Leveraging Aspect-Connectors to Improve Stability of Product-Line Variabilities

Marcelo Dias #1, Leonardo Tizzei #2, Cecília Rubira #3, Alessandro Garcia #4, Jaejoon Lee †5

# Institute of Computing, University of Campinas
Campinas, Brazil
1marcelo.dias@students.ic.unicamp.br
2tizzei@ic.unicamp.br
3cmrubira@ic.unicamp.br
4Informatics Department, PUC-Rio
Rio de Janeiro, Brazil
5alessandro.garcia@puc-rj.br
† Computing Department, Lancaster University
Lancaster, UK
5j.lee@comp.lancs.ac.uk

Abstract—One of the design goals of Product Line Architectures (PLAs) is to remain stable while accommodating changes of stakeholder’s requirements. However, the stability of a PLA is largely dependent on how modularized are the decisions crosscutting multiple architectural variation points. Their scattered implementation often leads to a number of side effects, such as architecture-wide modifications. This paper proposes a novel component model to encapsulate architectural variation points inside aspect-connectors, called Connector-VPs. Our component model addresses limitations of emerging aspect-oriented models, such as XPIs, which do not allow a modular implementation of crosscutting variability decisions in a PLA. The role of a Connector-VP is both binding aspectual-level components to base-level ones, and isolating crosscutting decisions at architectural variation points. We have evaluated the PLA stability designed using our solution in the presence of heterogeneous evolutionary scenarios in the context of component-based PLAs. The results show that our solution tends to promote superior PLA resilience on these scenarios.

Index Terms—Component-based software development; Aspect-Oriented Programming; Software Product Lines; Software Architecture; Architectural Variability;

I. INTRODUCTION

Software product line (SPL) engineering aims at improving development efficiency for families of software systems in a given domain [7]. This concept promotes large-scale reuse through a Product Line Architecture (PLA) that is common to a variety of similar products in terms of their architectural elements. The combination of SPL and Component-based Development (CBD) is a well-known technique to rapidly and efficiently derive products from a set of reusable assets [3]. In the CBD, software systems are developed composing interoperable and reusable blocks called software components [24]. A component-based PLA fosters explicit representation of component specification and contributes to reduce coupling and increase cohesion, thereby improving SPL modularity and evolvability [4].

In the context of component-based PLAs, an architectural variation point is a place on the PLA where decisions are made to derive different products [25]. Architectural variation points are associated with non-kernel features (optional or alternative ones). Evolution scenarios involving non-kernel feature cause changes in architectural variation points. These scenarios usually involve inclusion of non-kernel features and/or changing kernel features into non-kernel ones. In this context, it is important for organizations to achieve design stable PLAs and achieving a controlled evolution of architectural variation points lies at the heart of it. A stable PLA means that it can endure evolutionary changes by sustaining its modularity properties.

One of the modern approaches to support enhanced modularity is Aspect-Oriented Programming (AOP) [15]. Some works advocate that aspects can be used to facilitate PLA evolution by using aspects to modularize PLA variabilities [2], [8], [21]. Aspectual-level components can be used to implement crosscutting non-kernel features, by defining pointcuts that advise base-level components of a PLA. However, recent studies [20], [8] have identified that the use of conventional AOP mechanisms can lead to PLA instabilities in specific evolution scenarios. The reason is twofold: (i) the use of conventional AOP leads to a high coupling between aspectual and base components of a PLA, thereby generating pointcut instabilities, and (ii) many decisions on architectural variation points are still difficult to modularize with AOP.

Emerging AOP approaches, such as XPIs [13], address the first problem but not the second one. XPIs aim at decreasing the tight dependency caused by AOP. This approach employs explicit abstract interfaces, called Crosscut Programming Interfaces (XPIs), to decouple aspects from the base code intercepted by them. A XPI specifies sets of base code points where aspects should be plugged in.

