Early aspects in "Aspect-Oriented Process for a Smooth Transition"

Fernando Pinciroli[1], José L. Barros-Justo[2]

[1] Research Institute of the Faculty of Informatics and Design, Champagnat University, Mendoza, Argentina
pincirolifernando@uch.edu.ar
[2] School of Informatics (E.S.E.I.), University of Vigo, Ourense, Spain
jbarros@uvigo.es

Abstract. The aspect-oriented paradigm brought new expectations about building software in a modular way and improving many quality attributes, but it also came with new challenges. One of them is the lack of casuistry of its use in the industry. Software development project leaders are already facing enough risks and should not add more, such as incorporating an immature approach. Furthermore, there are no aspect-oriented proposals using standard notations and covering the full Software Development Life Cycle (SDLC). We have elaborated an alternative called Aspect-Oriented Process for a Smooth Transition (AOP4ST), that allows the smoothly incorporation of the aspect-oriented paradigm in the current industrial projects and offers a complete homogenous proposal for the phases of the SDLC. In this paper, we present the first stages of AOP4ST, seeking the incorporation of the aspect-oriented paradigm in the industry with the least impact, but trying to take full advantage of the possibilities it offers.

Keywords: business process modeling; requirement modeling; aspect-oriented software development; crosscutting concerns; UML; BPMN; AOP4ST.

1 Introduction

The emergence of the aspect-oriented paradigm brought new expectations about the possibility of building software in a more modular way and improving its quality attributes, such as maintainability, flexibility, comprehensibility, reusability, etc.

However, the new paradigm also came with new challenges, since there is not enough casuistry of its use in the industry guiding towards a way to apply it properly, particularly with regard to techniques, tools, notations and good practices.

Software development projects must deal with a large number of risks. It is not advisable to add new ones, such as incorporating an approach that is not sufficiently mature and that requires training people in poorly known tools and techniques, applying not well-tested methods, lacking support from vendors, and many more.

Similarly, there are no methodological proposals employing standard notations and covering the full SDLC, so in case of applying the aspect-oriented paradigm, we are

obliged to compose a method by picking up parts from different authors, who worked each phase of the SDLC in isolation [1].

In this context, we decided to move forward with an alternative that allows us to incorporate the aspect-oriented paradigm in the current projects in the industry, and offering a complete proposal by unifying homogenously the different phases of the SDLC.

In this paper, we present the models of AOP4ST corresponding to the early stages of the SDLC. AOP4ST is a framework process for software development whose aim it the incorporation of the aspect-oriented paradigm in a smooth way, causing the least possible negative impact in the industry, but trying to make the most with all of the advantages that current aspect-oriented paradigm offers.

In section 2 we briefly present the AOP4ST's schema. The subsequent sections explain the different parts of the process: in Section 3 we offer a description of the business model and Section 4 describes the user requirement model. Next, in Section 5, we present the three views that make up the system requirement model: functional, static and state views. Finally, Section 6 presents the conclusions, and highlights some open issues and future work.

2 About AOP4ST

AOP4ST is a framework process, it is not a method nor a methodology. It covers the whole SDLC, but we are presenting here its structure for the early phases, commonly known as "early aspects" [2][3].

AOP4ST's name highlights two main concepts: a) the AOP, "Aspect-oriented Process", indicates that it is truly aspect-oriented, ensuring that can be reached the widely known benefits of this paradigm; b) the 4ST, "for a Smooth Transition", points to the possibility of applying this process in the industry immediately, because it employs widespread techniques, notations, standards, tools, etc. and allows to move to an aspect-oriented reality, taking advantage of the current state of the paradigm, until their own tools, techniques, etc. were imposed and completely accepted on the market.

The problem that AOP4ST tries to solve is how to bring the benefits of the aspect-oriented paradigm to the whole SDLC at the same time that are being used techniques, tools and standards currently widespread in the industry. In addition, the use of well-known techniques and tools allows to incorporate this paradigm gradually, until the different existing proposals have sufficient diffusion and maturity to warrant their employment in real and complex projects.

