects in object-oriented (OO) software architectures (Casais, 1995).

The difficulty of analyzing the stability of aspect-oriented architecture designs stems from the fact that aspectual compositions can take significantly different forms according to the dominant architectural decompositions applied. Join point models are sensitive to the semantics of the architectural styles instantiated and composed in the context of a particular application. For instance, the join points in a client–server architecture are essentially different from the ones available in a blackboard architecture. Also, the set of crosscutting concerns and the nature of their manifestations might differ due to the interaction and topology constraints defined by certain styles. The situation becomes even more intricate in hybrid software architectures where multiple styles are composed in heterogeneous ways.

In this context, this paper presents an exploratory quantitative study (Section 2) where we have analyzed the stability of aspect-oriented software architectures designed for a multi-agent system (MAS). We have compared the influence of non-aspectual and aspectual decompositions on the architectural stability of crosscutting concerns found in architectural designs based on four architectural styles (Section 3). We present AO composition based on style semantics and examples of AO architectural designs and related pointcuts (Section 4). These alternative style-based designs of the same system were the target of typical architectural changes so that we could contrast ripple effects detected in both conventional and pointcut-based bindings (Section 5).

We have used a conventional suite of architectural stability metrics in order to search for evidence with respect to the following research questions: to what extent ripple effects increase or decrease when aspect-oriented compositions are used? how different types of crosscutting concerns and architectural styles influence positively and negatively the architecture stability? which join point models and other aspectual composition characteristics led to more architectural instabilities? (Section 6) Finally, we also discuss related work and provide a correlation of our findings with a previous implementation-level stability study (Greenwood, 2007) we have performed (Section 7) and draw some conclusions (Section 8).

2. Study settings

This section describes the configuration of our study – the criteria for selecting the target system and the definition of the analytical methodology (Section 2.1). We also present a brief overview of the selected target system (Section 2.2).

2.1. Methodological procedures

2.1.1. The case study

The first major decision concerning our investigation was the selection of the target system to be used in the case study. The Conference Management System (CMS) (Ciancarini et al., 1996) is a typical multi-agent system (MAS) with the purpose of providing automated support for a number of time-consuming activities related to the management of scientific conferences. The CMS meets a number of relevant criteria for our intended analysis. First and foremost, it is a realistic and non-trivial system with existing alternative implementations based on heterogeneous middleware systems (TuCSoN, xxxx; Mamei et al., 2005). These systems realize heterogeneous architectural designs, following classical and MAS-specific architectural styles. Second, the MAS domain entails complex software architectures with diverse categories of crosscutting concerns (Garcia et al., 2008). In particular, the CMS application encompasses a number of crosscutting concerns ranging from more widely-scoped system properties, such as coordination and error handling, to fine-grained agent features, such as agent autonomy, learning, and code mobility (Garcia et al., 2008; Garcia et al., 2004). Also, CMS is the most widely-used benchmark in the MAS community (Ciancarini et al., 1996).

2.1.2. Heterogeneous architecture designs

Each architectural design decision for existing CMS versions has been extensively discussed and evolved over time in a controlled manner. The first CMS (Garcia et al., 2004; Garcia et al., 2008) was a three-tiered application that embedded a blackboard component to manage conference information. The original CMS has progressively evolved in accordance to different MAS-specific hybrid styles, namely reflective blackboard (Silva et al., 2002), reactive coordination (TuCSoN, xxxx), and stigmergic coordination (Mamei et al., 2005). The evolved designs were developed in response to new emerging requirements and to address modularity problems identified in previous CMS versions. New requirements were also systematically identified in qualitative and quantitative assessments of the alternative CMS architectures that we have carried out over the past five years (Garcia et al., 2004; Garcia et al., 2008; Molesini et al., 2007).

We have derived ADL-based architectural representations from existing running implementations in Java and AspectJ (Garcia et al., 2008; Garcia et al., 2004) of CMS as the basis for our analysis. Respective documentation (e.g. Ciancarini et al., 1996; TuCSoN, xxxx; Mamei et al., 2005) was also exploited to refine the architectural descriptions expressed in ACME (Garlan et al., 1997) and AspectualACME (Garcia et al., 2006). The releases of the ADL representations were produced based on architecture-level modifications devired from a set of change requests; such releses were could not be produced from existing implementation releases as we needed to equally apply the same changes to all the architectural versions so that we could compare them.

2.1.3. Analysis steps

On investigating the influence exerted by styles on the manifestation of architectural crosscutting concerns, our analysis was divided in four major phases: (i) the first phase reviewed how classical styles were instantiated and composed to derive the original CMS architecture, (ii) the second phase identified occurrences of crosscuttings in such an original CMS architecture, (iii) the third phase involved the discussion of different styles used afterwards to implement alternative AO and non-AO CMS architectures, and (iv) the fourth phase analyzed the similarities and divergences of the crosscutting concerns found in the CMS (including the original and alternative architectures).

2.2. Conference Management System

The Conference Management System (CMS) is an application that supports the management of scientific conferences (Ciancarini et al., 1996). It involves several non-trivial design details, from the main organization issues to paper submission, peer review and proceedings production. The agents enrolled in this system represent a number of people involved, such as chairs and reviewers. Setting up and running a conference is a multi-phase process,

involving several individuals and groups. During the submission phase, authors send papers, and are informed that their papers have been received and have been assigned a submission number. In the review phase, the program committee (PC) has to handle the review of the papers: contacting potential referees and asking them to review a number of the papers. There are various strategies that can be used to distribute the papers, for example, titles and abstracts may be broadcast, then a bidding mechanism can be implemented, or the PC-chair decides to distribute the papers based on the expertise of the various PC-members. Also, the system must prevent the PC-chair from accessing or inferring information about their own submissions. In the final phase, authors need to be notified of these decisions and, in case of acceptance, must be asked to produce a revised version of their paper. The publisher has then to collect these final versions and print the proceedings.

3. Architectural designs for the CMS

This section presents the architectural designs for the CMS in terms of four architectural styles (Section 3.1) and discusses how each style deals with the crosscutting concerns that were identified in the original CMS architecture (Section 3.2).

3.1. Architectural styles

The original architecture of the CMS uses the blackboard style (Nii, 1986; Buschmann et al., 1996; Clements et al., 2007; Shaw et al., 1996). Additionally, three stylistic variations of the blackboard style were used to derive alternative architectural designs for the CMS: reflective blackboard (Silva et al., 2002), reactive coordination (TuCSoN, xxxx) and stigmergic coordination (Mamei et al., 2005) styles.

3.1.1. Blackboard style

The basic architecture of the CMS in the blackboard style consists of a blackboard, a collection of knowledge sources (KSs) and a control component. The `Blackboard_CMS` component is the central data storage structure (Fig. 1) where submitted papers, information about the authors, review templates and list of accepted papers can be inspected or updated by several KS components, such as `PC-Chair`, `Reviewer`, etc. Two connectors, `inspection` and `update`, describe the interaction between the KSs and the `Blackboard_CMS` component. The connectors embody access control policies to the blackboard, so that the PC-chair and reviewers have different read and write access rights. `Control_CMS` is the control component, responsible for managing the course of the complex workflow that characterizes the CMS. It can be viewed as a specialist in directing problem solving, by considering the overall benefit of the contributions that would be made by activating specific agents (`PC-Chair` or `Reviewers`). The `Control_CMS` component needs to be informed about `Blackboard_CMS` state changes regularly. This interaction follows a publish-subscribe protocol. `Control_CMS` interacts with each KS: (i) to query whether they can perform some action that contributes to the overall solution, and to select one KS, and (ii) to activate (or trigger) the execution of the selected KS. These interactions are described by the connectors `selection` and `activation`, respectively (Fig. 1). Since `Control_CMS` selects and activates the KSs, CMS agents are simple reactive agents and do not have complex cognitive skills.