In the context of aspect-oriented PLAs, the employment of XPIs can decouple aspectual-level components from the core architecture, thus improving the PLA modularity. However,
the combined use of components, aspects, and XPIs to design PLAs does not suffice to address the scattered implementation of inter-related decisions at architectural variation points. Components usually have to include in their implementation some additional code in order to support all possible variability decisions of the architectural variation points. In this case, the support implementation of such variability decisions is usually scatteredly implemented over the components. Hence, evolution scenarios on architectural variation points would also imply in changes traversing the related components, thereby decreasing the PLA stability.

We propose the concept of aspect-connectors for improving architecture variability, called Connector-VPs. While Connector-VPs are employed to bind aspectual components to XPIs of base-level components, they also encapsulating the support implementation for variability decisions from components. The goal of our solution is to design stable PLAs by encapsulating the implementation of such otherwise scattered decisions inside Connector-VPs. The employment of Connector-VPs avoids changes in architectural variation points from being propagated to the components and their interfaces.

This paper also presents a comparative study to evaluate the positive and negative impact of using Connector-VPs to design component-based PLAs. The objective is to quantitatively and qualitatively assess to what extent the use of aspect-connector for architecture variability promotes PLA design stability in the presence of various types of change. In our investigation, we have adopted a component implementation model called COSMOS* [12]. We have focused on eight releases of a SPL called MobileMedia [16]. Two alternative implementations of MobileMedia product line were involved and compared in our study: (i) one using COSMOS* component model; and (ii) one using COSMOS* combined with the use of Connector-VPs. It is worth mentioning, that in both implementations, XPI approach was employed to decouple aspectual-level components from base-level ones. We have employed conventional metrics for change impact [23] and modularity [26] for evaluating the PLA stability of the two implementations. The results pointed out that the application of Connector-VPs tends to promote superior PLA resilience than the other approach involved in this study.

The paper is organized as follows: Section II presents some necessary concepts to understand the rest of this paper. Section III presents the novel COSMOS*-VP implementation model, which applies our proposed solution. Section IV describes the empirical study, which provides data for the change impact analysis in Section V and for the modularity analysis in Section VI. Section VII presents some works related to this one, and in Section VIII we draw the conclusions, list some limitations of our study, and plan the future work.

II. BACKGROUND

A. COSMOS* Component Implementation Model

According to Szyperski [24], a software component is an unit of modularity with explicit provided and required interfaces. It can also be deployed independently and is subject to composition by third parties. The COSMOS* implementation model is representative of component models because it has all these characteristics. The main benefits of COSMOS* is twofold. First, COSMOS* explicitly represents architectural units, such as components, connectors and configuration, thus providing traceability between the software architecture and the respective source code. Second, COSMOS* is considered a platform-independent model, as it is based on a set of design patterns.

COSMOS* defines five sub-models, which address different perspectives of component-based systems: (i) the specification model specifies the components; (ii) the implementation model explicitly separates the provided and required interfaces from the implementation; (iii) the connector model specifies the link between components using connectors; (iv) composite components model specifies high-granularity components; and (v) system model defines a software component which can be straightforwardly executed.

Figure 1 (a) shows an architectural view of a COSMOS* component called FavouritesMgr and Figure 1 (b) shows the detailed design of the same COSMOS* component. COSMOS* components are internally divided in specification (spec package) and implementation (impl package). The specification is the external view of the component, which is also sub-divided in two parts, one that specifies the provided services (spec.prov package) and the other makes dependencies explicit (spec.req package). For instance, IManager and IFavourites interfaces are provided interfaces and IPersistence is a required interface. The impl package has three mandatory classes: (i) a ComponentFactory class, responsible for instantiating the component; (ii) a Facade class that realizes provided interfaces, following the Façade design pattern [10]; and (iii) a Manager class that realizes IManager interface and provides meta-information about the component. It is possible to have an optional class called ObjectFactory, which aims at reducing coupling between implementation classes within the component. FavControl, which supports the implementation of the IFavourites interface and requires services from other components via IPersistence interface, is an example of an auxiliary class.