The whole SDLC include the business model, not considered for the authors offering aspect-oriented approaches for the early stages of the SDLC. There are few incomplete proposals about aspect-oriented business modeling [4].

Our approach arises from several factors, which we have to face in the industry and in the adoption of new technologies for software development.

First, it is well known that the different software development paradigms initially appear in the programming phase and then continue their definitions upstream, along the SDLC [5][6]. The aspect-oriented paradigm follows the same pattern, that is why

we can find more proposals for the programming phase than for the early phases of the SDLC.

Second, many proposals about software development sound promising and offer benefits difficult to refuse, but their massive use in industry depends on many factors. A well-known case is that of the object-oriented databases, that beyond the benefits they offered and the enormous popularity of the object-oriented languages and development tools today, they have not at all achieved the leading role in industry that could be expected [7].

Finally, software development projects have to deal with many risks, and the main function of project leaders is to minimize the damage that these risks can cause. The use of immature technologies, tools newcomers to the market, techniques that have not been tested enough, etc., would be very risky decisions to take by who has the responsibility to carry out a successful software development project. On the other hand, the availability of well-known tools and techniques and the adherence to standards and best practices will help professionals to make good estimates and to take better decisions.

AOP4ST is based on the hypothesis that it is possible to design an aspect-oriented software development process that encompass techniques, tools, notations and standards of widespread use in the current practice. This development process is suitable for the early stages of the SDLC, making the most with the benefits of the aspect-oriented paradigm in real-world settings. Under certain circumstances, it is possible to make use of existing techniques, tools and standard notations, right now, while specific theoretical and practical instruments are developed and introduced in the market, achieving enough dissemination and support to justify its use in real software development projects.

AOP4ST's basic structure for early aspects is composed of three models: business model, user requirements model and software requirements model. The last one is divided into three views: functional, static and state views. Concerns can be progressively discovered along these models and views.

3 The business model in AOP4ST

The business model starts the process in AOP4ST, since we seek to have it as the first layer of the enterprise architecture [8]. In an enterprise architecture, this layer has to pull on the rest of the models downstream in order to meet the business goals.

Besides this, an early concern detection can help to improve the software robustness. If concerns are detected, separated and encapsulated away from the elements of the problem domain, we may obtain:

1. Cleaner business models, since they are not tangled with unrelated issues to the problem domain.
2. Business models totally focused on the needs of the very specific business processes.
3. Crosscutting concerns modeling and encapsulating specific processes not belonging to the problem domain.

4. The possibility to focus each analyst on each party's specific goals: the functional perspective is modeled separately of the quality perspective for the business models.

Five main aspect-oriented activities must be done in AOP4ST's business model: concern detection, concern separation and modeling, composition rules checking, conflict resolution, aspect-oriented modeling.

3.1 Concern detection

The business model can be structured in several ways. A typical way is to divide it into primary processes, support processes and management processes [9], and besides to them, the reusable processes. The latter are processes that are not instantiated in themselves, but are done from any of the first kinds of processes and can be shared. Regardless of the type of process to which they belong, each process must be located within a specific package. These packages correspond to the concerns that are detected in this first model, and will host the concerns throughout the whole SDLC.

Reusable processes can be of two types: belonging to the domain of the problem and not belonging to it. In the first case, these are typical activities of the domain of the problem, that are repeated in several processes and which we, normally, could associate with functional requirements.

In the second case, these are activities that are independent of the problem domain and can be found even in different problem domains, e.g. access control, security, audit, logging, etc. They are, typically, quality attributes, also known as non-functional requirements (Figure 1), that are tangled with the activities belonging to the problem domain and scattered along all the processes.

