On the Quantitative Analysis of Architecture Stability in Aspectual Decompositions

Ambra Molesini  Alessandro Garcia  
Lancaster Univ., United Kingdom  
{a.molesini,a.garcia}@lancaster.ac.uk

Christina Chavez  Thais Batista  
UFBA, Brazil  UFRN, Brazil  
flach@dcc.ufba.br  thais@ufrnet.br

Abstract

Architectural aspects are expected to modularize widely-scope concerns that naturally crosscut the boundaries of system components at the software architecture level. However, there is no empirical knowledge about the positive and negative influences of aspectual decompositions on design stability. This paper analyzes the influence exerted by the aspect-oriented composition mechanisms in the stability of crosscutting concerns in an evolving multi-agent software architecture. Our investigation encompassed a comparative analysis of aspectual and non-aspectual decompositions based on different architectural styles. 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. The evaluation focused upon a number of architecturally-relevant changes that are typically performed through real-life maintenance tasks.

1. Introduction

Proponents of aspect-oriented software development [9] claim that aspectual software decompositions cope better with changes in the presence of the so-called crosscutting concerns. Architecting aspect-oriented software designs basically implies on using new composition mechanisms that allow for quantifying otherwise crosscutting concerns, such as error handling and distribution. In fact, the core of existing aspect-oriented extensions [11] to existing architecture description languages (ADLs) reside on the explicit support for expressing quantifications of architecturally-relevant crosscutting behaviours [1, 12]. Aspect-oriented ADLs [11, 16, 18, 19] foster fine-grained composition specifications through pointcut specifications. Architectural pointcut descriptions allow for the flexible binding of “aspectual components” to multiple points – the join points – in a behavioural architecture description, not previously supported by conventional ADLs.

However, there is no empirical understanding on the positive and negative effects of aspect-oriented architectures in promoting architectural design stability. In particular, there is a pressing need for analyzing the extent of the benefits and drawbacks of the enriched aspect-oriented composition capabilities in the presence of architecturally-relevant changes. However, it has been empirically observed that design stability is directly dependent on the underlying composition mechanisms [3, 13, 25]. 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 [3].

The difficulty of analyzing the stability of aspect-oriented architecture designs stems from the fact that aspectual compositions can significantly take different forms according to the dominant architectural decompositions being applied. Join point models are sensitive to the semantics of the architectural styles being instantiated and composed in the context of a particular application. For instance, the join points in a layered 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 empirical study where we have analyzed the stability of aspect-oriented software architectures designed for a multi-agent system (MAS) (Section 2). We have compared the influence of non-aspectual and aspectual decompositions (Sections 3 and 5) on the architectural stability of crosscutting concerns (Section 4) in such a MAS architecture. These alternative designs of the same system were the target of typical architectural changes so that we could contrast ripple effects being detected in both conventional bindings and pointcuts (Section 6). We have used a conventional suite of architectural stability metrics in order to search for evidence with respect the following research questions: to what extent ripple effects increase or decrease when aspect-oriented
compositions are used? How different types of crosscutting concerns, architectural styles, and join point models influence positively and negatively the architecture stability? Finally, we also discuss related work and provide a correlation of our findings with a previous implementation-level stability study [13] we have performed (Section 7).

2. Study Settings

This Section describes the configuration of our study including the choice of the case study and our analytical methodology (Section 2.1), as well as the description of the target system used in our investigation (Section 2.2).

2.1. Methodological Procedures

The Case Study. The first major decision concerning our investigation was the selection of the case study. The selected system was the Conference Management System (CMS) [5], 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 [24, 14]. These systems realize heterogeneous architectural designs, following conventional and MAS-specific architectural styles. Second, the MAS domain entails complex software architectures with rich categories of crosscutting concerns [12]. 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 [12, 10]. Also, the CMS is the most widely-used benchmark in the MAS community [5].

Heterogeneous Architecture Designs. Third, each architectural design decision for existing CMS versions has been extensively discussed and evolved over time in a controlled manner. The first CMS version [12, 10] was a three-tier client-server that employed a blackboard subsystem to manage conference information. The original CMS has progressively evolved in accordance to different MAS-specific hybrid styles, namely reflective blackboard, reactive coordination, and stigmergic coordination (Section 3). Fourth, the evolved designs were developed with the goal of satisfying new emerging requirements and addressing 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 (e.g. [12, 10, 15]).