![Fig. 1. (a) An architectural view of a COSMOS* component; (b) A detailed design of the COSMOS* component.](image-url)
approaches decouple the specification of the base code places where aspects are plugged from the aspects itself, in order to improve system modularity. The XPI approach [13] is one of the proposed approaches. While it employs Crosscut Programming Interfaces (XPIs) to specify base code places separately from aspects, it neither limits the use of existing aspect-oriented mechanisms nor require new ones.

In the context of the combined use of components and aspects, we call by aspectual-level components those which implement crosscutting features, and are plugged into base-level components in order to provide their features. Aspectual-level components support the separation of features (in this paper, features are considered equivalent to concerns), and, consequently, increases the system modularity. Applying the XPIs approach to the combined use of components and aspects, the aspects of an aspectual-level component and the XPIs of a base-level component are separated, thus improving system modularity [17].

COSMOS\* can combine aspects and the concepts of XPI to the context of components in order to implement aspectual-level components. These components use aspects to intercept base-level components in order to provide their functionality. The main goal is to take advantage of the benefits of the three approaches involved, thus increasing the modularity of component-based architectures. Some characteristics of this combination are similar to other aspectual component models, such as separation between aspectual-level and base-level components, as DyMAC [17] and FAC [22], and the employment of aspect-connectors, as FAC. However, the combined use of COSMOS\* and XPI, compared to these approaches, presents some advantages. For example, it does not need new programming mechanisms as the FAC, and its connectors can be used not just to encapsulate non-functional concerns as in DyMAC (see Section VII).

COSMOS\* can combine aspects and the concepts of XPI to the context of components in order to implement aspectual-level components. These components use aspects to intercept base-level components in order to provide their functionality. The main goal is to take advantage of the benefits of the three approaches involved, thus increasing the modularity of component-based architectures. Some characteristics of this combination are similar to other aspectual component models, such as separation between aspectual-level and base-level components, as DyMAC [17] and FAC [22], and the employment of aspect-connectors, as FAC. However, the combined use of COSMOS\* and XPI, compared to these approaches, presents some advantages. For example, it does not need new programming mechanisms as the FAC, and its connectors can be used not just to encapsulate non-functional concerns as in DyMAC (see Section VII).

The encapsulation of support implementations for architectural variation points should be encapsulated within architectural connectors, thus allowing components and architectural variation points to be changed independently. Without such architectural connector, a change in these implementations may affect several components of a PLA. Furthermore, a component that implements variation points of a certain PLA might hinder its reusability in other PLAs, since it holds details of the PLA.

The support implementations for architectural variation points should be encapsulated within architectural connectors, thus allowing components and architectural variation points to be changed independently. Without such architectural connector, a change in these implementations may affect several components of a PLA. Furthermore, a component that implements variation points of a certain PLA might hinder its reusability in other PLAs, since it holds details of the PLA.

C. Evolving Component-based PLAs with Aspects

Some works advocate that aspects can be used to facilitate PLA evolution by using aspects to modularize PLA variabilities [2], [8], [21]. In that case, aspectual-level components are used to implement non-kernel features. The employment of XPIs can decouple aspectual-level components from base-level ones, thus improving the PLA modularity.

However, the XPIs concept does not suffice to solve the scattering of architectural variation points over architectural elements of a component-based PLA. The scattering is created by implementing an architectural variation point across a set of components. That is usually necessary in order to support all variability decisions related to the architectural variation point.

In an illustrative example, suppose that the PLA of a mobile phone SPL has to handle different types of media, such as music and photo. From the assets of this SPL, it is possible to derive mobile phones which handle (i) music, (ii) photo, or (iii) both music and photo. A possible way to implement this PLA would be creating one component for media, one for music, and other for photo. The MediaMgr component would handle operations that are common to both photo and music (e.g. create, delete) and MusicMgr and PhotoMgr components would handle operations specific to its media type, respectively. Since at least one media type must be chosen, MediaMgr is a kernel component implemented on the base level. MusicMgr and PhotoMgr components are implemented on the aspect level and intercept MediaMgr component in order to provide Music and Photo features, respectively. The MediaMgr component is not aware of the particularities of each type of media, since it deals with general operations. Thus, both PhotoMgr and MusicMgr components must check whether the data originated from MediaMgr component is of the appropriate type. This data checking is only necessary when there are more than one type of media, and it represents a support implementation for the decisions of the architectural variation point which is scattered on two components, namely PhotoMgr and MusicMgr. Hence, evolution scenarios related to this architectural variation point would imply in changes in at least two components.