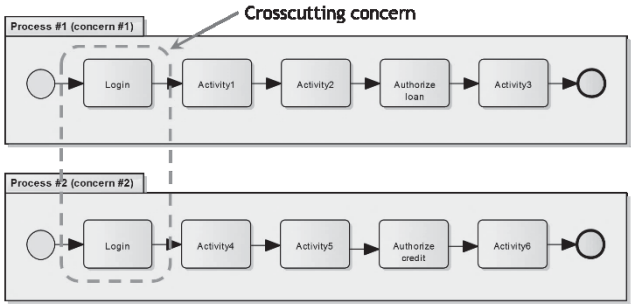

Fig. 1. Tangled and scattered crosscutting concerns.

Concern detection is done with the deliberate intention of doing so. In order to achieve a better success, we try to detect separately, the concerns corresponding to functional requirements and to non-functional ones. The former concerns are more difficult to be detected, because they depend on the wording of the modeler. The latter are simpler, because they are clearly distinguishable from the activities belonging to the problem domain and, besides, it is possible to have a list of standard categories of non-functional requirements to follow in a systematic way. Manual or automated aspect mining techniques can be used in order to detect concerns [10].

3.2 Concern separation and modeling

Concern separation is done by placing the detected ones in a zone of reusable processes in the model, so as to be able to form a catalog of reusable processes that is accessible by all modelers, in a kind of catalog of "reuse with" resources.

Concerns must be encapsulated into packages, along with the rest of the common elements of the model, and using a notation based on the proposal of Charfi et al. [11] although adjusted to use only elements belonging to the standard BPMN 2.0. This notation is also used to specify pointcuts, that are conditions that explicitly indicate join points. Join points are the points were the concerns will be composed again. In the base process, we use an annotation element to indicate the join point (Figure 2) and the separated concern is modeled inside of a pool element. This pool includes a "Proceed" activity, that represents the join points and that indicates if the composition must be done before, after or around the join point (Figure 3).

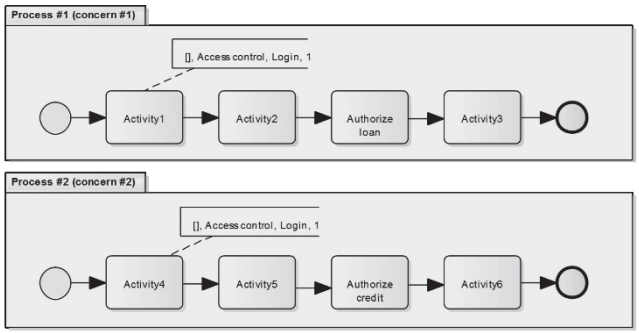

Fig. 2. Pointcut represented with annotation elements and indicating the join points.

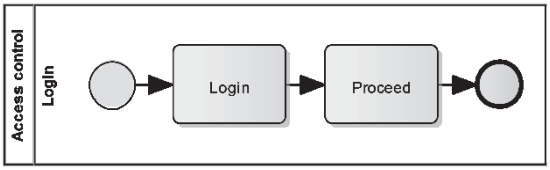

Fig. 3. Concern modeled and encapsulated, with a "Proceed" activity indicating the join points.

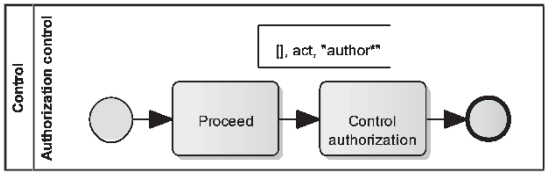

Fig. 4. Implicit concern composition.

We have incorporated a notation to indicate join points implicitly and to add concerns into the processes whenever a particular condition is met. The concern

presented in Figure 4 will be composed after all the activities called “Control authorization” in the model of Figure 2.

3.3 Composition rules checking and conflict resolution

When concerns are separated, we must be sure that they will be able to be re-composed correctly. The evolution of models over time could also produce new concerns that, at the time of composing, might not be adequately integrated.