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 version, (iii) the third phase involved the discussion on different styles used afterwards to implement alternative CMS architectures, and (iv) the fourth phase analyzed the similarities and divergences of the crosscutting concerns present in such alternative designs and the ones observed in the second phase. Before we describe each of the four phases, the next subsection provides a brief overview of the CMS system.

2.2. Conference Management System

The Conference Management System (CMS) is an application that supports the management of scientific conferences [5]. 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 Styles for the CMS

This Section depicts the structure of the Conference Management System (CMS), presenting component-and-connector (C&C) views based on the blackboard and the reflective blackboard styles. Two other stylistic variations (reactive coordination and stigmergic coordination) are briefly commented.
3.1. Blackboard Style

The basic architecture of a blackboard system consists of a blackboard, a collection of knowledge sources and, eventually, an explicit control component [17, 2, 21]. The Blackboard_CMS component is the central data storage structure where submitted papers, information about the authors, review templates, list of accepted papers, etc., can be inspected or updated by knowledge sources. PC-chair, Reviewers, Authors, and PC-members are the knowledge sources (KSs). They are agents that solve specific problems related to submissions, paper assignments, reviews, etc. Two connectors, inspection and update, describe the interaction between the KSs and the Blackboard_CMS component. The Control_CMS component is responsible for managing the course of the complex workflow that characterizes the CMS. Control_CMS needs to be informed about Blackboard_CMS state changes regularly. This interaction follows a publish-subscribe protocol, described by the event-bus connector. Control_CMS interacts with each KS (i) to query whether they can perform some action that contributes to the overall solution, and select one KS, and (ii) to activate (or trigger) the execution of the selected KS. These interactions are described by the connectors query and activation, respectively. Since Control_CMS is a central element that selects and activates KSs, CMS agents do not need to have complex cognitive skills and can be simple reactive agents.

3.2. Reflective Blackboard Style

The reflective blackboard style [22] combines the blackboard and reflection styles [2]. Communication is centralized on the blackboard and Control_CMS is transparently inserted in the desired points of inter-agent communications by using reflective features. The reflection pattern provides specific architectural elements, the Object_CMSs and the Object_CMS Protocol (MOP_CMS), for dynamically changing the structure and behaviour of the target blackboard component. Elements of the blackboard (e.g., tuples) are associated via MOP_CMS with Object_CMSs defined in the Control_CMS components; they are used to implement, for instance, complex coordination protocols.

Control_CMS has a sophisticated internal structure (Figure 1) that includes a MOP_CMS component and possibly several Object_CMS components. Rule enforcement on the conference management workflow (e.g., bidding and automatic matching based on keywords in paper assignment phase) is encapsulated in separate Object_CMSs. This means that the interaction among Blackboard_CMS, PC-chair and Reviewer can be simplified.

The MOP_CMS monitors the events occurring inside Blackboard_CMS, and the events flow from Blackboard_CMS to MOP_CMS through the reification connector. Reification means that structural and behavioral information of Blackboard_CMS is exposed to Control_CMS. Typically, a reification connector is defined for linking the meta-level and the KSs, depending on which kind of adaptation is necessary in certain applications; in the specific case of CMS, reification connects only the Blackboard_CMS component. If an event is associated with one or more Object_CMSs, these elements are put in execution by MOP_CMS by means of the object call connector. The execution of the Object_CMS(s) can modify the internal state of the Blackboard_CMS component; therefore, these modifications flow from Object_CMS(s) to Blackboard by means of the state changing connector. Data are represented by tuples, stored in the Blackboard_CMS. In this style, Control_CMS is not connected to PC-chair and Reviewer. This implies a greater autonomy for PC-chair and Reviewer.

3.3. Other Styles

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 pattern is that the behaviour 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 the TuCSoN [24] – an infrastructure for the agent coordination – where the tuple centre rep-
represents the programmable coordination medium. The architectural description of the CMS using the reactive coordination style can be found in [7].

**Stigmergic Coordination Style.** The stigmergic coordination style is another evolution of the blackboard style, particularly suited for self-organizing systems. Stigmergy [14] 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 is adopted by the TOTA infrastructure [14] that enables stigmergic interactions among distributed agents. The CMS description using the stigmergic coordination style can be found in [7].

