Aspectual vs. Component-based Decomposition: A Quantitative Study

Dimitri Van Landuyt, Steven Op de beeck, Johan Grégoire, Riccardo Scandariato and Wouter Joosen
DistriNet, Dept. Of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
first.lastname@cs.kuleuven.be

Andrew Jackson,
Siobhán Clarke
Distributed Systems Group
Trinity College Dublin
Dublin 2, Ireland
first.lastname@cs.tcd.ie

ABSTRACT

This position paper presents ongoing work in which we explore the claim that aspect-oriented decomposition, at the architectural and design level, promotes the software engineering property of *separation of concerns*. To this end, we empirically compare an aspect-oriented design methodology with a state-of-the-art component-based design. These methodologies were used to build two designs based on the same set of requirements, in complete separation of each other. During this experiment, we observed a gap in the aspect-oriented methodology, in that it does not explicity address the need to document, communicate, reuse and systematically handle high-level compositions. *Composition designation diagrams* represent our first attempt at modeling this composition overview.

1. INTRODUCTION

This paper presents currently ongoing work in which we try to assess the value of aspect orientation in architecture and design by empirically comparing an aspect-oriented design methodology with a state-of-the-art component-based design methodology. We apply both methodologies to the same set of requirements, resulting in two functionally equivalent designs. In this paper we focus on how these designs differ in terms of separation of concerns (SoC). For aspect-oriented design methodology we use the Theme/UML approach. The component-based design is documented with UML 2.0 [14].

The quantitative evaluation is conducted with metrics that are well-established for the component-based methodology. However, there still exists a need to reformulate some of these metrics to make them applicable to the aspect-oriented design as well. For this reason, and also since we target a comparison both at module level and at system level, we provide a mapping between top-level concepts such as module or interface to the specific constructs of the Theme/UML approach.

AOSD.07 AARCH, March 12, 2007, Vancouver, British Columbia.

The need to document, communicate, reuse and systematically handle high-level compositions in aspect-oriented design demands a kind of overview on the aspect-oriented architecture that is not available in the aspect-oriented methodology we apply. Our first attempt at modeling this composition overview[1] is the *composition designation diagram*. This is presented in Section 4.

The rest of the paper is organized as follows. The Theme/UML approach is shortly discussed in Section 2. Section 3 presents a quick overview of the experiment. Composition designation diagrams are introduced in Section 4. Section 5 presents preliminary results w.r.t. SoC, and identifies a trade-off between SoC and comprehensibility. Finally, Section 6 presents related work and Section 7 concludes the paper.

2. THEME/UML

Theme/UML [4] is an aspect-oriented approach that supports the separation and composition of crosscutting concerns in design. Theme/UML extends the Unified Modeling Language (UML) by adding new constructs like the theme classifier (adaptation of the UML package construct) and the merge relationship between themes. A theme encapsulates different types of models, e.g. class and sequence diagrams are typically used within a theme to design both the structure and behaviour of a concern. A theme has no reference to any concept defined externally to the theme itself. This property is called *declarative completeness* and ensures that each concern can be designed in complete isolation from other concerns.

In Theme/UML we refer to themes that model non-crosscutting concerns as *base themes* and, contrasting, themes that model crosscutting concerns as *aspect themes*. This crosscutting behaviour is described using template parameters, i.e. parameters of the aspect theme. Template parameterization is a UML technique [14] that can be applied to classifiers (such as classes and collaborations), packages, and operations. Almost any kind of element can be a template

[1] The notation presented here is not formalized enough to be called a viewpoint (or viewtype) [13, 1, 12]. This is why we avoid using the term.

Generic concept	OO (CB Design)	AO (Theme/UML)
module	component	theme
provided interface	provided interface	aspect theme template parameters
required interface	required interface	selected join points in base/aspect theme
module composition	interface matching	theme composition

Table 1: Mapping between generic concepts and approach-specific concepts.

parameter. The arguments that are bound to these parameters are the join points for this crosscutting behaviour. Binding these arguments to the parameters is called instantiation of the aspect theme. This instantiation combined with the merge relationship defines what is called theme composition.