The support implementations for architectural variation points should be encapsulated within architectural connectors, thus allowing components and architectural variation points to be changed independently. Without such architectural connector, a change in these implementations may affect several components of a PLA. Furthermore, a component that implements variation points of a certain PLA might hinder its reusability in other PLAs, since it holds details of the PLA.

The encapsulation of support implementations for architectural variation points in specific architectural connectors also helps to isolate the implementation of features from the implementation of variation points. Thus, changing one should not affect the other. For instance, when an optional

![Figure 2: Detailed design of an aspectual-level component.](image-url)
feature becomes an alternative feature, only the element where the decisions related to this feature are supported should be modified, that is, the specific architectural connector which implements the architectural variation point.

### III. COSMOS*-VP EXTENSION

The COSMOS*-VP model extends COSMOS* model by providing guidelines to specify aspect-connectors for architectural variation points, called Connector-VPs and materialize them into source code. Connector-VPs avoid architectural variation points to be scatteredly implemented over several PLA architectural elements. Once the architectural variation points are moved from components to Connector-VPs, changes in architectural variation points are avoided from being propagated to the components and their interfaces, thus facilitating PLA evolution.

**TABLE I**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delegation Interfaces</td>
<td>Aspects used to extend Abstract Aspects of aspectual-level components.</td>
</tr>
<tr>
<td>Interception Interfaces</td>
<td>Aspects used to advise base-level components in order to provide the non-kernel features of the aspectual-level components.</td>
</tr>
<tr>
<td>Adapter</td>
<td>A class which implement an Adapter design pattern [10] between the Interception Interfaces and the Delegation Interfaces of a Connector-VP.</td>
</tr>
</tbody>
</table>

Table I summarises the elements introduced by COSMOS*-VP to the COSMOS* connector model, in order to allow the specification and implementation of Connector-VPs. A Connector-VP provides mechanisms to mediate the binding of the Abstract Aspects of aspectual-level components to the XPIs of base-level components. This binding is necessary in order to provide the non-kernel features of aspectual-level components to base-level components. Connector-VPs bind the components by using Delegation Interfaces to extend the Abstract Aspects, and using Interception Interfaces to advise the base-level components XPIs. The use of such interfaces separates the Connector-VPs specification from its implementation, avoiding a Connector-VP from being an instable hard-wired connector between Abstract Aspects and XPIs.

The Adapter design pattern is used to mediate the connection between Delegation Interfaces and Interception Interfaces (see Table I). While the Adapter mediates the connection, it provides a place to implement the necessary support for variability decisions of the architectural variation point. Hence, the employment of the Adapter helps a Connector-VP to encapsulate an architectural variation point. That is, without a mediator, as an Adapter of a ConnectorVP, the base-level components are directly advised by aspectual-level ones, thus, the implementation that supports the variability decisions have to be scatteredly implemented inside components. Another benefit of the Adapter is that some mismatches between Abstract Aspects and XPIs, connected by a Connector-VP can be adapted.

It is worth mentioning that all aspectual-level components connections at a specific architectural variation point must be mediated by only one Connector-VP. That allows the Connector-VP to encapsulate the support implementation for all possible decisions of the point.

**Fig. 3.** (a) An architectural view of a Connector-VP; (b) A detailed design of a Connector-VP.

Figure 3 (a) illustrates the architectural view of a Connector-VP, namely MediaFavVP, which mediates the connection between the aspectual-level component FavouritesMgr to the base-level component MediaMgr. The Delegation Interface DIOptional extends the Abstract Aspect of FavouritesMgr, and the Interception Interface IIOptional is used to intercept MediaMgr, using XPIMedia, in order to provide the optional feature of FavouritesMgr.