### 4. Architectural Crosscutting Concerns

An aspect-oriented architecture represents concerns that cannot be localized in individual components using other forms of architectural decompositions. These concerns are crosscutting and represented by architectural aspects. This section overviews the architectural crosscutting concerns encountered in the CMS architectures and used in our analysis. In order to determine which concerns are or are not crosscutting we have used the following strategies [12]: (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, we have identified three different crosscutting concerns: coordination, error handling, and security. Due to space limitation, we will focus our discussion on coordination (Section 4.1) and error handling (Section 4.2). We have used a metrics suite to support our identification of tangling, scattering and interface bloat [20] and the measures can be found at [7].

#### 4.1. 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 \textit{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 \textit{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 \textit{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 \textit{PC-chair} interfaces, the information generated for this protocol, pieces of the \textit{Control_CMS} component and its interfaces (possibly because the \textit{Control_CMS} must trigger, for example, different \textit{Reviewer} agents or a different number of Reviewers) and also the activations of the agents. The \textit{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 agents involved in the system.

Coordination is not a crosscutting concern in the reflective blackboard and reactive coordination styles. In the reflective blackboard style, the coordination is realized by means of \textit{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 paper partitioning and assignment example, the protocol change only requires a modification of the specific \textit{Object_CMS} that manages the protocol for the reflective blackboard style, and a modification on the reactions of the reactive coordination style.

#### 4.2. Error Handling

Error handling is widely recognized as a global design issue and has been extensively referred in the literature as a classical crosscutting concern in systems following different kinds of architecture decompositions [23, 8]. In the blackboard style, error handling is a crosscutting concern as exceptional conditions must be propagated from the blackboard to several KSs. Similarly, error handling is also crosscutting concern in all the other styles as error handling need to involve several different entities, for instance, when communication exceptions occur during collaborations between
knowledge sources. In fact, in the Reflective blackboard and reactive coordination decompositions of the CMS system, both agents and coordination medium are also involved in error treatments. Each component should be equipped with specific code in order to react to system’s failures.

5. Aspect Composition

An architectural style provides distinct component types and connector types, with style-specific semantics. The behavior of the components and connectors employed by a style can, therefore, act as a basis to define an high-level style-specific, semantics-based join point model for software architecture models expressed in that style. These style-specific join point models can be exposed and used in quantified expressions, so that, both syntax-based composition and style-based composition can be supported at the architectural level.

Section 5.1 presents a join point model for each architectural style described in Section 3. The kinds of join points for each $jpm$ are listed in the rows of a table, together with a description of the places they manifest. We depart from some documented uses of each style [17, 24, 14] or at least one good source of style guide [21, 6, 22] publicly available. Nevertheless, the choice of a representative set of style-specific join points is still a very subjective task. Section 5.2 presents examples of architecture-level pointcut designators. We illustrate syntax-based pointcuts and style-based pointcuts.

5.1. Join Point Models

**Blackboard.** Figure 2 presents a joint point model for the blackboard style. Although we may find other good candidates for joint points in this style, we have selected six different kinds of joint points. The inspect blackboard join point kind describes the points where data is read from the blackboard, in the context of an interaction with some KS. The update blackboard join point kind describes the points in which some KS writes data in the blackboard. The signal status change joint point kind 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. The query join point kind denotes the points where the Control makes a query over each KS to determine if they are potential contributors. Based on the query results, the Control selects a KS to activate. The select source join point kind denotes the points where the Control selects a KS. The activate source join point kind describes the points where the Control puts in execution the more suitable KS.

**Reflective Blackboard.** This style combines two styles through the union of their design vocabularies, and conjoining their constraints. The hybrid join point model is defined by the resulting types defined by the conjunction. Figure 3 presents the join point models for reflection, blackboard and reflective blackboard styles.

The reflection style presents three different join points: (i) *reification*: the base level notifies its status changed; (ii) *Object_CMS call*: the MOP_CMS puts in execution the more suitable Object_CMS and (iii) *adaptation*: the Object_CMS changes the state of base level. Two join points are aligned and unified: *signal change status* (blackboard style) and the *reification* join point (reflection style). This unification allows the MOP_CMS component to capture the events generated by the Blackboard_CMS. The select next source and the invoke source join points from the blackboard style are not necessary. The other join points in the third column are the union of the remaining join points of reflection and blackboard styles.