3. METHODOLOGY

The experiment we are conducting is set up as follows. First, we select two teams of designers, which we will call the AO team and the OO team from now on. In each team, we select a lead designer (or main architect). Then a subset of requirements is selected from a voluminous requirements document produced in a project called DigiNews [10], in which a next-generation system for the creation and distribution of news and related content is described. This subset is used as a common starting point between the teams. In a following phase, both teams —strictly separated from each other— follow their respective design approach. This produces an aspect-oriented design in Theme/UML and a component-based design. These designs describe different systems that offer exactly the same functionality —hence they provide a fair base for comparison.

In a next step, we select typical metrics for component-based design and architecture [3, 2], and apply them on the Theme/UML design. To this end, we map the generic concepts [16] of *module*s, *provided* and *required interfaces* to the concepts used in the Theme/UML approach. This mapping is presented in Table 1. The notions of theme and component both map to the generic concept of module. The provided interface in Theme/UML maps to the template parameters of an aspect theme. A base theme has no provided interface. The selected join points in a base theme (or aspect theme) represent the required interface in Theme/UML. With a selected join point, we mean those join points that are bound to the template parameters of an actual composition. Finally, theme composition in Theme/UML and interface matching in component-based design map to the generic concept of module composition. Note that by postulating a mapping between selected join points in a base theme in Theme/UML and a required interface, we implicitly assume modularity —an notion that has recently been questioned [15].

By expressing the different metrics in terms of this more generic terminology, they are now applicable to both designs. We focus on the following key question: "Which design achieves the best separation of concerns?" The indicating metric we employ for this quality attribute is *concern diffusion over modules* (CDM) [8]. The diffusion of a concern is defined as the percentage of "modules" (i.e. either components or themes) that contribute to the realization of that concern. Lower values are desirable for this metric, as they indicate a higher SoC.

Concern diffusion	OO (CB Design)	AO (Theme/UML)
w.r.t. own concerns	0.59	0.13
w.r.t. other team's concerns	0.63	0.36

Table 2: The results for the *concern diffusion over modules* metric.

4. COMPOSITION DESIGNATION

This section illustrates the use of aspects at the architectural level. It presents a subset of the Theme/UML design required for realizing the functionality of one specific use case called "pull an article". This is one of the core use cases needed for requesting news content from the digital publishing system. More specifically, in this use case the main actor —the media consumer— uses a digital newsreader to fetch the latest news articles from the digital publishing system.

The Theme/UML approach does not explicitly offer means for documenting architectural decisions. However, it is clear that since the AO team started from requirements and ended up with a detailed design, they implicitly made some key architectural decisions, but lacking a good way to document them. To fill this gap, the AO team created the so-called *composition designation diagrams*, modeling the top-level abstraction entities (themes) and their interrelations (theme compositions). Figure 1 shows the *composition designation diagram* for the "pull an article" use case. It shows the top-level entities, which are either base themes or aspect themes. An example of a base theme is *publishing* (see left-hand side of Figure 1). Aspect themes in the figure are *messaging, profiling, distribution,* and *security,* which all extend the base behaviour in a crosscutting manner.

The example also illustrates the concept mapping introduced in Section 3. Themes correspond to modules and are represented by UML packages. A dependency arrow leaving a theme denotes a provided interface. In a similar fashion, a dependency arrow arriving at a theme denotes a required interface. Each name associated with an arrow indicates a composition. A composition is a distinct entity, separate from the themes it composes. During architecture it indicates the high-level specification of the crosscutting dependency between the involved themes. Later on, during detailed design, this specification is further refined by identifying concrete join points and other required contextual information.

5. SEPARATION OF CONCERNS

First mentioned by Dijkstra [5], SoC is now widely accepted as one of the major desirable qualities of the design of a software system. It is also one of the major arguments heard in favor of aspect orientation. However, we can not find much empirical proof in literature for the statement that an aspect-oriented decomposition provides a better SoC than a classical component-oriented decomposition. This is what we investigate.