3.1.2. Reflective blackboard style

The reflective blackboard style (Silva et al., 2002) combines two well-known architectural styles (Buschmann et al., 1996) to develop MASs: the *blackboard* and the *reflection* styles. The reflection style provides specific architectural elements – meta-objects and the meta-object protocol (MOP) – for dynamically changing the structure and behavior of the target blackboard component. Elements of the blackboard (e.g. tuples) are associated via MOP with meta-objects defined in the *control* component; they can be used to implement, for instance, error handling strategies and complex coordination protocols. In systems that follow the reflective blackboard style, application agents and data are encapsulated at the base level while control resides at the meta-level. Fig. 2 presents a reflective-blackboard view for the CMS. Communication is centralized on the `Blackboard_CMS` blackboard and the `Control_CMS` component is transparently inserted in the desired points of inter-agent communications by using reflective features. The `Control_CMS` component has a sophisticated internal

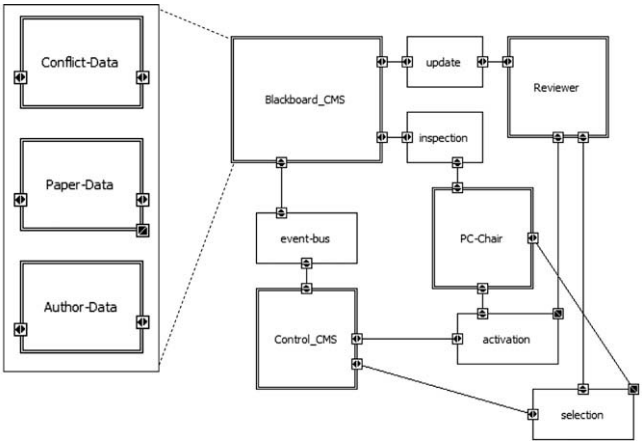

Fig. 1. Blackboard design for CMS.

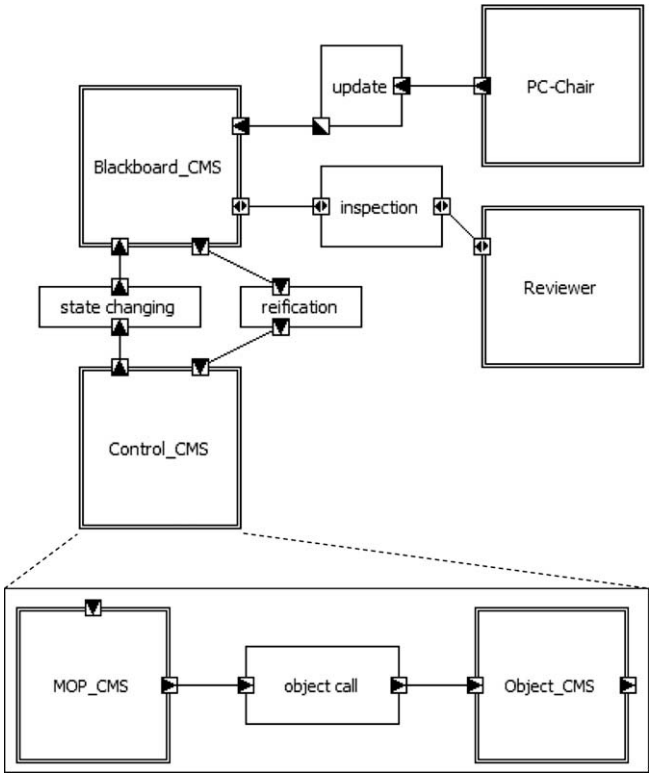

Fig. 2. Reflective blackboard design for CMS.

structure that consists of the MOP_CMS component and several Object_CMS components (the *meta-objects*). The MOP_CMS component and Object_CMS components are responsible for dynamically changing the structure and behavior of the Blackboard_CMS. Elements of the Blackboard_CMS are associated via MOP_CMS with the Object_CMSs defined in the Control_CMS component. This structure implies that rule enforcement on the CMS workflow (e.g., bidding and automatic matching based on keywords in paper assignment phase) and security policies (e.g. access control) can be properly encapsulated in separate meta-objects. This means that the interaction among Blackboard_CMS, PC-Chair and Reviewer can be simplified: it is not necessary to specify access control policies inside different connectors because these policies are managed by the meta-objects in a more natural way. For example, the access to specific information is given to agents via special tuples stored in Blackboard_CMS and created by the meta-objects. Now a change in an access policy implies the modification of the content of the tuple(s) produced by the meta-object responsible for this policy only and do not involve change in the Blackboard_CMS's interfaces or connectors.

3.1.3. Reactive coordination style

The reactive coordination style is a refinement of the blackboard style in order to address the management of complex coordination protocols. It provides a repository of tuples that can be accessed concurrently. The idea behind this style is that the behavior of a communication abstraction like a shared data space is easily defined as the observable state transition following a communication event. This is supported by *reactions*. A reaction is defined as a set of operations which may atomically produce effects on the coordination media state. Typically, a reaction is represented as a particular tuple and its content is written according to a specific reaction language. This style is adopted by TuCSoN (xxxx), an infrastructure for agent coordination in which a *tuple centre* represents the programmable coordination medium.

Fig. 3 presents a reactive-coordination view for the CMS. The TC_CMS component is the tuple centre. TC_CMS is composed by the Reaction_TS component and the Data_TS component – each of them is also a tuple space. Reaction_TS is the container for reactions written by the users (or by the agents) and Data_TS is the container for data in the system (information about papers, authors, etc.). These two components are strictly related: Reaction_TS monitors Data_TS by means of the monitoring connector. This connection allows the management of complex coordination patterns: when a new event occurs inside Data_TS, this information flows through the monitoring connector to the Reaction_TS component. If necessary, the state of Data_TS is changed by the Reaction_TS component by means of the information that flows through the data modification connector.

3.1.4. Stigmergic coordination style

The stigmergic coordination style is another refinement of the blackboard style, particularly suited for self-organizing systems. Stigmergy (Mamei et al., 2005) refers to the kinds of indirect interactions occurring among situated agents that, by affecting and sensing the properties of a shared environment, reciprocally affect each others' behavior. So far, the most widely used stigmergic mechanisms in MAS include pheromone-based behaviors, relying on agents depositing markers that the environment can diffuse and evaporate. This style presents a pheromone-based approach: tuples can be seen as pheromones, each of them with its own propagation rule that specifies both the intensity and the evaporation's scope of the pheromone. The tuple space acts only as a shared data-space where the tuples are stored and executed. TOTA (Mamei et al., 2005) is an infrastructure that enables stigmergic interactions among distributed agents.

Fig. 4 presents a stigmergic-coordination view for the CMS. The TS_CMS component represents the tuple space, a shared data-space like a blackboard. This component stores tuples and puts them in execution. The Tuple component is composed by three different components: Maintenance Rule, Propagation Rule, and Content. The Content component is an ordered set of typed fields representing the information carried on by the tuple. The Propagation Rule component determines how the tuple should be distributed and propagated across the network. Maintenance Rule determines how a tuple distributed structure should react to events occurring in the environment. Both Maintenance Rule and Propagation Rule can affect Content by means of the data-modification connectors. When a new event occurs inside TS_CMS this information flows through the event-bus connector to the Maintenance Rule component that is put in execution by the TS_CMS. The Maintenance Rule component changes the state of TS_CMS and, at the same time, this change could both: (i) trigger the execution of Propagation Rule by means of the common Tuple's interface to the state modification connector and, (ii) affect the Content component. The execution of the Propagation Rule component changes the state of TS_CSM that generates a new event and so on. Periodically, the Propagation Rule

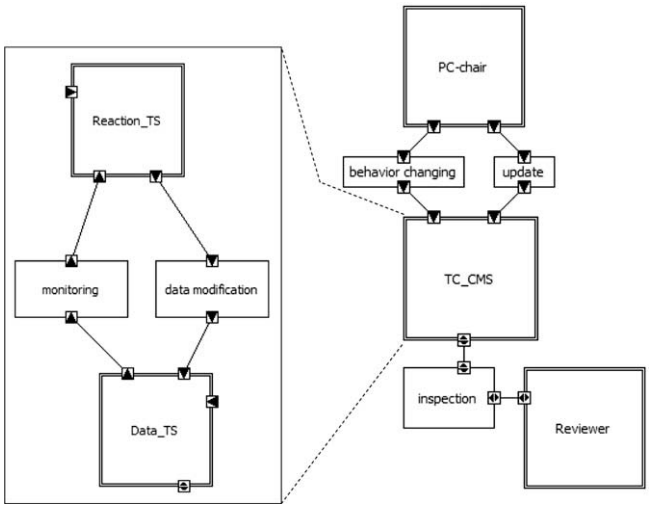

Fig. 3. Reactive coordination design for CMS.