Figure 3 (b) presents how the MediaFavVP can be implemented using AspectJ. In the figure, we have omitted some required packages of the components, for the sake of clarity. To provide the behaviour extended from FavouritesConcern to the MediaMgr an adapter, called InterAdapter, is created between the IIOptional and DIOptional interfaces. InterAdapter must have one method for each one of the advises extended by DIOptional. And, each advice of DIOptional, which is a realization of one Abstract advice of FavouritesConcern, is implemented in order to intercept its correspondent method of the InterAdapter. IIOptional, during the execution of its advises, which intercept MediaMgr, will provide the optional feature Favourites to MediaMgr calling the InterAdapter methods.

This mechanism permits support decisions implementations to be put between the components inside the MediaFavVP. These implementations can be created in IIOptional or in InterAdapter, thus, avoiding them to be scattered over the components. These implementations can determine, for example, under which conditions the optional feature will be provided.

It is important to notice that although the complexity of the Connector-VPs, their elements are very simple, and most of them can be semi-automatically generated.
based on the XPIs and Abstract Aspects connected by the Connector-VP.

![Diagram](https://via.placeholder.com/150)

**Fig. 4.** (a) MobileMedia COSMOS*-XPI PLA; (b) MobileMedia COSMOS*-VP PLA.

Figure 4 illustrates how an architectural variation point can be isolated inside a Connector-VP. The Figure 4 (a) shows a slice of MobileMedia PLA, implemented using COSMOS* combined with XPIs, containing an architectural variation point related to the choice of two alternative components, namely PhotoMgr and MusicMgr. Alternative components are aspectual-level components which implement the alternative features. MusicMgr and PhotoMgr components are implemented on the aspect-level and intercept MediaMgr component in order to provide Music and Photo features, respectively. The MediaMgr is a base-level component and it deals with general operations of medias support. Both PhotoMgr and MusicMgr component must check whether the data received advising MediaMgr component is of appropriate type. This data checking is not necessary when just one of the components is chosen, and it represents a support implementation for the possible decisions of the architectural variation point, which is scattered over the components.

The Figure 4 (b) shows the same slice of the PLA implemented using COSMOS*-VP. A Connector-VP, called MediasVP, mediates the binding of the alternative components to the MediaMgr. The data checking is implemented into the MediasVP adapter, and depending on the checking result, the appropriated alternative component will provide its feature. Thereby, the architectural variation point was encapsulated only inside the MediasVP.

### IV. Empirical Settings

This section presents the empirical settings used to assess the use of COSMOS*-VP to design stable PLAs.

#### A. Target software product line

In order to exemplify and evaluate our solution, we present a software application, called MobileMedia [8], which is a SPL for mobile applications that manipulates photo, music, and video on mobile devices, such as mobile phones. The system uses various technologies based on the Java ME platform, such as SMS, WMA and MMAPI. It has two implementations with the same functionalities but implemented with different approaches: one uses AO programming and has approximately 12 KLOC and the other uses only OO programming and has 11 KLOC. MobileMedia endured seven evolution scenarios, which led to eight releases. It is possible to derive 200 products from the last release. The scenarios comprise different types of changes involving kernel, optional, and alternative features, as well as non-functional concerns. The purpose of these changes is to exercise the implementation boundaries and, thus, assess the design stability of the PLA. Table II summarises the evolution scenarios in MobileMedia.

#### B. Study Definition and Execution

The objective of this comparative study is to assess quantitatively and qualitatively to what extent the specification and implementation of Connector-VPs, by using COSMOS*-VP model, promote PLA design stability in the presence of various types of changes. In this study, we compare two models to implement PLAs: (i) COSMOS* combined with AOP and XPIs, called COSMOS*-XPI; and (ii) using COSMOS*-VP. It is worth mentioning that COSMOS*-VP also employs AOP and XPIs to decouple components from aspectual-level to those of base-level of a PLA. In our study, we have used change impact and modularity metrics in order to evaluate PLAs stability.