To improve the concern composition, we have designed a set of composition rules that allows to reduce possible errors when integrating again the concerns into the processes [12]. We have analyzed the possible consequences of integrating the different types of concerns (“after”, “before”, and “around”) with the different elements of BPMN 2.0, and produced a set of recommendations that lead to a syntactically correct composition. This will also help to improve the semantics of the processes and to reduce potential conflicts.

3.4 Aspect-oriented modeling

Since base processes and concerns are modeled within packages, the composition relationships and the relationships among concerns can be represented with a “concern model”, build with UML package diagrams (Figure 5). In addition, it is possible to present a more detailed application of the concerns in the different join points by means of a “join point model”, where the packages are presented as "white boxes", showing the join points inside (Figure 6).

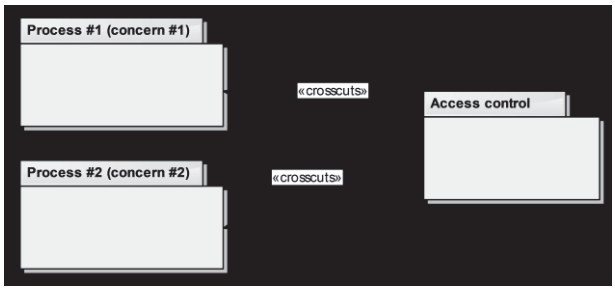

Fig. 5. Concern model.

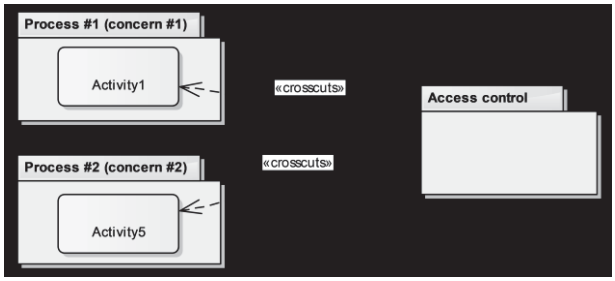

Fig. 6. Join point model.

4 The user requirements model in AOP4ST

The business model and the user requirements model crosscut the systems belonging to the organization. Each business process can describe activities that are supported by different systems. Similarly, the implementation of a user requirement could impact several systems, so this model of user requirements does not belong to a particular system but to the global solution.

In the business model, the processes were described placing them into specific packages that correspond to concerns. The same packages existent in the business model are re-created in this model of user requirements, so that the user requirements that are now detected are perfectly delimited to the process that requires them.

Functional user requirements will be easier to locate within a specific concern, while non-functional user requirements and business rules are more likely to be global, that is, to apply to several or even all concerns. They are usually referred to as crosscutting concerns. In the business model, several crosscutting concerns had already been detected when we modeled processes not belonging to the domain of the problem, but in this model, will arise new crosscutting concerns (Figure 7). These concerns also have an important influence among them, due to the positive and negative contribution relationships [13].

In addition to modeling the user requirements in the corresponding packages, it will be necessary to specify the relationships among them, so that the aspect-oriented models mentioned in the previous section (concern and join point models) are used again, now including the relationships among user requirements.