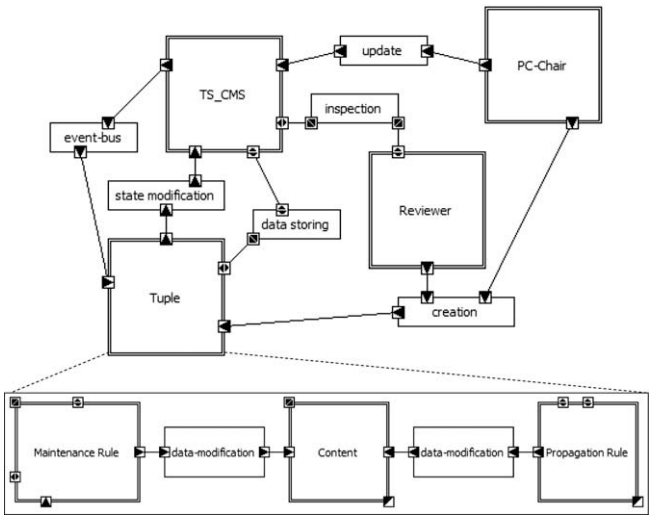

Fig. 4. Stigmergic coordination design for CMS.

component of all the tuples stored in TS_CMS are put in execution for the evolution of the system.

PC-Chair and Reviewer agents generate tuples and their internal components by the using the creation connector. Agents therefore become responsible for complex interaction protocols and this leads to coordination and security crosscutting concerns. When PC-Chair or Reviewer generates a tuple, it is stored in TS_CMS using data storing and update connectors. When an agent needs to retrieve a tuple, the inspection connector is used.

3.2. Architectural crosscutting concerns

Crosscutting concerns cut across the boundaries of modular units and tend to be *scattered* over several modules and *tangled up* with other concerns. The natural consequences of crosscutting are lower cohesion and stronger coupling between modular units, reduced comprehensibility, evolvability and reusability of software artifacts. In order to determine which concerns are or are not crosscutting in the CMS, we have used the following strategies (Garcia et al., 2008): (i) identify *architectural tangling*, that is, the mix of multiple concerns together in the same component or connector; (ii) identify *architectural interface bloat*, that is, search for evidence of increase in the complexity of component interfaces; and (iii) identify *architectural scattering*, that is, concerns that are spread in different components and connectors.

After a careful study involving the MAS-specific architectural styles presented in Section 3.1, we have identified three different crosscutting concerns: *coordination*, *error handling*, and *security*. The following subsections discuss the nature of each of these crosscutting concerns in terms of CMS architecture alternatives. We have used a metrics suite to support our identification of tangling, scattering and interface bloat (Sant'Anna et al., 2008) and the measures can be found at Chavez et al. (2007).

3.2.1. Coordination

Coordination is the regulation of diverse elements into an integrated and harmonious operation. Coordination is a crosscutting concern in the blackboard style. In fact, this style provides support only for the modular implementation of simple coordination protocols. Complex coordination protocols are necessarily realized by the multiple involved agents. As a result, responsibilities associated with the coordination protocol are scattered around all the agents involved in the protocol. For example, the PC-chair should manage all the workflow's steps from the submission phase to the final review. In each step, this agent should check if all the conference's rules are respected. Also, it is in charge of interacting with PC-members and coordinating them in order to assign each paper to the right number of suitable PC-members and collect the paper reviews. This process could be better modularized if the blackboard could provide more sophisticated coordination services to the PC-chair.

Modifications to the coordination protocol are very problematic because they involve the modification of the interfaces associated with all components. Consider, for example, the process of paper partitioning and assignment. Suppose that the PC-chair needs to change the paper partitioning and assignment process in order to keep it manageable while scaling up the conference dimension (often related to the number of submitted papers). Several modifications may be necessary in different parts of the CMS to deal with such a change: the PC-chair interfaces, the information generated for this protocol, pieces of the Control_CMS component and its interfaces (possibly because Control_CMS must trigger, for example, different Reviewer agents or a different number of reviewers) and also the activations of the agents. The Reviewers' interfaces and code must be changed in order to support this new protocol. Also, the blackboard interfaces need to be changed because it needs to store different kinds of information and to interact in new ways with the system' agents.

Coordination is not a crosscutting concern in the reflective blackboard and reactive coordination styles. In the former style, the coordination is realized by means of Object_CMSs that are allocated inside the control component, while in the reactive coordination style, the complex coordination protocols are implemented by reactions typically allocated inside the tuple centre. In our partitioning and assignment example, the protocol change only requires a modification of the specific Object_CMS that manages the protocol for the reflective blackboard style, and a modification on the reactions of the reactive coordination style.

3.2.2. Error handling

Error handling is widely recognized as a global design issue and has been extensively referred to in the literature as a classical crosscutting concern in systems following different kinds of architecture decompositions (Soares et al., 2002; Filho et al., 2006). In the blackboard style, error handling is a crosscutting concern since exceptional conditions must be propagated from the blackboard to several knowledge sources. Similarly, error handling is a crosscutting concern in the other styles as error handling involves several different entities (e.g., when communication exceptions occur during collaborations between knowledge sources). In fact, in the reflective blackboard and reactive coordination decompositions of the CMS system, agents and the coordination medium are involved in error handling. Each component should be equipped with specific code in order to react to system's failures.

3.2.3. Security

Information security is the process of protecting data from unauthorized access. In the context of CMS, access control is a security key problem, specially if, as it is often the case, PC-members are allowed to submit papers: in this case, one must prevent them from accessing (or even just inferring) information about their own submissions. In the blackboard style, security is a crosscutting concern because the code for the access control is spread over several agents (and tuples in the case of stigmergic coordination). System maintenance and improvement become harder because several different entities must be modified. The crosscutting manifestation is similar in the stigmergic coordination style. For instance, a modification in the CMS access policy implies in changing the Reviewer interface to the TS_CMS component and also the modification of the connectors between TS_CMS and Reviewer because the flow of information and the kind of tuples that the components exchange are different.

Security is not a crosscutting concern in the reflective blackboard style because the security policies could be incorporated in meta-objects inside the control component. Similarly, in the reactive coordination style, security is not a crosscutting concern because the code relative to a specific access policy is encapsulated in the agent class that implements this policy.

4. Aspect-oriented software architectures

Architectural aspects are expected to modularize those widely-scoped concerns, which cannot be localized in individual components using conventional architectural decompositions (Quintero et al., 2005; Chitchyan et al., 2005). Several aspect-oriented architectural description languages (AO ADLs) support aspectual decompositions and provide *aspects* (or equivalent mechanisms) to modularize crosscutting concerns. They also support *architecture-level pointcuts* for selecting join points that are relevant to the composition of aspects at the architectural level. Pointcuts select a set of join points based on some property; for instance, *syntax-based pointcuts* select join points based on the names of

Table 1
Blackboard's join point model.