During requirement analysis, both teams came up with a set of concerns they deemed relevant. For fair comparison, the *concern diffusion over modules* (CDM) metric is applied

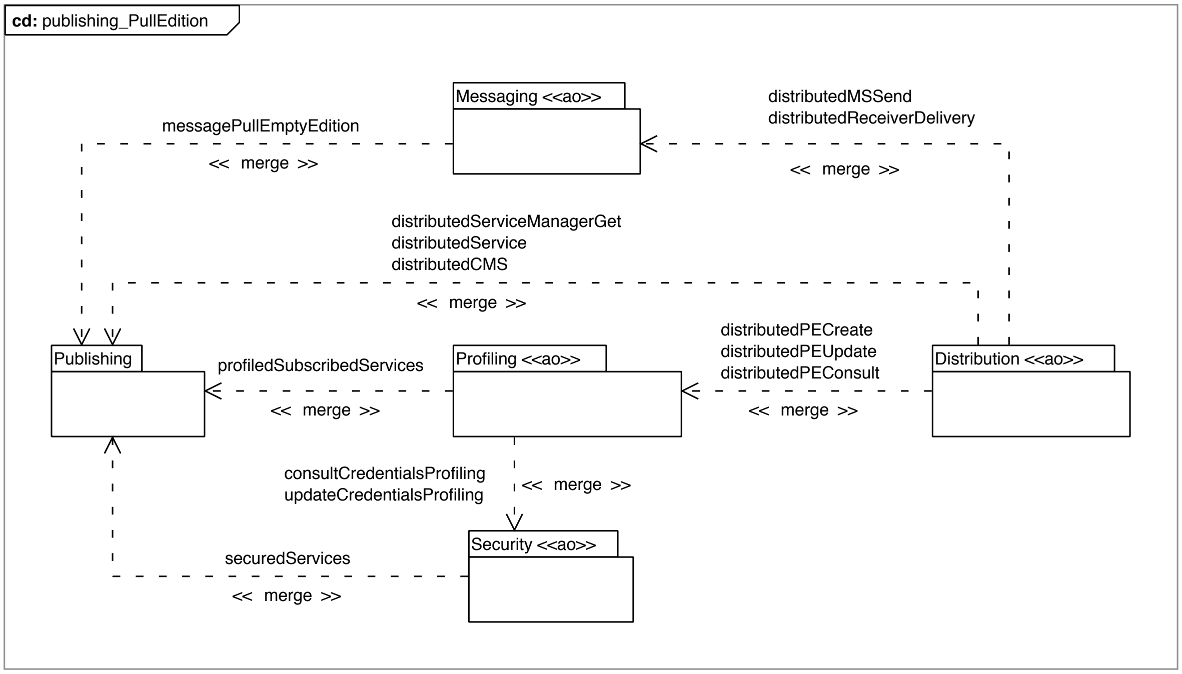

Figure 1: Composition designation diagram for the "pull an article" use case.

to both sets of concerns. The results of this calculation are shown in Table 2. The first row shows the results for the CDM metric with respect to each team's own concern set. The second row shows the results with respect to the other team's concerns.

This table shows that in both cases, the Theme/UML design has lower concern diffusion. This result suggests that the Theme/UML decomposition is superior to the classical component-oriented decomposition w.r.t. SoC. This can be attributed to the inherent concern-oriented nature of the Theme/UML approach.

However, it is our impression that there exists a trade-off between SoC and comprehensibility of design. By shifting more complexity to the composition, it becomes harder to understand the consequences of that composition. However, it can be argued that improved tool support for aspect-oriented design can partially alleviate this disadvantage. Evidently, these claims need further investigation, both quantitatively and qualitatively. This is part of our ongoing work.

6. RELATED WORK

In [11] Kulesza, et al. introduce a graphical notation to represent a so-called aspect-oriented agent architecture. This agent architecture is used for modeling a family of software agents in the domain of Multi-Agent Systems. The model allows one central component (containing all non-crosscutting features) and several *"aspectual components"* representing crosscutting features. Aspectual components in this context expose *crosscutting interfaces* that may add new state or behaviour to other components or *intercept* existing component behaviour. Central components expose their services through *normal interfaces.* The main similarity between this graphical representation and the one we present in this paper is that they both model concerns at the architectural level. However, what is described in this related work is not system level architecture but rather the high-level representation of a software agent —still this may be regarded as a subsystem of some kind. This approach also takes on an asymmetrical organization [9] for composition, while the model we propose is more symmetrically organized. Another difference is that this approach focuses more on defining interfaces for possible interaction, while we focus on describing specific compositions between modules.