**Other Styles.** The reactive coordination style is a stylistic specialization of the blackboard style and may strengthen the constraints, provide more specialized versions of some of the element types, or add or remove some element types, with respect to the base style. For example, the Tuple Centre is a specialized version of the blackboard component, and is defined as the composition of the Data_TS and Reaction_TS tuple spaces. This begs the question as to how the new, derived join point model differs from the join point model of the original, base style. Our case study helped us to identify three different situations: (i) base joint points are reused in the derived style, (ii) new kinds of join points are introduced by the derived style, (iii) two joint points overlap and must

---

<table>
<thead>
<tr>
<th>Join point</th>
<th>Where (points in execution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inspect blackboard, update blackboard</td>
<td>KS, Blackboard</td>
</tr>
<tr>
<td>signal status change</td>
<td>Blackboard, Control</td>
</tr>
<tr>
<td>query source, select source, activate source</td>
<td>Control, KS</td>
</tr>
</tbody>
</table>

**Figure 2. Join point model in the Blackboard style**

<table>
<thead>
<tr>
<th>Reflection Join Points</th>
<th>Blackboard Join points</th>
<th>Conjunction Join points</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>reification</em></td>
<td>inspect blackboard, update blackboard</td>
<td>update Blackboard, inspect blackboard</td>
</tr>
<tr>
<td><em>Object_CMS call</em></td>
<td><em>signal status change</em></td>
<td><em>reification</em></td>
</tr>
<tr>
<td><em>adaptation</em></td>
<td><em>query source, select source, activate source</em></td>
<td><em>adaptation</em></td>
</tr>
</tbody>
</table>

**Figure 3. Join point model in the Reflective Blackboard style**
be reconciled in the derived join point model, and (iv) base join points are not used or necessary in the derived style.

This style introduces three new kinds of join points: (i) data modification: the Reaction_TS component changes the tuples stored inside the Data_TS component; (ii) behavior changing: an agent changes the contents of the Reaction_TS; and (iii) notify status change: the Data_TS component notifies its status changes. The notify status change join point kind overlaps with signal status change (from the Blackboard style); they should be reconciled. The query source, select source and the activate source join points are not necessary in the Reactive Coordination style because the information between the Reactive_TS and the agent are exchanged by means of data stored in the Data_TS component.

The proposed join point model for the stigmergic coordination style can be found in [7].

5.2. Architecture-level Pointcuts

Architecture-level pointcuts are mechanisms for specifying sets of join points that are relevant to the modularization of certain crosscutting concerns at the architectural level [4].

<table>
<thead>
<tr>
<th>Style</th>
<th>Pointcut Designators</th>
<th>Join Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackboard</td>
<td>change coordination status</td>
<td>update blackboard status</td>
</tr>
<tr>
<td></td>
<td>read coordination info</td>
<td>inspect blackboard</td>
</tr>
<tr>
<td></td>
<td>invoke coordination source</td>
<td>select source activate source</td>
</tr>
<tr>
<td>Reflective Blackboard</td>
<td>reify coordination</td>
<td>update blackboard reification</td>
</tr>
<tr>
<td></td>
<td>read coordination info</td>
<td>inspect blackboard</td>
</tr>
<tr>
<td></td>
<td>change coordination data</td>
<td>Object_CMS call adaptation update blackboard</td>
</tr>
</tbody>
</table>

**Figure 4. Style-based Pointcuts for Coordination**

**Style-based pointcuts.** Style-based pointcut designators name expressions that specify join points based on style semantics. Table 4 presents a set of style-specific pointcut designators to be used by the Coordination crosscutting concern. They denote a representative set of join points of interest for the Blackboard and Reflective Blackboard styles. For instance, the change coordination status pointcut, associated with the Blackboard style (Table 4), matches join points where a knowledge source updates the Blackboard or the Blackboard notifies its status changes by means of a particular event.