Join point type	Where
Inspect blackboard, update blackboard	KS, Blackboard
Signal status change	Blackboard, control
Query source, select source, activate source	Control, KS

components and ports, while *style-based pointcuts* select join points based on style semantics (Chavez et al., 2009). In order to support aspect composition, a join point model that indicates the valid join points must be provided (Kiczales et al., 2001), usually by the style designer.

AspectualACME (Garcia et al., 2006) is an AO ADL that extends ACME (Garlan et al., 1997) with a specialized connector that localizes the interaction between components that play the role of aspects ("aspectual components") and regular components. In our study, we have defined four ACME families, one for each style described in Section 3.1. Furthermore, each family specification is related to a join point model for the corresponding style (Section 4.1). AspectualACME supports both syntax-based and style-based pointcuts, but in our study, only the AO designs in which aspect composition is based on style semantics have been used (Section 4.2).

4.1. Style-specific join point models

A *style-specific join point model* defines a set of join point types that are related to some architectural style. In this section, we present joint point models for three styles presented in Section 3.1. We depart from some documented uses of each style (Nii, 1986; TuCSoN, xxxx; Mamei et al., 2005) or at least one good source of style guide (Shaw et al., 1996; Clements et al., 2007; Silva et al., 2002) publicly available. Tables 1–3 present joint point models for the blackboard, reflective blackboard and reactive coordination styles. A join point model for the stigmergic coordination style can be found in Chavez et al. (2007).

Stylistic specializations of the blackboard style may strengthen its constraints, specialize element types, add element types, etc. This begs the question as to how the join point model of the derived style differs from the join point model of the base style. Our case study helped us to identify four different situations: (1) base joint points are reused in the derived style, (2) new kinds of join points are introduced by the derived style, (3) joint points overlap and must be reconciled in the derived join point model, and (4) base join points are not necessary in the derived style. These situations are illustrated in the following paragraphs.

4.1.1. Blackboard

Table 1 presents a joint point model for the blackboard style that comprises six joint point types: (i) *inspect blackboard* describes the points where data is read from the blackboard, in the context of an interaction with some KS; (ii) *update blackboard* describes the points in which some KS writes data in the blackboard; (iii) *query source* denotes the points where the Control makes a query over each KS to determine if they are potential contributors in order to select a KS to activate; (iv) *select source* denotes the points where the Control selects a KS; (v) *activate source* join point type describes the points where the Control puts in execution the more suitable KS; and (vi) *signal status change* describes the points in which the blackboard notifies its status changes (for example, the writing of data in the blackboard generates a status change event).

4.1.2. Reflective blackboard

This style combines two base styles through the union of their design vocabularies, and conjoining their constraints. An hybrid join point model is defined from the resulting types defined by the conjunction. The join point model defined for the reflection style consists of three join point types: (i) *reification* denotes the points where the base level notifies its status changed; (ii) *metaobject call* denotes the points where the `MOP_CMS` puts in execution the more suitable `Object_CMS`; and (iii) *adaptation* denotes the points where the `Object_CMS` changes the state of base level. Join points from the two base join point types may overlap: *signal status change* (blackboard style) and the *reification* (reflection style) join point types need to be reconciled. This unification allows the `MOP_CMS` component to capture the events generated by the `Blackboard_CMS`. The *query source*, *select source* and the *activate source* join point types become unnecessary. Table 2 presents join point models for the blackboard and the reflection styles, and the resulting join point model for the reflective blackboard style.

4.1.3. Reactive coordination style

In the reactive coordination style, the `TM_CMS` component (Fig. 3) is a specialization of the blackboard component that comprises two tuples spaces (`Data_TS` and `Reaction_TS`). The join point model for this style introduces three new join point types (Table 3): (i) *data modification* denotes the points where the `Reaction_TS` component changes the tuples stored inside the `Data_TS` component; (ii) *behavior changing* denotes the points where an agent changes the contents of the `Reaction_TS`; and (iii) *notify status change* denotes the points where the `Data_TS` component notifies its status changes. The *notify status change* join point overlaps with *signal status change* (blackboard style) and they should be reconciled. The *query source*, *select source* and the *activate source* join points (blackboard style) are not necessary in the reactive coordination style because the information between the `Reaction_TS` and the agent are exchanged by means of data stored in the `Data_TS` component.

4.2. Architecture-level pointcuts

Architecture-level pointcuts select join points that are relevant to the composition of aspects at the architectural level. Table 4 presents style-based pointcuts for the Coordination concern. These pointcuts select join points of interest based on the semantics of blackboard and reflective blackboard styles. For instance, the *change coordination status* pointcut, associated with the blackboard style, matches join points in which a knowledge source updates the

Table 2
Reflective Blackboard's join point model.

Blackboard	Reflection	Reflective blackboard
Inspect blackboard		Inspect blackboard
Update blackboard		Update blackboard
Query source, select source, activate source		–
Signal status change	Reification	*reification* (reconciled)
	Metaobject call	Metaobject call
	Adaptation	Adaptation

Table 3
Reactive Coordination's join point model.

Blackboard	Reactive coordination
Inspect blackboard	Inspect blackboard
Update blackboard	Update blackboard
Query source, select source, activate source	–
Signal status change	*notify status change* (reconciled)
	Data modification
	Behavior changing

blackboard or the blackboard notifies its status changes by means of a particular event.

Pointcuts for error handling select points where failures can be observed: the *coordination exception* pointcut matches points of possible failures in coordination; the *data storing exception* pointcut matches points of possible failures in the data storing; the *data reading exception* pointcut matches points of possible failures in the data reading; the *communication exception* pointcut matches points of possible failures in communication. Pointcuts for Security specify points where security is an issue of concern: the *data secure storing* pointcut matches points where the the data must be stored in a secure way; the *data secure reading* pointcut matches points where the data must be read in a secure way.

Fig. 5 presents an example in AspectualACME in which a syntax-based pointcut is used for selecting the points for composing the Coordination concern, represented by the `Coordinator` component. The pointcut matches components with a port named *inspectBB*. The `coord` connector describes the interaction between `Coordinator` and the components that will be coordinated (`PC_Chair` and reviewers `Rev1`, `Rev2` and `Rev3`). The *glue* clause specifies that the crosscutting concern (`source`) affects the base (`sink`) after reaching the selected join points. The *Attachments* block provides the bindings: the `readCoordInfo` port of `Coordinator` is connected to every port named `inspectBB`, regardless of components' name (denoted by the * wildcard).

5. Analysis: AO vs. non-AO architectures

This section presents both the measurement procedures and data analysis. We have used classical evaluation indicators and procedures to analyze the stability of the CMS architectures under assessment (Section 2.1), as described in Section 5.1. The analysis was performed based on a set of changes (Section 5.2) and according to two different perspectives.

First, we have analyzed the data to compare the stability of non-aspect-oriented and aspect-oriented software architectures (Section 5.3). Following this perspective, we have also analysed which particular types of changes or architecture characteristics led to a higher degree of instability in non-aspect-oriented or aspect-oriented software architectures. In particular, we have examined the instabilities in the architectural composition descriptions in both non-AO and AO architectures (Section 5.4).

Second, we have disregarded the non-AO software architectures and focused on analysing stability-related phenomena observed in aspect-oriented software architectures based on different styles. For instance, we have tried to answer questions, such as: (i) Do richer joinpoint models lead to more stable aspect-oriented architectures?, and (ii) What are the aspectual composition characteristics that led to more instabilities? This second step of our analysis is described in Section 6.