The original AO implementation of the MobileMedia was the input for our empirical study. In order to execute the comparative study, we have performed the following steps:

1. **Step 1.** Refactor the first release (R1) of the original AO implementation to COSMOS*-XPI and COSMOS*-VP implementations.
2. **Step 2.** Evolve the refactored COSMOS*-XPI and COSMOS*-VP implementations according to the evolution scenarios described in Table II.
3. **Step 3.** Collect change impact and modularity metrics for eight COSMOS*-XPI and COSMOS*-VP releases;
4. **Step 4.** Compare the results of COSMOS*-VP against COSMOS*-XPI implementation.

As a result of Step 1 execution, two new implementations were created, named COSMOS*-XPI and COSMOS*-VP implementations, each one with eight releases (R1-R8). During the execution of steps 1 and 2, we strictly followed the same implementation decisions made by the original MobileMedia developers, such as extracting exception handling code according to Castor et al. [9], and aspectizing all optional and alternative features. During the execution of Step 3, we have used the same metric suites of the original MobileMedia empirical study [8]. The majority of the metrics were collected using tools, such as Aopmetrics [1].

### V. Change Impact Analysis

This section describes the change impact on PLA elements. The change impact on PLA elements is measured by the number of components and connectors changed or added. The greater the number of architectural elements affected (i.e., changed or added), the greater is the impact on the PLA. A PLA is resilient if its elements are little impacted by evolutions. The change impact metrics has been collected comparing each release to its previous one (e.g. comparing R2
A. Inclusion of Kernel Features

This section presents the results of the change impact caused during the inclusion of the kernel features ExceptionHandler and LabelMedia, which were included in R2 and R3, respectively. The overall results, presented in Figure 5(a), show that COSMOS*-VP PLA had the lowest number of PLA elements changed. That happened because COSMOS*-VP succeeded in isolating changes inside base-level components and inside Connector-VPs. Hence, the changes were not propagated to the aspectual-level components connected by the Connector-VPs. For example, in R3, the metainformation about media was changed from String to ImageData type, which implied changes in various XPIs which were relying on that reference. ImageData also embodies the same information previously contained in String type. On COSMOS*-VP PLA, this type mismatch could be adapted by the Connector-VPs. Thus, these changes were isolated in base-level components by the Connector-VPs. That was not possible in COSMOS*-XPI PLA. Due to the use of hard-wired connectors, which do not have adapter capabilities, the changes had to be propagated to the aspectual-level components.

In the number of PLA elements added, the result was the same in both PLAs. That happened because kernel features were implemented by base-level components in both PLAs. As COSMOS*-VP does not differ from COSMOS*-XPI on how kernel features are implemented by base-level elements, the same base-level components and connectors needed to be added in both PLAs. Furthermore, no new aspect-connector was needed. Thereby, the same number of PLA elements was added in both PLAs.

B. Inclusion of Optional Features

This section describes the inclusion of the optional features Favourites, CopyMedia, and SMS (R4, R5, and R6). During the inclusion of these features, COSMOS*-VP PLA presented the lowest number of PLA elements changed and added (see Figure 5(b)). The results of COSMOS*-VP were possible because of two reasons. First, Connector-VPs could be used to connect more than one optional component, added during the evolutions to the aspectual level of the PLAs, to base-level ones. A Connector-VP can be reused to connect more than one aspectual-level component, since the components connected are associated with the same architectural variation point. That is not possible with COSMOS*-XPI aspect-connectors, which led to a higher number of added connectors in COSMOS*-XPI PLA. Second, due to the scattered implementation of architectural variation points over COSMOS*-XPI PLA elements, each inclusion of new optional component changed more elements in COSMOS*-XPI than in COSMOS*-VP PLA.

C. Inclusion of Alternative Features

The last two releases (R7 and R8) included the alternative features Music and Video, respectively. The overall results show a great advantage for COSMOS*-VP against COSMOS*-XPI (see Figure 5(c)). COSMOS*-VP presented a much lower number of PLA elements added and changed. The employment of Connector-VP facilitated the COSMOS*-VP PLA evolution by isolating from components the implementation that supports the variability decisions of the architectural variation points.

