On the Modularity Assessment of Software Architectures: *Do my architectural concerns count?*

Cláudio Sant'Anna[1,2], Eduardo Figueiredo[2], Alessandro Garcia[2], Carlos J. P. Lucena[1]
[1]Computer Science Department, PUC-Rio, Brazil
[2]Computing Department, Lancaster University, UK
[claudios, lucena]@inf.puc-rio.br, [a.garcia, e.figueiredo]@lancaster.ac.uk

ABSTRACT
Much of the complexity of software architecture design is derived from the inadequate modularization of key broadly-scoped concerns, such as exception handling and persistence. However, conventional architecture metrics are not sensitive to the driving architectural concerns, thereby leading a number of false positives and false negatives in the design assessment process. Hence, there is a need for assessment techniques that support a more effective identification of early design modularity anomalies relative to crosscutting concerns. In this context, this paper proposes a concern-driven measurement framework for assessing architecture modularity. It encompasses a mechanism for documenting architectural concerns, and a suite of concern-driven architecture metrics. We evaluated the usefulness of the proposed framework while comparing the modularity of aspect-oriented (AO) and non-AO architecture design alternatives in three different case studies.

Keywords
Architecture Modularity, Separation of Concerns, Measurement

1. INTRODUCTION
Modularity has been playing a pervasive role in early design stages even before the emergence of the software architecture discipline [13]. Software engineers consider modularity as a key principle when comparing architecture alternatives and analysing architecture degeneration [9]. In fact, software engineers are fostered to design good architectures by relying on a plethora of modularity mechanisms available in: (i) architecture description languages (ADLs), such as ACME [8], (ii) catalogues of architectural styles [2, 13], and (iii) well-know high-level design principles, such as narrow component interfaces, reduced architectural coupling, and the like.

However, strict reliance on these architecture mechanisms is not enough to achieve truly modular architecture designs [1, 3, 10]. Moreover, assessment and improvement of early design modularity is even more challenging. Quantitative assessment techniques are needed for evaluating architecture alternatives. Software metrics are a powerful means to provide modularity indicators of architecture design [3]. The software metrics community has consistently used notions as module coupling and cohesion to derive measures of software architecture quality [1, 10, 16].

In fact, the conception of the right architecture decomposition is still a deep bottleneck to the software design process. A number of widely-scoped architecture concerns that emerge in early development phases need to be simultaneously modularized. An architectural concern is some important part of the problem that we want to treat in a modular way at the architecture specification [4, 5]. Much of the complexity of software architecture design is derived from the inadequate modularization of architectural concerns, such as graphical user interface (GUI), exception handling, and persistence. Architects tend to naturally give priority or focus on the modularization of certain concerns, while choosing a certain combination of existing architecture styles or relying on a particular way for architecture decomposition. As a result, a number of concerns end up having a crosscutting impact on the system architectural decomposition, and systematically affecting the boundaries of several architectural elements, such as components and their interfaces [5, 14].

Although typical architecture modularity problems are related to the inadequate modularization of concerns, most of the current quantitative assessment approaches do not explicitly consider concern as a measurement abstraction. A number of architecture quantitative assessment methods are targeted at guiding decisions related to modularity, without calibrating the measurement outcomes to the driving architectural concerns. It imposes certain shortcomings, such as the ineffective identification of desirable and undesirable couplings. Also, this necessity becomes more apparent in an age that a number of different forms of architecture decompositions are emerging. For example, aspect-oriented software architectures and feature-oriented product-line architectures [15] support different composition mechanisms for enhancing the separation of certain concerns. A number of case studies have pointed out that detection of certain design flaws can be observed in early design stages [14, 17].

Software architects need, therefore, quantitative assessment approaches that support them to identify modularity anomalies related to inadequate modularization of architectural concerns. We believe that concern-driven quantitative assessment improves the architecture modularity analysis, because it makes more evident the overall influence of the (in)adequate modularization of widely-scoped design concerns. In this context, the contributions of this paper are threefold: (i) a initial list of shortcomings associated with conventional architecture modularity measurement (Section 2), (ii) a concern-driven framework for assessing architecture modularity (Section 3), and (iii) a initial evaluation of the proposed framework (Section 4). In addition to a concern-sensitive metrics suite, the framework includes a mechanism for documenting concerns in architecture descriptions. We evaluated the usefulness of the measurement framework while comparing the modularity of aspect-oriented (AO) and non-AO architecture design alternatives in the context of three case studies.