5.1. Measurement procedures

We have used a metrics suite to support the architecture stability assessment across the releases of all the non-AO and AO versions of the CMS system. The suite is composed of metrics for quantifying change propagation, including the number of components added and changed, and connectors added and changed. We have also counted the number of bindings added or changed in the non-AO architectures, and the number of pointcuts added or changed in the AO architectures. The purpose of using these measures is to assess the change effects, when implementing the various modification scenarios. The higher the number of changes, higher is the probability of architectural ripple effects manifesting. In addition, change impact metrics enable us to draw paradigm-independent quantitative assessments of both AO and non-AO architectures generated. Such change propagation metrics are impartial in the sense they also help to capture potential side effects caused by the aspectual decompositions in the presence of different types of architectural modifications.

5.2. Change scenarios

In order to perform the stability measurements, a set of eight change scenarios have been applied to all the non-AO and AO architecture versions of the CMS system. We have collected the results counting the tally of new elements and the ones that suffered some change impact. The selected change scenarios are representative of typical types of architecturally-relevant changes we have observed in CMS releases. They are also varied in terms of types of modifications performed. For instance, we have applied fine-grained and coarse-grained changes involving both crosscutting and non-crosscutting concerns. Such changes were carried out in both the non-AO and AO architectures in order to measure their architecture stability. The purpose of the heterogeneous change scenarios was to expose the non-AO and AO architectures to recurring architecturally-relevant maintenance tasks.

The list of the scenarios is as follows: (S1) introduction of sub-committees and vice-chairs, (S2) introduction of the external reviewers, (S3) refinement of security constraints, (S4) introduction of new coordination protocols, (S5) improvement in the error handling strategies, (S6) introduction of new organizational rules and changes in the existing rules, (S7) evolution in the agents' capabilities, and (S8) introduction of a new paper assignment strategy. The main target of scenarios S1, S2, S6–S8 were non-crosscutting concerns, while the others were crosscutting concerns.

Table 4
Style-based pointcuts for coordination.

Style	Pointcuts	Join point types
Blackboard	Change coordination status	Update blackboard, signal status change
	Read coordination info	Inspect blackboard
	Invoke coordination source	Select source, activate source
Reflective blackboard	Reify coordination	Update blackboard, reification
	Read coordination info	Inspect blackboard
	Change coordination data	Update blackboard metaobject call, adaptation

```
Family BlackboardFam
        extends SharedDataFam with {
   Component Type BB
           extends SharedDataT with {
      Ports inspect, update : p_provide ... };
   Component Type KS = {
      Ports inspectBB, updateBB : p_use ... };
   Component Type Control = { ... };
   ...
}
System CMS : BlackboardFam =
      new BlackboardFam extended with {
   Component PC_Chair: KS;
   Component Rev1, Rev2, Rev3: KS;
   ...
\*  Coordination Crosscutting Concern  *\
   Component Coordinator = {
      Port changeCoordStatus: ... ;
      Port readCoordInfo: ... ;
      Port invokeCoordSource: ... ; };
   Connector coord = {
      crosscuttingRole source; baseRole sink;
      glue source after sink; };
\*  Pointcut description - aspectual bindings  *\
   Attachments {
      Coordinator.readCoordInfo to coord.source;
      coord.sink to *.inspectBB;
   ... }
}
```

Fig. 5. Syntax-based composition for coordination in the blackboard design for CMS.

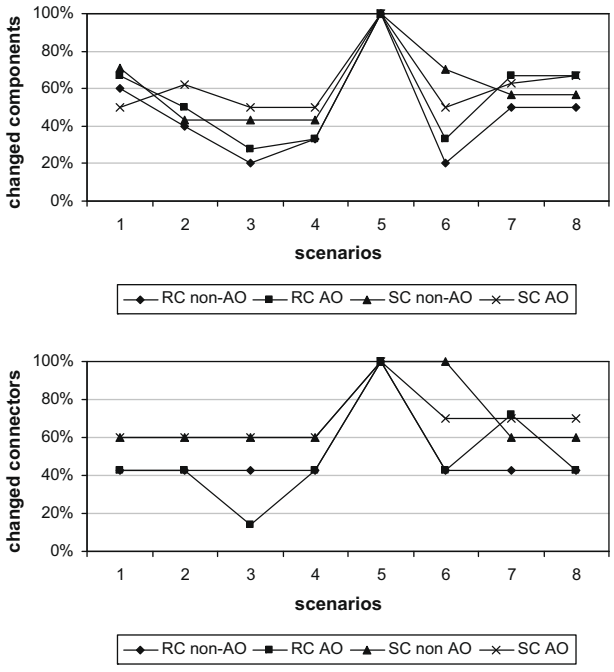

Fig. 6. Change propagation measures.

5.3. Stability analysis: AO vs. non-AO architectures

This section reports the experimental results of the measurement process. The discussion here focuses on the most interesting results, while the raw data and exhaustive result descriptions can be found at Chavez et al. (2007). Fig. 6 reports the change propagation measures of the Reactive Coordination (RC) and Stigmergic Coordination (SC) architectures through all the releases generated after each scenario. The graphic contrasts the outcomes of the non-AO and AO versions for both these stylistic decompositions. The number of changes is shown in percentage values. The graphic concentrates on the number of components and connectors changed because they have captured significant differences through the scenarios; also, new components and connectors are added only in the first two scenarios and no major differences were observed.

5.3.1. Non-AO architecture fragilities

AO architectures were sensibly more stable when the main focus of a change was a crosscutting concern. This observation was consistently visible through measurements gathered from the CMS releases generated in scenarios 3 and 4. Even though Fig. 6 shows similar results for AO and non-AO designs, the AO architectures generally require more new components for realizing a change while the non-AO architectures require existing components to be modified more extensively for the same change. This finding shows that the AO architecture satisfies more closely the desirable open-closed principle (Meyer, 1988) in the presence of evolving crosscutting concerns. In other words, AO architectures are more open to extensions of modules realising crosscutting concerns and closed to modifications.

The only exception was the scenario 5 when global exception handling strategies were the target of the alterations. All the components needed to be changed in both AO and non-AO versions. This occurred because exception declarations in the CMS architectures, whether aspectual or not, are defined in interfaces of all the components. For instance, a network recovery protocol must support communication failure exceptions while a re-coordination protocol must support coordination failure exceptions. The intrinsic nature of the change in S5 required that exception declarations needed to be all changed so that proper exceptional information was being propagated throughout the interface boundaries. For example, the new error handling policy took into account time-out issues and internal component information and it needed to be propagated together with the exceptions throughout all the system layers.

5.3.2. AO architecture fragilities

Interestingly, we have observed that non-AO architectures tended to be more stable when the main focus of a change was a non-crosscutting concern. In fact, ripple effects were observed in the AO architectures through scenarios 2, 7 and 8; they comprise three out of five scenarios where non-crosscutting concerns are the main change target (Fig. 6). It is also important to highlight that the superiority of non-AO architectures was independent of stylistic choices (Section 3.1), join point models (Section 4.1), and types of changes (Section 5.2) – whether coarse-grained or fine-grained. This finding somehow reinforces results observed in an implementation-level empirical study that we have recently conducted Greenwood (2007). However our stability analysis in the previous study was not carried out at the architectural level and the target application was realizing an N-Tier software architecture.

We have observed that the main reason for such ripple effects is that widely-scoped crosscutting concerns tend to naturally affect the same join points related to components and connectors realizing

non-crosscutting concerns. Hence, when changing the later ones, a channel of changes tends to also traverse the specification of the components and connectors in the AO architecture versions. There were cases where pointcuts related to exception handling, coordination, and security all shared the same join points.