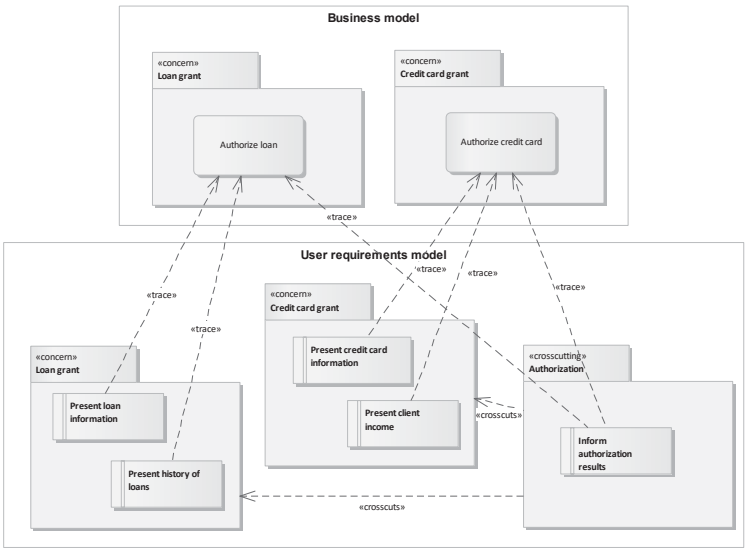

Fig. 7. Traceability between business and user requirements models.

5 The requirement model in AOP4ST

When implementing a user requirement, the solution could impact one or more computer systems. Thus, it will be necessary to divide this model of system requirements for each of the computer systems that are involved in the solution. We must develop a "system requirements model", or simply "requirement model", for each of the computer systems, and divide it into three views: functional, static and state views.

5.1 Requirement model: functional view

The functional view is elaborated using the use case technique, mainly based on the proposal of Jacobson and Ng [14], with some adjustments that we have found necessary, especially in the specification of the use cases, as well as in their models, in order to be able to automatically create test cases from the scenarios.

Again, we must create the same packages that exist in the previous model, which correspond to the concerns found so far, to place within them the use cases that we will find in this model. Since each requirement of the previous model might require an implementation in some of the computer systems of this model, we will only create the packages for the computer systems that will be impacted by the solution. If the solution of each user requirement could involve more than one use case, we will create a new package for each use case. It is important to keep in mind that each use case corresponds to a concern.

5.2 Requirement model: static view

In the static view of the requirements model we will maintain the division of computer systems and create a package for each of the packages in the use case view. In this way, we will build a logical class diagram for each use case by using UML.

Clarke y Baniassad [15] show how to elaborate these diagrams and how to compose the final classes later on. We have extended these models and incorporated practices that allow better traceability among models, for purposes of impact analysis for maintenance, regression testing, etc.

5.3 Requirement model: state view

With this third view, we consider that it will be possible to have a complete model of the solution. Since the state diagrams describe the life cycle of objects belonging to a single class, there is no need of creating the package structure again in this view, since

the state diagrams will be appended to their respective classes. We use the UML notation to build the state diagrams.

A state diagram presents the potential states of the objects of a class and their possible transitions. The state of an object is the abstraction of the values of its attributes and their relationships at a given time, any changes in the attributes and relations of a class in the static view can cause a strong change in its corresponding state diagram. We have included the "revised" tagged value for all the states that should change to "false" when a state could be affected for a change on the static view.

6 Conclusions

This article presents the main ideas about AOP4ST, a framework process for aspect-oriented software development, focusing on the early stages of the SDLC. With this approach, we expect to obtain a double gain: to reduce the negative impact of employing a paradigm that is not yet mature enough on real projects in the industry and, to take advantage of the benefits of the current state of the aspect-oriented paradigm.

The main virtue of this proposal lies in the use of widely disseminated standards in the industry, and with good practices, software tools, suppliers, human resources, etc. that are enough to allow to face a software development project in real-world settings.

Another very important outcome is the homogeneity and cohesion among the models, which allow for a coherent transition from one to another, enabling pre and post-requirement specification traceability and impact analysis.

The third outstanding feature is that concerns are emerging naturally and progressively throughout the models.

So far, we have been able to perform some theoretical and practical validations of AOP4ST. From the theoretical point of view, we have submitted it for consideration to a symposium of doctoral theses [16], from which we have received very rich feedback. We have also applied the criteria established by Jalali [4] for the measurement of AOP4ST's business model, and the results placed it in a privileged position.