2. WHY CONCERN-DRIVEN ARCHITECTURE ASSESSMENT?
Current quantitative architecture assessment approaches [1, 10, 16] usually rely on traditional abstractions such as component (or

module) and its interfaces in order to undertake the measurements. Based on these abstractions, they define and use metrics for quantifying attributes such as coupling between components, component cohesion, interface complexity, and so forth. Figure 1 depicts an architecture that will serve as a running example throughout this paper, and as an illustration for the limitations of conventional architecture metrics. It shows a partial, simplified graphical representation of the architecture description (component-and-connector view [19]) of a real Web-based information system, called Health Watcher [4, 17]. The design is structured mainly following the Layer architectural style [2]. In this system, for instance, persistence is an architectural concern. It is addressed by the Data_Manager and Transaction_Manager components. Business is also considered an architectural concern and is addressed by the Business_Manager component. In the light of this example, we discuss the limitations of current architecture metrics.

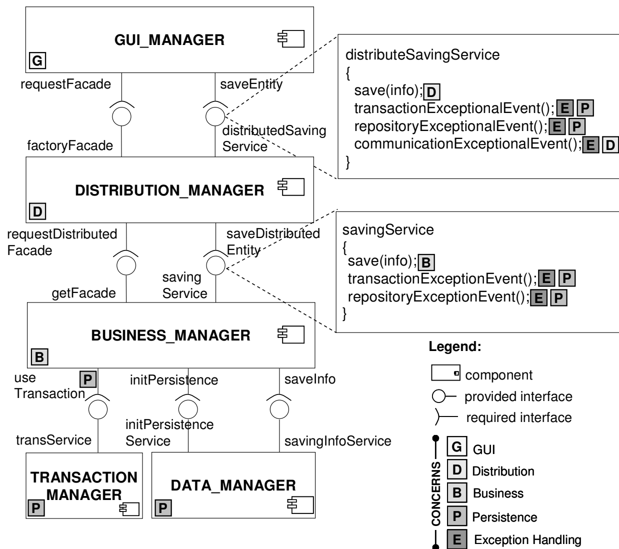

Figure 1. Architecture of the Health Watcher system

Inaccuracy on Modularity Analysis. When choosing certain architecture abstractions, styles and mechanisms for decomposition, architects may leave some concerns non-modularized. In other words, these concerns are not satisfactorily captured in separate modular units in the architecture description, i.e. only localized within one or a few components with well defined interfaces. In the architecture shown in Figure 1, the exception handling concern is partially addressed by abnormal events, such as transactionExceptionalEvent and repositoryExceptionalEvent, exposed in several interfaces of the components. In this system, it is important to well modularize the global exception detection strategies because they are similarly applied over several key services defined by the system architecture. Current architecture metrics are not able to highlight that this concern has a wide impact on several interfaces. The main problem is that they do not rely on the identification of the architectural elements related to each concern, thereby causing a number of false negatives in the architecture assessment process. The reason is that typical architecture metrics are not able to explicitly capture this kind of modularity-related impairment as, for instance, exception handling is an architectural property typically diffused all over the architecture elements. Hence, this implies that existing metrics are often inaccurate to support the identification of non-modularized architectural concerns.

The Tyranny of Dominant Modularity Attributes. Software architecture measurement also suffers from what we call the tyranny of the dominant architectural modularity principles. The fact that the assessment of certain principles, such as low coupling and narrow interfaces, are overemphasized, other equally important design principles, such as separation of concerns, have been neglected in architecture measurement processes. It might hamper trade-off analysis in scenarios where priorities need to be given, for instance, either to the separation of a specific concern or to the degree of coupling of architectural components involved. More importantly, it is not possible to understand how the separation of certain architectural concerns influence other modularity attributes, such as coupling. As shown in Figure 1, the exception handling concern affects the distributedSavingService and savingService interfaces, since they have to expose exceptional events. These events contribute to increase the complexity of those interfaces. Even though the traditional metrics can provide information about the interface complexity, they do not support the architects to reason about the fact that the exception handling concern is the main contributor for that complexity. Hence, the identification of the impact of architectural concerns on traditional attributes is hindered.