5.4. Composition-level instabilities

5.4.1. Conventional bindings vs. pointcuts

Figs. 7–9 report the composition-level measurements, i.e. pointcut-level change measures in AO architectures and binding-level change measures in non-AO architectures. We concentrated again only on the changed pointcuts and bindings because the addition metrics did not present significant differences through most of the scenarios. We have considered that significant ripple effects occurred when the number of pointcuts or bindings changed was higher than two. As a result, pointcuts were more stable than conventional bindings in the blackboard architecture (Fig. 7). Major instabilities of coordination-specific bindings were observed in three scenarios (4, 6 and 8) against only one major coordination pointcut stability (scenario 7). A security-related instability case (scenario 5) was also observed in the non-AO blackboard version of CMS. More instabilities were observed in the exception handling bindings in the AO versions than in the non-AO versions for the reasons mentioned above. They also needed to be changed when the focus of a change is not exception handling. This finding was also detected in the other styles (Figs. 8 and 9).

5.4.2. Stylistic compositions might ameliorate crosscutting

It is often the case that the combination of stylistic rules are performed in order to produce hybrid architectures that improve the satisfaction of multiple system requirements. Composition of styles might ameliorate or eliminate the presence of crosscutting in non-AO architectures, sometimes more effectively than AO architectures. For instance, reactive coordination architectures fall in this scenario for the security concern; it was helpful to modularly capture certain security policies associated with illegal accesses to the blackboard. As a consequence, the crosscutting nature of security was reduced. In fact, only one ripple effect in scenario 7 is shown in Fig. 9. However, the non-AO version was still more superior in this particular case. The reason is that aspect-oriented software architectures have shown to be more effective for cases that require finer-grained composition mechanisms. On the other hand, the specific stylistic compositions in the non-AO reactive coordination architectures were more effective for addressing coordination (Fig. 9).

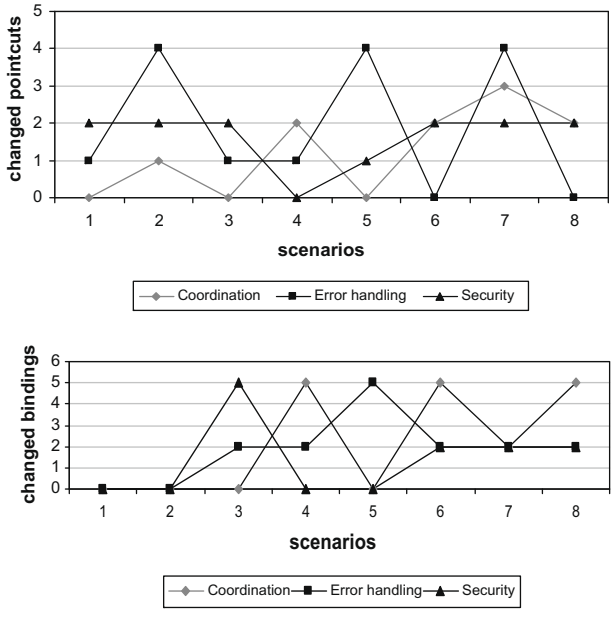

Fig. 7. Changed AO (top) and non-AO (bottom) bindings for BBS.

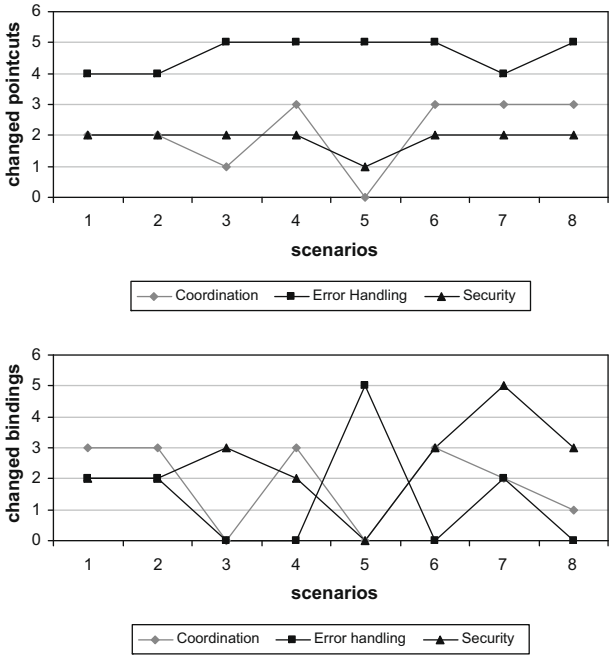

Fig. 8. Changed AO (top) and non-AO (bottom) bindings for RBBS.

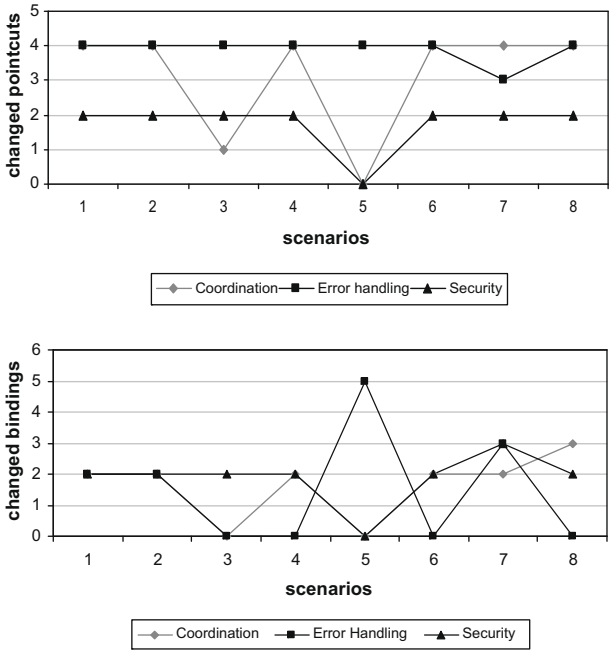

Fig. 9. Changed AO (top) and non-AO (bottom) bindings for RCS.

6. Analysis: characterizing stable aspect-oriented architectures

This section presents a more detailed analysis of potential factors that led to the beneficial or harmful use of aspect-oriented

composition mechanisms through the different stylistic architecture decompositions.

6.1. Do richer join point models increase architecture stability?

Hybrid architectural styles, such as reflective blackboard and stigmergic coordination, offer richer join point models in the sense they expose more join point types to be used in aspectual bindings (Section 4.2). This means that software architects have more powerful means to describe aspectual compositions and, therefore, more opportunities to modularly capture intricate crosscutting concerns. For instance, AO reflective blackboard architectures can include aspectual bindings at certain points not available in aspect-oriented blackboard architectures. In addition to the blackboard-specific join point types (Table 1), aspects in reflective blackboard architectures can also be bound at meta-level events, including reification, metaobject call, and adaptation (Table 2).

Given the higher expressiveness of aspect-oriented hybrid architectures, someone could hypothesize such architectural designs tend to be more stable than those offering limited types of join points. However, we have observed that richer join point models do not necessarily lead to more stable architectures (Section 5.4). In certain circumstances, AO blackboard architectures were visibly more stable than AO reflective blackboards and AO stigmergic architectures. Fig. 10 shows that the number of changes in conventional components and conventional connectors were consistently lower in most releases of the AO blackboard architectures. The only evident exception was the seventh release, when agent capabilities were modified. This reinforces the fact that software architects need to carefully consider the selection of more simplistic AO architectural decompositions, especially when composition flexibility is not an overarching design goal.

6.2. What are harmful aspectual composition characteristics?

We have observed two main aspectual composition factors that led to the manifestation of architectural instabilities. The first factor was the definition of "partial" aspectual compositions, such as in the exception handling case discussed in Section 5.3. Exception handling was only partially aspectized in the sense that exception detection was left as a responsibility assigned to the non-aspectual components; only the exception handlers were implemented by the aspectual components. As a result, only pointcuts for exception handling were created. In such cases, certain changes related to the addition of new exceptions required often modifications in both aspectual and non-aspectual components. The key problem was that strategies for exception detection and handling sometimes are strongly coupled.