\* Blackboard Style Elements *\*

Family BlackboardStyle = {
  Component Type Blackboard_CMS {...};
  Component Type Reviewer {Port storeData; ... };
  Component Type PC_Chair { Port storeData; ... }
  Connector Type event-bus {...};
  ...
}

System CMS: BlackboardStyle {
  Component Chair: PC_Chair;
  Component Rev1, Rev2, Rev3: Reviewer;
  ...
}

\* Coordination Crosscutting Concern *\*

Component Coordinator = {
  Port changeCoordinationStatus;
  Port readCoordinationInfo;
  Port invokeCoordinationSource;
}

Connector Coord = {
  baseRole sink;
  crosscuttingRole source;
  glue source after sink;
}

\* Pointcut description - aspectral bindings *\*

Attachments {
  Coordinator.readCoordinationInfo to Coord.source;
  Coord.sink to *.storeData;
  ...
}

**Figure 5. Coordination Aspect in the Blackboard Style**

Due to space limitations, we provide here just a few examples of pointcut designators for Error Handling and Security. Pointcut designators for Error Handling specify 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 communication exception pointcut matches points of possible failures in communication.

Pointcut designators for Security specify points where security is an issue of concern: the data secure storing pointcut matches points where 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.

**Syntax-based pointcuts.** Syntax-based pointcuts rely on naming conventions and other syntax constructs rather than in high-level abstractions. Figure 5 presents an example of a syntax-based pointcut used for the architectural composition of the Coordination crosscutting concern, represented by the Coordinator component. The Coord connector is
used in the aspectual composition to describe the interaction between Coordination and the components that will be coordinated (PC_Chair and Reviewers). The glue clause specifies that the crosscutting concern affects the base code after reaching the join points specified by the pointcut. The Attachments block describes the aspectual bindings, including one syntax-based pointcut: readCoordinationInfo of the Coordinator component affects the storeData operation of all components (represented by the * wildcard).

The example is written in AspectualACME [11], an ADL that extends ACME by defining an extension of the connector element with a new kind of interface, specific roles and a glue clause that describes the details of the connection. It also provides a quantification mechanism in the attachments part. This is the place where structural join points are identified in ACME; thus it is possible to use wildcards in order to denote names or part of names of components and their ports. More information can be found in [1, 11].

6. Measurement and Analysis

This Section presents both the measurement procedures and data analysis. We have used classical evaluation indicators and procedures to analyze the stability of all the non-AO and AO architectures (Section 2.1) under assessment, as described in the following subsections.

6.1. 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 by 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. Higher the number of changes, higher is the probability of architectural ripple effects manifesting. In addition, the reason was to make a paradigm-independent measurement 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.

6.2. 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 to 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 architecture evolution 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 managing of error handling strategies, (S6) introduction of new organizational rules or 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.

6.3. Evidences of Architectural Instabilities

This Section reports the experimental results of the mea-
Error handling

Security

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, 8; they consist three out of five scenarios where non-crosscutting concerns are the main change target (Figure 6). It is also important to highlight that the superiority of non-AO architectures was independent of stylistic choices (Section 3), join models (Section 5.1), and types of changes (Section 6.2) - whether coarse-grained or fine-grained. This finding somehow reinforces results observed in an implementation-level empirical study that we have recently conducted [13]. However, our stability analysis in the previous study has not being carried out at the architectural level and 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.

6.4. Composition-Level Instabilities

Conventional Bindings vs. Pointcuts. Figures 7, 8 and 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 (Figure 7). Major instabilities of coordination-specific bindings were observed in three scenarios (4, 6, 8) against only one major coordination pointcut instability case (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 (Figures 8 and 9).

**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 Figure 9. However, the non-AO version was still more superior in this particular case. The reason is that aspect-oriented software architectures has 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 (Figure 9).

7. **Related Work**

The literature on architectural aspects provides some examples of classical aspects at architecture design, such as error handling, persistence, distribution, and coordination [11, 16, 18, 19]. 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 aspected. 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 [13]. 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 6.4, the kind of decomposition supported by different conventional styles may 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 (Figures 7, 8 and 9). The stability of the aspectual compositions (i.e. 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. Further studies should be carried out to produce a broader catalogue of style-specific crosscutting concerns for supporting software architects.

8. Conclusions and Future Work

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 4) typical for this system. One of the main outcomes was that the AO architectures tend to have a more stable design particularly when a change targets a crosscutting concern; the AO architectures tended to require less invasive changes. However, they did not scale much when changes were made in non-crosscutting concerns. Ripple effects were often observed in pointcut specifications. One of the potential reasons is the syntax-based nature of pointcut languages; however, more systematic studies need to be carried out to confirm this. Future work will also be devoted to assess aspect-oriented architectures in the light of different architectural attributes other than stability.

References