In addition, current coupling metrics are inaccurate to identify architectural inter-concern dependencies. Coupling architecture metrics quantify the dependence between components, assessing, in this way, the dependence between only the primary concerns modularized within components. Concerns which are not entirely modularized by the architecture abstractions do not have modular boundaries in the sense that their boundaries are not well defined by component interfaces. Hence the dependence between such non-modularized concerns or even between non-modularized and modularized concerns cannot be measured by traditional measures. In Figure 1, the distribution concern depends on the persistence concern because the Distribution_Manager component includes persistence-related exception events, which are not modularized by any component. Moreover, the exception handling concern interacts with the persistence concern because the transactionExceptionEvent and repositoryExceptionEvent events are related to both concerns.

3. A CONCERN-DRIVEN MEASUREMENT FRAMEWORK

This section is targeted at defining a concern-sensitive measurement framework for assessing architecture modularity. It complements existing architecture metrics by explicitly promoting concern as a measurement abstraction. The framework aims at supporting the software engineers to: (i) anticipate modularity problems caused by architecturally-relevant concerns, and (ii) compare alternatives of architecture design solutions with respect to their ability to modularize distinct sets of prioritized concerns.

Our framework mainly relies on evaluating the modularization of architectural concerns. Therefore it includes metrics for quantifying separation of concerns and their interactions. For instance, it quantifies the diffusion of a concern realization within architecture specification elements, such as components and interfaces. The metrics suite also evaluates how a particular concern realization affects traditional attributes such as coupling, cohesion and interface complexity. Hence it includes metrics for assessing these attributes. Our concern-oriented metrics focus on the evaluation of software architecture representations, such as UML-based or ADL specifications. Also, in order to consider concern as an abstraction

in the measurement process, there is a need to explicitly document the concerns in the architecture. Therefore, our approach also includes a notation to support the architect with the documentation of the driving architectural concerns (Section 3.2). Using this notation, the architect can assign every architecture element (components, interfaces, and operations) to one or more concerns.

3.1 Concern-Driven Metrics

Table 1 presents a summary of the architecture metrics suite with a brief definition for each of the metrics and their association with distinct modularity attributes they measure. The metrics definition is agnostic to specific ADLs. Therefore, in order to apply the metrics, it is necessary to adapt their definition to the specific abstractions of the architecture description approach in use.

Looking again to the architecture in Figure 1, using the proposed metrics (Table 1) we can now quantify the effects of the exception handling concern in the architecture. After documenting that the exception events are related to the exception handling concern, we can compute the concern-driven metrics. The results will show that the exception handling concern is spread over several components and interfaces. Moreover, the results of the Lack of Concern-based Cohesion metric for the GUI_Manager, Distribution_Manager and Business_Manager components will show that there is more than one concern present in each of those components. In this way, the architect will be warned that in the Business_Manager component, for instance, besides the business concern, there are other concerns contributing for the complexity of the component.

3.2 Concern Templates

In order to support to application of the concern-driven metrics, we defined what we call as concern template. A concern template captures the architecture elements associated with key concerns, letting the architects to represent all the architectural implications related to a concern in a single place. The template includes the following information: (i) name of the concern; (ii) architecture elements, such as components, interfaces, relationships, operations, events and so forth, related to the concern; and (iii) composition rules to describe how the elements with respect to a concern affect architectural elements related to other concerns.

Figure 2 shows how to use the template to support the description of architectural designs relative to the exception handling concern in the Health Watcher system architecture (Figure 1). The template shows the three operations assigned to the persistence concern and how they are composed with the other elements in the architecture, that is, in which interfaces they are present. Based on this information is it possible to compute all the concern-driven metrics we defined.

```
Concern: Exception Handling
Architecture Elements:
    transactionExceptionalEvent() ;
    repositoryExceptionalEvent() ;
    communicationExceptionalEvent() ;
Composition Rules :
Add event communicationExceptionalEvent() to
    interface distributedSavingService;
SavingSet = distributedSavingService, savingService;
Forall I in SavingSet
   Add event transactionExceptionalEvent() to I;
   Add event repositoryExceptionalEvent() to I;
```

Figure 2. Concern Template.

4. EVALUATION AND DISCUSSION

We undertook three case studies in order to carry out an evaluation of our measurement framework. We have used the framework to perform modularity comparisons of two different architecture alternatives for each of three systems. The first alternative is always an aspect-oriented (AO) architecture. The second version is based on one or more specific architectural styles [2, 13], herein referred as non-aspectual (non-AO) architecture.