In R7, which included the alternative feature Music, the kernel feature Photo was turned into alternative. In R8 the alternative feature Video was also included. These evolution...
scenarios led to a big impact on most of the architectural variation points of the PLAs. They were impacted due to the necessity of considering the new Music and Video components created. The architectural variation points, which were associated with just optional components, had changed in order to allow different combinations of the optional components and the new alternative ones (Photo, Music, and Video). On COSMOS*-XPI PLA, the optional components were changed in order to create new Abstract Aspects to be connected to the new alternative components. And also new aspect-connectors between these components had to be created. On COSMOS*-VP, only the Connectors-VPs, already used to connect the optional components to the PLA core, had to be changed to provide the optional features to the new alternative components included. Therefore, the use of Connector-VP decreased the number of elements changed and added in these releases, providing the best result to the addition of alternative features (R7 and R8), and also to turn a kernel feature into alternative (R7).

VI. Modularity Analysis

This section presents the results for the modularity analysis according to two metrics, namely cohesion and coupling. These metrics were chosen because they are previously-valdated stability indicators as presented in several experimental studies (e.g. [8], [9], [11]). The majority of these metrics can be automatically collected by applying metric tools, such as Aopmetrics [1].

The modularity of all PLAs is discussed using Lack of Cohesion of Methods (LCOM) [6] and Efferent Coupling (Ce) [6] metrics. The notion of cohesion is related to encapsulation, that is keeping related things together. Thus, a high LCOM may indicate bad design. Efferent coupling refers to the degree of interdependence between parts of a design, which means that a high interdependence can harm maintainability.

Figure 6 presents PLA modularity in terms of average LCOM and average Ce of all PLA elements measured in each release. The overall results show that COSMOS*-VP presents a PLA implementation as modular as COSMOS*-XPI. It means that the employment of Connector-VPs to avoid scattered implementation of architectural variation points did not harm the modular PLA design provided by combined use of components, AOP and XPIs approach.

Regarding LCOM (Figure 6 (a)), COSMOS*-VP PLA presents slightly higher LCOM than COSMOS*-XPI. That happened because the COSMOS*-XPI aspect-connectors are very cohesive, once they are made by just one aspect bridging XPIs and Abstract Aspects. Although, COSMOS*-VP Connectors-VPs are simple, they are more complex than COSMOS*-XPI aspect-connectors, thereby they have higher LCOM.

Figure 6 (b) illustrates the results for Ce. COSMOS*-XPI and COSMOS*-VP have very similar overall results for Ce, with a slightly advantages for COSMOS*-VP on last two releases (R7 and R8). The inclusion of alternative features in these releases implied the creation of new aspect-connectors on COSMOS*-XPI PLA. As explained in Section V-C, these connectors, with their dependencies, were not created in COSMOS*-VP PLA.

VII. Related Work

Mezini and Osterman [19] propose a new model called Caesar, which allows multiple different decompositions simultaneously. Caesar comprises the concept of collaboration interfaces, which differ from standard interfaces in two ways: (i) collaboration interfaces introduce modifiers to annotate required and provided operations; (ii) it uses interface nesting in order to express the interplay between multiple abstractions of a component. Different from Caesar, Cosmos-VP defines an implementation model which does not demand new mechanisms and can be implemented in mainstream programming languages. Furthermore, Caesar does not comprise the concept of aspect-connectors, which is a key concept in COSMOS*-VP model.

Some works propose the integration of components and aspects into new models promoting the encapsulation of advice code (e.g. [17], [22]). Like COSMOS*-VP, one of their goals is to increase reusability of advice code. FAC approach [22] also comprises new programming mechanisms and an architecture description language (ADL) to support their approach. Lagaisse and Joosen [17] describe a framework whose functional layer contains the core application and the middleware layer offers non-functional services. Aspect-connectors link both layers, which allows non-functional crosscutting concerns to be separately encapsulated. COSMOS*-VP approach differs by using connectors to encapsulate variation points.