The second and more influencing factor was higher density of aspectual components affecting the same join point. Such a high density causes a strong, albeit indirect, coupling amongst all the aspects involved. As a consequence, when a change implied a modification or removal of that join point, a number of pointcuts and interfaces of the aspectual components need to be modified. Although less problematic, such a strong inter-aspect coupling also caused some side effects when a change affected one of the involved aspectual components.

The observation above led us to infer that the sharing of join points by different architectural aspects should be carefully applied by architects and designers. We have noticed that such multiple aspectual extensions to the same behaviour entail similar ripple effect problems observed in large specialisation trees and multiple inheritance trees in OO designs (Casais, 1995). However, when designing AO architectures, designers can anticipate high density of shared join points and avoid it whenever it is possible. For instance, in order to not be detrimental to design stability, architects might decide for prioritizing the "aspectization" of a subset of crosscutting concerns that do not interact much with each other. In fact, in previous implementation-level studies, we have observed that high degree of concern overlaps might be harmful to the software stability (Greenwood, 2007; Figueiredo et al., 2008).

7. Related work and study limitations

This section presents a comparison with related work (Section 7.1) and discusses study constraints and imperfections (Section 7.2).

7.1. Comparison with related work

The literature on architectural aspects provides some examples of classical aspects at architecture design, such as error handling, persistence, distribution, and coordination (Garcia et al., 2006; Navasa et al., 2005; Pinto et al., 2005; Quintero et al., 2005). However, there is not much discussion on the impact of AO compositions on the architectural stability, thereby giving the impression that those concerns should be always aspectized. There is little related work focussing either on the quantitative assessment of AO architectural solutions in general, or on the empirical investigation about the design stability of AO decompositions (Greenwood, 2007). Substantial empirical evidence is missing even for crosscutting concerns that software engineers face every day, such as persistence, distribution and error handling.

As noticed from our observation in Section 5.4, the kind of decomposition supported by different conventional styles may

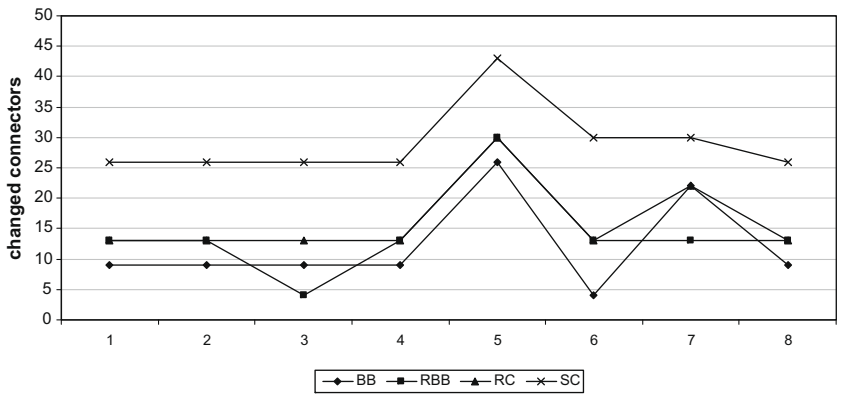

Fig. 10. Changed conventional connectors.

favor the clean modularization of some concerns while others may be not well-modularized. In fact, styles have directly interfered with the nature of the crosscutting concerns at the architectural description of the CMS system (Figs. 7–9). The stability of the aspectual compositions (i.e. based on pointcuts) were observed to be style-dependent, such as coordination. The chosen styles in different CMS versions emphasized the separation of some concerns of the problem and suppressed others. Hence, some concerns are expected to be well localized within specific kinds of modular units defined by the style, while others are expected to crosscut their boundaries. For instance, coordination was not crosscutting in architectures following the reflective blackboard and reactive coordination styles. Additional studies should be carried out to produce a broader catalogue of style-specific crosscutting concerns for supporting software architects.

7.2. Study constraints and imperfections

Even though this study fully satisfies our initial goal of providing a first exploratory investigation on the impact of evolving AO architectures on design stability, our procedures certainly have some limitations. These limitations will contribute to further explorations using other experimental procedures and systems as targets. For instance, our analyses are restricted to the instances of architectural styles used in different CMS releases. Hence, next studies should assess the stability of AO architectures when combined with the application of other styles not investigated in this study. Further studies could also investigate the stability impact of aspectizing other crosscutting concerns not investigated here, such as persistence and distribution. In our previous studies, we have analyzed the stability of such crosscutting concerns but from the implementation point of view only.

Second, our analyses concentrate on the history of architecture alternatives for one software system providing, therefore, a single point of observation. However, our target case study is a representative choice of a number of blackboard-based information systems for several reasons. It addresses recurring characteristics and crosscutting concerns for such systems, such as distribution, error handling, security, and coordination. In addition, the analysed changes are heterogeneous and capture typical change requests within software projects from this nature. More importantly, the design of the CMS system realizes best design practices, which have being systematically being enhanced through the years (Section 2). It provides evidence that the observerd negative changes in AO and non-AO architectures (Sections 5 and 6) are not merely a matter of lack of systematic design choices.

8. Conclusions

The transfer of aspect-oriented technologies to the mainstream of the software development is largely dependent on our ability to empirically understand their positive and negative effects on design stability and other qualities. Stability occupies a pivotal position in the design of good system architectures. Building stable architectures is a challenging task mainly because the architects need to reason and make decisions with respect to a number of architecturally-relevant crosscutting concerns. In this way, the main contribution of our work was to systematically analyze to what extent AO architectures are stable in the presence of different types of changes.

We have adopted the conference management system as a running example for assessing various facets of design stability in non-AO and AO architectural decompositions. This included the analysis of three crosscutting concerns (Section 3.2) for this system. One of the main outcomes was that the AO architectures tended to have a more stable design particularly when a change targeted a crosscutting concern – the AO architectures tended to require less invasive changes. However, they did not scale well when changes targeted non-crosscutting concerns. Ripple effects were often observed in pointcut specifications. As observed in our study, one of the potential reasons is that multiple pointcuts, defined to capture different widely-scoped aspects, commonly share join points associated with evolving base elements. Hence, the aspects themselves are tightly coupled to each other via volatile architectural elements. If these elements are changed or removed, they tend to cause instabilities in the AO architecture alternatives, thereby confirming the outcome of a recent empirical study targeting the use of aspects for framework architecture composition (Lobato et al., 2008).

Our study outcomes associated with exception handling aspects diverged from our previous observations in an evolving N-Tier architecture (Greenwood, 2007). In the previous investigation, the AO architecture has shown to be more stable in the presence of exception handling aspects. In the CMS case, we have detected the contrary: a considerable number of changes manifested in the exception handling pointcuts. The reason was that the exception-related changes in the CMS releases involved the incorporation and change of exception interfaces (Section 5.3); in our previous studies, the changes were restricted to the exception handlers modularized by the exception handling aspects. Also, a reduction of instabilities was observed in the non-aspectual decompositions when hybrid architectural styles were applied. Finally, against common intuition, we have observed that pointcuts with higher degree of quantification had no impact on the (in)stability of aspect-oriented software architectures.

Acknowledgements

This work is supported in part by the European Commission Grant IST-33710 – Aspect-Oriented, Model-Driven Product Line Engineering (AMPLE), Grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe), Grant 486125/2007-6: Brazilian Council for Scientific and Technological Development(CNPq), and Grant 219/2008: Fundação de Amparo à Pesquisa do Estado da Bahia (Fapesb).

References

Bahsoon, R., Emmerich, W., 2004. Evaluating architectural stability with real options theory. In: Proceedings of the 20th IEEE International Conference on Software Maintenance, 2004, pp. 443–447.

Baldwin, C.Y., Clark, K.B., 1999. Design Rules: The Power of Modularity, vol. 1. MIT Press, Cambridge, MA, USA.