The first of our series of case studies was a multi-agent system framework, called AspectT [5]. The evaluation included an architecture mainly based on the Mediator pattern, in addition to an AO architecture. The second study encompassed publisher-subscriber [2] and aspect-oriented versions of the MobiGrid architecture [18], a framework used to develop mobile agents in Grid environments. The third case study involved the AO and non-AO layered architecture of the Heath Watcher system, presented in Section 2. Figure 3 shows a partial view of the AO architecture of the Health Watcher system. In the following we present some data gathered with the application of our metrics in the architectures of the Health Watcher system and discuss how they support overcoming the limitations of conventional metrics (Section 2). Due to space limitation, we will give more attention to the results for the Health Watcher architecture, since it is our running example in this paper. The complete description of the gathered data is

Table 1. Suite of Concern-Driven Architectural Metrics

Attribute	Metric	Definition
Concern Diffusion	Concern Diffusion over Architectural Components (CDAC)	It counts the number of architectural components which contributes to the realization of a certain concern.
	Concern Diffusion over Architectural Interfaces (CDAI)	It counts the number of interfaces which contributes to the realization of a certain concern.
	Concern Diffusion over Architectural Operations (CDAO)	It counts the number of operations which contributes to the realization of a certain concern.
Coupling Between Architectural Concerns	Component-level Interlacing Between Concerns (CIBC)	It counts the number of other concerns with which the assessed concerns share at least a component.
	Interface-level Interlacing Between Concerns (IIBC)	It counts the number of other concerns with which the assessed concerns share at least an interface.
	Operation-level Overlapping Between Concerns (OOBC)	It counts the number of other concerns with which the assessed concerns share at least an operation.
Coupling Between Components	Afferent Coupling Between Components (AC)	It counts the number of components which require service from the assessed component.
	Efferent Coupling Between Components (EC)	It counts the number of components from which the assessed component requires service.
Component Cohesion	Lack of Concern-based Cohesion (LCC)	It counts the number of concerns addressed by the assessed component.
Interface Complexity	Number of Interfaces	It counts the number of interfaces of each component.
	Number of Operations	It counts the number of operations in the interfaces of each component.

reported elsewhere [7].

Identification of non-modularized concerns. Analysing the results for the concern diffusion metrics (Table 1) relative to the Health Watcher system, we can see that the persistence and exception handling concerns are spread over more architecture elements in the non-AO solution. For instance, in the non-AO architecture, the persistence affects more components (CDAC metric) – 5 vs. 2, more interfaces (CDAI metric) – 22 vs. 9 – and more operations (CDAO metric) – 154 vs. 45. This occurs mainly because in the AO solution the persistence-specific exceptional events are modularized within the Transaction_Manager aspectual component and, as a consequence, do not need to be addressed by the interfaces of Business_Manager, Distribution_Manager and GUI_Manager components as in the non-AO solution.

Breaking the tyranny of the dominant architectural modularity attributes. Analysing the results of the Number of Interfaces (NI) and Number of Operations (NO) metrics, which are conventional metrics, we can see that three components in the Health Watcher architectures, namely GUI_Manager, Distribution_Manager and Business_Manager, have more complex interfaces in the non-AO architecture. For instance, the Business_Manager component has more interfaces (11 vs. 9) and more operations (64 vs. 57) in the non-AO architecture. Although this information is important, it does not give any clue about the reasons for that. In this way, the results for the Lack of Concern-based Cohesion metric complement this information in the sense that it showed that those components which have more complex interfaces also have more concerns affecting them in the non-AO solution. Therefore, these concerns can be one of the causes for the higher interface complexity. Also, in the case studies about the AspectT and MobiGrid architectures, the metrics highlighted that the bidirectional coupling between several components was caused by a concern spread over them. The metrics also showed that the AO versions of these architectures were better alternatives than non-AO ones. The reason was that the AO architectural abstractions were able to separate the concern from those components, eliminating, as a consequence, the bidirectional coupling between them.