Regarding the practical validation, we were able to test separately the AOP4ST's models in several companies. AOP4ST was fully applied to re-modeling a whole model of one of the most important biochemical laboratories in Argentina.

We believe that there is still a lot of effort needed to validate AOP4ST and to achieve a greater maturity in the architectural and test models. We are now working on these issues.

For Spanish-speaking readers, it is possible to find more information about AOP4ST in [16] and [17], and about AOP4ST's business model in [18] and [19].

References

1. Magableh, A., Shukur, Z. and Ali, N. M. "Systematic review on aspect-oriented UML modeling: A complete aspectual UML modeling framework," J. Appl. Sci., vol. 13, no. 1, pp. 1–13 (2013).
2. Bakker, J., Tekinerdogan, B. and Aksit, M. "Characterization of Early Aspect Approaches," Early Asp. Asp. Requir. Eng. Archit. Des. Work., p. 7 (2005).
3. Rashid, A., Moreira, A. and Tekinerdogan, B. "Early aspects: aspect-oriented requirements engineering and architecture design," Software, IEE Proc., vol. 151, no. 4, pp. 153–155 (2004).
4. Jalali, A. "Assessing Aspect Oriented Approaches in Business Process Management," in Perspectives in Business Informatics Research, 13th International Conference, BIR 2014, pp. 231–245 (2014).
5. Capretz, L. F. "A brief history of the object-oriented approach," SIGSOFT Softw. Eng. Notes, vol. 28, no. 2, p. 6– (2003).
6. "History of Programming Languages," in History of Programming Languages Conference.
7. Leavitt, N. "Whatever Happened to Object-Oriented Databases?" Computer (Long. Beach. Calif)., vol. 33, no. 8, pp. 16–19 (2000).
8. Greefhorst, D. and Proper, E. Architecture Principles. The Cornerstones of Enterprise Architecture, vol. 6, no. 3 (2011).
9. ABPMP, BPM CBOK V.3.0 - Business Process Management BPM Common Body of Knowledge (2013).
10. Pinciroli, F. "Considerações acerca da mineração de aspectos," Perspect. em Ciências Tecnológicas, vol. 5, no. 5, pp. 83–101 (2016).
11. Charfi, A., Müller, H. and Mezini, M. "Aspect-oriented business process modeling with AO4BPMN," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6138 LNCS, pp. 48–61 (2010).
12. Pinciroli, F. "Aspect-oriented business process composition rules in AOP4ST," in 35th International Conference of the Chilean Computer Science Society (SCCC 2016) held in conjunction with the 42th Latin American Computing Conference (CLEI 2016), pp. 1–6 (2016).
13. Pinciroli, F. "Improving software applications quality by considering the contribution relationship among quality attributes," Procedia Comput. Sci. 3rd Int. Work. Comput. Antifragility Antifragile Eng. (ANTIFRAGILE 2016), vol. 83, pp. 970–975 (2016).
14. Jacobson, I. and Ng, P. Aspect-oriented software development with use cases. Addison-Wesley (2005).
15. Clarke, S. and Baniassad, E. Aspect-oriented analysis and design. The Theme approach. Boston: Addison-Wesley (2005).
16. Pinciroli, F. "Aspect-Oriented Process for a Smooth Transition," in Ph.D. Symposium of the IEEE 11 Congreso Colombiano de Computacion (2016).
17. Pinciroli, F. "AOP4ST – Aspect-Oriented Process for a Smooth Transition," in WICC 2015 - XVII Workshop de Investigadores en Ciencias de la Computación (2015).
18. Pinciroli, F. and Zeligueta, L. "El modelo de negocio en AOP4ST," in WICC 2016 - XVIII Workshop de Investigadores en Ciencias de la Computación (2016).
19. Pinciroli, F. and Zeligueta, L. "Modelado de negocios orientado a aspectos con AOP4ST," in WICC 2017 - XIX Workshop de Investigadores en Ciencias de la Computación (2017).