Batista, T. et al., 2006. Reflections on architectural connection: seven issues on aspects and ADLs. In: Early Aspects at ICSE 06, ACM Press, 2006, pp. 3–10. doi:<http://doi.acm.org/10.1145/1137639.1137642>.

Buschmann, F. et al., 1996. Pattern-Oriented Software Architecture: A System of Patterns, vol. 1. Wiley.

Casais, E., 1995. Managing class evolution in oo systems. OO Softw. Compos., 201–244.

Chavez, C. et al., 2007. Empirical studies on architectural composition. <http://www.dcc.ufba.br/~flach/Studies/>.

Chavez, C., Garcia, A., Batista, T., Oliveira, M., Sant'Anna, C., Rashid, A., 2009. Composing architectural aspects based on style semantics. In: Proceedings of the ACM Interantional Conference on Aspect-Oriented Software Development (AOSD), Charlottesville, USA, 2009, pp. 111–122.

Chitchyan, R. et al., 2005. Survey of aspect-oriented analysis and design approaches. <http://www.aosd-europe.net/documents/index.htm/analys.pdf>.

Ciancarini, P. et al., 1996. A case study in coordination: conference management on internet. <ftp://ftp.cs.unibo.it/pub/cianca/coordina.ps.gz>.

Clements, P. et al., 2007. Documenting Software Architectures – Views and Beyond, SEI Series in Software Engineering, 9th Printing, Addison-Wesley.

Figueiredo, E. et al., 2008. Evolving software product lines with aspects: an empirical study on design stability. In: ICSE'08: Proceedings of the 30th International Conference on Software Engineering, ACM, New York, NY, USA, pp. 261–270. doi:<http://doi.acm.org/10.1145/1368088.1368124>.

Filho, F. et al., 2006. Exceptions and aspects: the devil is in the details. In: Young, M. et al. (Eds.), SIGSOFT FSE 2005: Proceedings. ACM, pp. 152–162.

Filman, R. et al. (Eds.), 2005. Aspect-Oriented Software Development. Addison-Wesley, Boston.
Garcia, A. et al., 2008. Taming heterogeneous agent architectures with aspects. Communications of the ACM 51 (5), 75–81.
Garcia, A. et al., 2004. Aspectizing multi-agent systems: from architecture to implementation. In: SELMAS 2004: Proceedings, pp. 121–143.
Garcia, A. et al., 2006. On the modular representation of architectural aspects. In: Gruhn, V., et al. (Eds.), EWSA 2006: Proceedings, LNCS, vol. 4344, Springer, 2006, pp. 82–97.
Garlan, D., Monroe, R.T., Wile, D., 1997. ACME: An architecture description interchange language. In: CASCON'97: Proceedings, Toronto, Ontario, pp. 169–183.
Greenwood, P. et al., 2007. On the impact of aspectual decompositions on design stability: an empirical study. In: Ernst, E. (Ed.), Proceedings of the 21st European Conference ECOOP, Berlin, Germany, July 30–August 3, 2007, Lecture Notes in Computer Science, vol. 4609, Springer, pp. 176–200
Jazayeri, M., 2002. On architectural stability and evolution. In: Ada-Europe'02: Proceedings of the Seventh Ada-Europe International Conference on Reliable Software Technologies, Springer-Verlag, London, UK, pp. 13–23.
Kiczales, G. et al., 2001. Getting started with AspectJ. Com. ACM 44 (10), 59–65.
Lobato, C., Garcia, A., Kulesza, U., Staa, A., Lucena, C., 2008. Evolving and composing frameworks with aspects: the mobigrid case. In: Seventh IEEE International Conference on Composition-Based Software Systems (ICCBSS), Madrid, Spain, 2008, pp. 25–29.
Mamei, M. et al., 2005. Programming stigmergic coordination with the TOTA middleware. In: Dignum, F., et al. (Eds.), AAMAS 2004: Proceedings, ACM Press, pp. 415–422.
Meyer, B., 1988. Object-Oriented Software Construction. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
Molesini, A., Garcia, A., Chavez, C., Batista, T., 2007. On the interplay of crosscutting and MAS-specific styles, In: Oquendo, F. (Ed.), ECSA 2007: Proceedings, LNCS, vol. 4758, Springer, 2007, pp. 317–320.
Molesini, A., Garcia, A., Chavez, C., Batista, T., 2008. On the quantitative analysis of architecture stability in aspectual decomposition. In: Kruchten, P., Garlan, D., Woods, E. (Eds.), Proceedings of Seventh IEEE/IFIP Working Conference on Software Architecture (WICSA 2008), IEEE Computer Society, Los Alamitos, CA 90720-1314, 2008, pp. 29–38, Seventh IEEE/IFIP Working Conference on Software Architecture (WICSA 2008), February 18–22, 2008, Vancouver, BC, Canada. doi: <http://doi.ieeecomputersociety.org/10.1109/WICSA.2008.26>.
Navasa, A. et al., 2005. Aspect modelling at architecture design. In: Morrison, R., et al. (Eds.), EWSA 2005: Proceedings, LNCS, vol. 3527, Springer, 2005, pp. 41–58.
Nii, P., 1986. The blackboard model of problem solving. AI Mag. 7 (2), 38–53.
Pinto, M. et al., 2005. A dynamic component and aspect-oriented platform. Comput. J. 48 (4), 401–420.
Quintero, C. et al., 2005. Architectural aspects of architectural aspects. In: Morrison, R., et al. (Eds.), EWSA 2005: Proceedings, LNCS, vol. 3527, Springer, 2005, pp. 247–262.
Sant'Anna, C., Lobato, C., Kulesza, U., Garcia, A., Chavez, C., Pereira de Lucena, C.J., 2008. On the modularity assessment of aspect-oriented multiagent architectures: a quantitative study. International Journal on Agent-Oriented Software Engineering (IJAOSE) 2 (1), 34–61.
Shaw, M. et al., 1996. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.
Silva, O. et al., 2002. The reflective blackboard pattern: architecting large multi-agent systems. In: Garcia, A., et al. (Eds.), SELMAS 2002, LNCS, vol. 2603, Springer, pp. 73–93.
Soares, S. et al., 2002. Implementing distribution and persistence aspects with AspectJ. In: OOPSLA 2002: Proceedings, ACM Press, pp. 174–190. doi:<http://doi.acm.org/10.1145/582419.582437>.
TuCSoN at Source Forge. <http://tucson.sourceforge.net>.
Yau, S. et al., 1985. Design stability measures for software maintenance. IEEE Trans. Software Eng. 11 (9), 849–856.

Ambra Molesini has a research grant at DEIS (Department of Electronic, Computer Science and Systems) of the Alma Mater Studiorum - Università di Bologna. She received her Ph.D and M.Sc. cum laude both from the Alma Mater respectively in 2008 and 2004. Her research is focused on agents, coordination, security, software engineering in general, and agent-oriented software engineering (AOSE) in particular.

Alessandro Garcia is an Assistant Professor at the Informatics Department of the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil. He completed his Ph.D. research studies in March 2004 at PUC-Rio (Brazil) in cooperation with University of Waterloo (Canada). He received his M.Sc. degree in Computer Science from Campinas State University (UNICAMP), Brazil. His research interests include software architecture, aspect-oriented software development, exception handling, empirical assessment of contemporary modularization techniques, and multiagent systems.

Christina Chavez is an Associate Professor at the Computer Science Department of the Federal University of Bahia (UFBA), Brazil. She received her Ph.D. degree in Computer Science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil, and her M.Sc. degree in Computer Science from Campinas State University (UNICAMP), Brazil. Her current research interests include software design, software evolution and aspect-oriented software development.

Thais Batista is an Associate Professor at the Computer Science Department of the Federal University of Rio Grande do Norte (UFRN), Brazil. She received her Ph.D. and M.Sc. degrees in Computer Science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil. Her current research interests include software architecture, aspect-oriented development, and distributed systems.