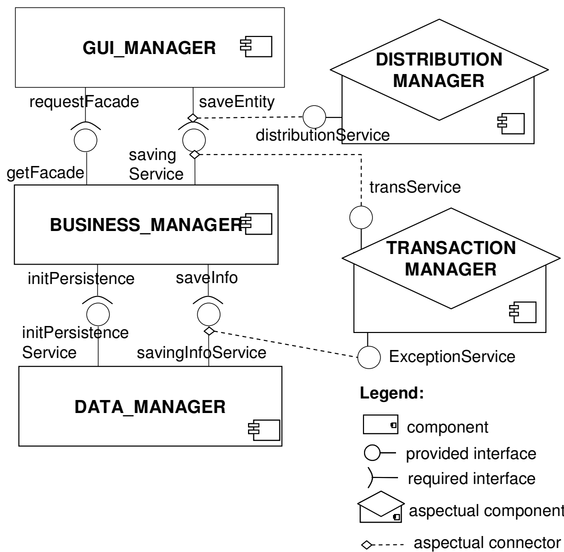

Figure 3. AO Architecture of Health Watcher System

5. CONCLUDING REMARKS

Up to date, to the best of our knowledge no concern-oriented metrics have been developed to evaluate software architecture modularity. There is only an initial work on concern-based metrics for the implementation level [11]. In this way, this paper proposes a measurement framework consisting the following: (i) a suite of concern-oriented metrics for evaluating architecture modularity, and (ii) a technique for documenting the concerns in the architecture. This paper also discusses some limitations of the conventional architecture metrics. In the future, we plan to develop a tool for automating both the architecture concerns' documentation and the metrics application. We also plan to: (i) analyse how the use of our framework to control the modularization of concerns at the architecture-level impacts in the implementation artefacts, since they simply represent projections of the architecture design, and (ii) validate the metrics in terms of their impact on external quality attributes, such as maintainability and reusability.

6. REFERENCES

[1] L. Briand, S. Morasca, V. Basili, Measuring and Assessing Maintainability at the End of High Level Design, Proc. IEEE Conf. Software Maintenance, 1993.

[2] F. Buschmann et al. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley, 1996.

[3] L. Dobrica, E. Niemela. A Survey on Software Architecture Analysis Methods. IEEE Trans. on Soft. Eng. 28 (7), July 2002, pp. 638-653.

[4] A. Garcia at al. On the Modular Representation of Architectural Aspects. In EWSA 2006, France, 2006.

[5] A. Garcia, C. Lucena. Taming Heterogeneous Agent Architectures with Aspects. Comm. of the ACM, July 2006. (accepted to appear)

[6] A. Garcia et al. Modularizing Design Patterns with Aspects: A Quantitative Study. In AOSD'05, 2005, pp. 3–14.

[7] Concern-Driven Measurement Framework for Assessing Architecture Modularity,
URL: http://www.lancs.ac.uk/postgrad/figueire/co_metrics

[8] Garlan, D. et al. ACME: An Architecture Description Interchange Language, Proc. CASCON'97, November 1997.

[9] Lindvall, M. et al. Avoiding Architectural Degeneration: An Evaluation Process for Software Architecture. In Proc. of the Intl. Symposium on Software Metrics, 2002, USA, p. 77.

[10] Lung, C., Kalaichelvan, K. An Approach to Quantitative Software Architecture Sensitivity Analysis. Proc. of the Int'l Conf. on SW Eng & Knowledge Eng., 1998, pp. 185-192.

[11] A. Di Stefano et al. Metrics for Evaluating Concern Separation and Composition. ACM Symposium on Applied Computing (SAC'05), March 13-17, 2005, New Mexico, USA, pp. 1381-1382..

[12] C. Sant'Anna, et al. On the Quantitative Assessment of Modular Multi-Agent System Architectures. NetObjectDays (MASSA), 2006.

[13] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, Apr. 1996.

[14] U. Kulesza et al. Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. Proc. of ICSM'06, Philadelphia, USA, September 2006.

[15] R. Filman et al (editors). Aspect-Oriented Software Development. Addison-Wesley, Boston, 2005.

[16] R. Martin, Stability, C++ Report, Feb. 1997.

[17] S. Soares, et al. Implementing Distribution and Persistence Aspects with AspectJ. In Proc. of OOPSLA'02, 2002, p. 174-190.

[18] R. Barbosa, A. Goldman. MobiGrid. In A. Karmouchet al, editors, MATA, vol.3284, LNCS, pages 147–157. Springer, 2004.

[19] L. Bass et al. Software Architecture in Practice. Addison Wesley, 2nd Edition, 2003.