In [6], France et al. describe aspect-oriented architecture models that consist of a base architectural model (primary model) and a set of aspect models. The approach allows to model concern solution models (aspect models) in isolation and integrate them into the primary model. This technique shares quite some similarities with the Theme/UML approach, however Theme does not require a single primary model. It allows multiple base themes and aspect themes alongside each other. Also, this approach does does not provide an explicit model specifying and documenting all necessary primary-aspect model compositions required to realize the complete system architecture.

7. DISCUSSION AND CONCLUSION

In this comparative study, we attempt to gain insight in the value of an aspect-oriented decomposition at the architectural level. More specifically, we focus on the software quality of separation of concerns (SoC). To achieve this, we are have designed a system for digital publishing following two fundamentally different design methodologies. These are an aspect-oriented methodology called Theme/UML and a component-based decomposition. To allow direct comparison, we defined a way to match the top-level concepts of module and interface in both approaches onto each other. Then we compared the two artefacts in a quantitative way.

A direct advantage we have observed in the concern-oriented nature of approaches such as Theme/UML is that it captures in a more direct manner the stakeholder concerns for a software system. This is definitively beneficial for the stakeholder as it improves traceability for his concerns throughout the life-cycle of that software system. This stakeholder-driven approach contrasts to the more technology-driven approach that a component-oriented design typically represents.

As this paper presents ongoing work, we will continue the described quantitative experiment, focusing on the qualities of complexity and maintainability. In this paper, the results presented in Section 5 are a first step in that direction: as shown in [7], SoC is one of many qualities influencing maintainability.

We are further going to investigate the nature of the trade-off between comprehensibility and SoC, and perhaps identify other trade-offs. More specifically, we will try to find to what extent the quality of comprehensibility is affected by placing more design complexity into composition, and whether this can be solved by improving tool support for aspectual decomposition.

We will also investigate to what degree we can account the observed differences between the two designs to the inherent nature of the applied approaches themselves, rather than to the difference between the design teams and their experience.

Finally, we will further investigate how to support an aspect-oriented decomposition in architecture and define different views on such an architecture.

Acknowledgments. We would like to thank the people that helped designing and setting up the experiment: Bert Lagaisse, Sam Michiels, Eddy Truyen. Andrew Jackson is supported by a European Commission grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe), 2004-2008.

8. REFERENCES

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[2] S. R. Chidamber, D. P. Darcy, , and C. F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Transactions On Software Engineering*, 24(8):629–639, August 1998.

[3] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions On Software Engineering*, 20(6):476–493, June 1994.

[4] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison Wesley Professional, 1st edition, March 2005.

[5] E. W. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag, 1982.

[6] R. B. France, I. Ray, G. Georg, and S. Ghosh. Aspect-oriented approach to early design modelling. *IEE Proceedings - Software*, 151(4):173–186, 2004.

[7] A. Garcia, C. Sant'Anna, C. Chavez, V. Silva, C. Lucena, and A. von Staa. *Software Engineering for Multi-Agent Systems II*, chapter Separation of Concerns in Multi-Agent Systems: An Empirical Study. Springer-Verlag, February 2004.

[8] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. von Staa. Modularizing design patterns with aspects: A quantitative study. In *Proceedings of the Aspect Oriented Software Design Conference*, 2005.

[9] W. H. Harrison, H. L. Ossher, and P. L. Tarr. Asymmetrically vs. symmetrically organized paradigms for software composition. Technical report, IBM Research Division, published under RC22685, 2002.

[10] W. Joosen, S. Michiels, E. Truyen, K. Vandebroek, and Dimitri Van Landuyt. A comprehensive model for digital publishing. Technical Report CW 443, Katholieke Universiteit Leuven, Department of Computer Science, April 2006.

[11] U. Kulesza, A. Garcia, and C. Lucena. An aspect-oriented generative approach. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 166–167, New York, NY, USA, 2004. ACM Press.

[12] T. I. of Electrical and E. E. I. S. Board. Recommended practice for architectural description of software-intensive systems (ansi/ieee-std-1471), September 2000.

[13] N. Rozanski and E. Woods. *Software Systems Architecture*. Addison-Wesley Professional, 1 edition, April 2005.

[14] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual, Second Edition*. Addison-Wesley, 2005.

[15] F. Steimann. The paradoxical success of aspect-oriented programming. volume 41, pages 481–497, New York, NY, USA, 2006. ACM Press.

[16] C. Szyperski and C. Pfister. Component-oriented programming: Wcop'96 workshop report, 1997.