The problem caused by the scattering of architectural variation point is related to optional feature problem, which occurs when optional features are mutual independent in the domain, but have mutual dependencies in their implementation [14]. This problem limits the variability of a PLA. Kästner et al. [14] survey different approaches to solve the problem and suggest that derivative modules and conditional compilation can eliminate implementation dependencies and thus restore PLA variability. However, both of these approaches have their shortcomings. Conditional compilation may harm separation of concerns and modularity [8]. Derivative modules approach states that code responsible for the dependency should be extracted from the features implementation modules and implemented as a new module. Nevertheless, having several modules implementing a feature can harm the stability of the PLA, since whether a evolution in the feature is required,
it may impact on all these modules. Connector-VP while mediates interactions among non-kernel features implementation, it avoid these features to be scatterly implemented over several PLA elements.

Lahire et al. [12] extend the SmartAdapters approach to support variability. The SmartAdapters approach specifies how a reusable concern should be composed with other concerns through a set of adaptations, which are described using a domain-specific language. The extended approach also provides supports for variability in the weaving process in order to make the concern more reusable. Whereas they use adapters to increase the reusability of concerns, our solution employs connectors to encapsulate architectural variation points aiming at increasing PLA stability.

VIII. CONCLUSION AND FUTURE WORK

PLA is a key artefact to achieve a controlled evolution and, hence, it is important for organisations to understand how PLAs evolve and which approaches better support PLA stability. The main contribution of this paper is a novel component implementation model, namely COSMOS*-VP, which improves the stability of component-based PLAs, by avoiding scattered implementation of architectural variation points over several PLA elements. To achieve more stable PLAs, our proposed solution encapsulates the implementation needed to support all variability decision related to architectural variation points into aspect-connectors, namely Connector-VP. Which are also employed to mediate the connection between aspectual-level components and base-level components.

COSMOS*-VP was compared in the presence of heterogeneous evolutionary scenarios against the combined use of components, aspects and XPIs to implement component-based PLAs. The overall results show that the COSMOS*-VP PLA was more resilient than the other approach involved. We concluded that encapsulating the implementation needed to support variability decisions inside aspects-connectors, called Connector-VPs, COSMOS*-VP reduces change propagation. This was similar in all types of change scenarios involving kernel, optional, and alternative features in MobileMedia. We identified two main threats to validity of our study case: (i) the evolution scenarios might not be representative; and (ii) MobileMedia might not be representative of industrial SPLs. Risk (i) cannot be completely avoided owing to the lack of documentation in the literature about industry representative evolution scenarios in SPL. Moreover, we minimize the risk exercising several evolution scenarios, which involved kernel, and non-kernel optional, and alternative features. Regarding risk (ii), even though MobileMedia is a small SPL, it is heavily-based on industry-strength technologies. Furthermore, it has been extensively used and evaluated in previous research [4], [5], [8]. In fact, we are working on a new case study of a more representative SPL. Although Connector-VPs can facilitate PLA evolution and the fact that they can be semi-automatically created, their complexity can harm the code comprehension. Thereby, the new case study must comprise an evaluation of the Connector-VPs drawbacks.

ACKNOWLEDGMENT

We would like to thank the anonymous referees for the insightful comments. Marcelo Dias is supported by Fapesp/Brazil under grant 2008/02501-9. Leonardo P. Tizzei is supported by Capes/Brazil under grant 05866/2007. Cecilia M. F. Rubira is partially supported by CNPq/Brazil productivity grant 301446/2006-7. Alessandro Garcia is partially supported by Faperj/Brazil distinguished scientist grant E-26/102.211/2009, by CNPq productivity grant 305526/2009-9, by CNPq Universal project grant 483882/2009-7, and by PUC-Rio productivity grant. And also, this research is partially supported by the Service Oriented Software Development through Product Line Engineering Technology project, which is funded by POSTECH, Phang, South Korea.

REFERENCES