Towards a Process to Design Aspect-Oriented Reference Architectures

Elisa Yumi Nakagawa, Rafael Messias Martins,
Katia Romero Felizardo, and José Carlos Maldonado

Dept. of Computer Systems
University of São Paulo
São Carlos, SP, Brazil
{elisa, rafaelmm, katiarf, jcmaldon}@icmc.usp.br

Abstract. Since software architecture design is essential in the software development, attention has been taken to this task. There are, however, little initiatives to systematize the reference architecture design, in special, when these architectures include aspects (from AOSD - Aspect-Oriented Software Development) in their structures. This paper presents ProSA-RA, a process to build aspect-oriented reference architectures, systematizing the development, design and evaluation of these architectures. In order to illustrate the feasibility of this process, we illustrate its application in the Visual Mining domain. Positive results have been achieved, mainly related to facilities to build reference architectures and to identify aspects in architectural level.

Keywords: reference architecture, aspect-oriented software development, visual mining

1 Introduction

Software architectures play a major role in determining system quality (e.g. performance, maintainability, and reliability), since they form the backbone for any successful software-intensive system [1, 2]. Thus, decisions made at the architectural level directly enable, facilitate, hamper, or interfere with the achievement of business goals as well as meeting functional and quality requirements. Essentially, software architecture is the structure or structures of the system which comprise software elements, the externally visible properties of those elements, and the relationships among them [3]. In this context, reference architectures for different domains — for instance, service-oriented systems, embedded systems, e-commerce, web browsers, among others — have emerged. A reference architecture captures the essence of the architectures of a collection of systems of a given domain. The purpose of a reference architecture is to provide guidance for the development of architectures for new systems or extended systems and product families. In other words, they can be seen as a knowledge repository of a given domain. Considering the relevance of software architectures, approaches

to design adequate software architectures have been proposed [4, 5]. Regarding to reference architectures, initiatives can be also found [6, 7].

In another perspective, Aspect-Oriented Programming (AOP) has arisen, supporting a better SoC (Separation of Concerns) and reflecting more adequately the way developers think about the system [8]. Essentially, this approach introduces a unit of modular implementation — the aspect — that has been typically used to encapsulate crosscutting concerns (i.e. concerns spread across or tangled with other concerns). Modularity, maintainability and facility to write software can be achieved with AOP [9]. Aspects have also been explored in the early life cycle phases, including architectural design [10–12]. According to Baniassad [13], aspects in early phases support easier identification and analysis of aspects during later activities and, as a consequence, a better SoC in the systems. Additionally, positive results exploring the use of aspects in reference architectures can also be found [14]. Although approaches to design software architectures and reference architectures have been proposed, there is a lack of work that systematize specifically the design of aspect-oriented reference architectures, i.e. architectures that include aspects (called architectural aspects[1] in this paper) in their structure.

In this scenario, we have proposed a process, named ProSA-RA, to design aspect-oriented reference architectures. Thus, in this paper, we have two main goals: (i) to present ProSA-RA, giving emphasis in how to deal with architectural aspects, as well as how to represent and evaluate reference architectures, and; (ii) to illustrate the use of ProSA-RA in the Visual Mining domain, in order to perform a critical evaluation of the utility of this process with respect to facility of creating aspect-oriented reference architectures. It is worth highlighting that ProSA-RA is part of a more comprehensive software development process centered in reference architectures.

The remainder of this paper is organized as follows. In Section 2 related work are presented. In Section 3 we present ProSA-RA. In Section 4 we illustrate the use of ProSA-RA to establish a reference architecture for Visual Mining domain. In Section 5 we discuss about achieved results. In Section 6 we summarize our contributions and discuss perspectives for further work.

2 Related Work

From the first work of Kruchten on iterative software development with a focus on software architecture [15], a number of work has recognized the value of considering explicitly software architectures in the system development processes [3, 5]. In this scenario, works that encompass the software architecture design have been proposed, for instance, Attribute-Driven Design (ADD) [3] and Rational Unified Process 4+1 View [16]. Regarding to reference architecture design, Muller [7] has proposed recommendations in order to create and maintain

[1] An architectural aspect refers to an element that crosscuts other architectural elements, for instance, packages and components, or even other architectural aspects.

reference architectures; basically, references architecture must be understandable, up-to-date and maintainable. In the context of product line development, Bayer [6] presents PuLSE-DSSA (Product Line Software Engineering - Domain-Specific Software Architecture), a systematic approach to define reference architectures capturing knowledge from existing system architectures. Other work have pointed out the need of formalizing processes to design reference architectures [17], since informal processes have still been used. Furthermore, processes that support the design of aspect-oriented reference architectures are not also found. This has therefore motivated the conduction of our work.

3 ProSA-RA

ProSA-RA is a process that systematizes the development, design and evaluation of aspect-oriented reference architectures. ProSA-RA is result of our experience in establishing aspect-oriented reference architectures for software engineering domain [18, 19]. The outline structure of ProSA-RA is illustrated in Figure 1. In short, to establish reference architectures by using ProSA-RA, information sources are firstly selected and investigated (in Step RA-1) and architectural requirements[2] are identified (in Step RA-2). After that, an architectural description of the reference architecture is established (in Step RA-3) and the evaluation of this architecture is conducted by using a checklist inspection approach (in Step RA-4). Analysts, software architects and domain experts are involved and conduct these steps. In more details, the steps of ProSA-RA are:

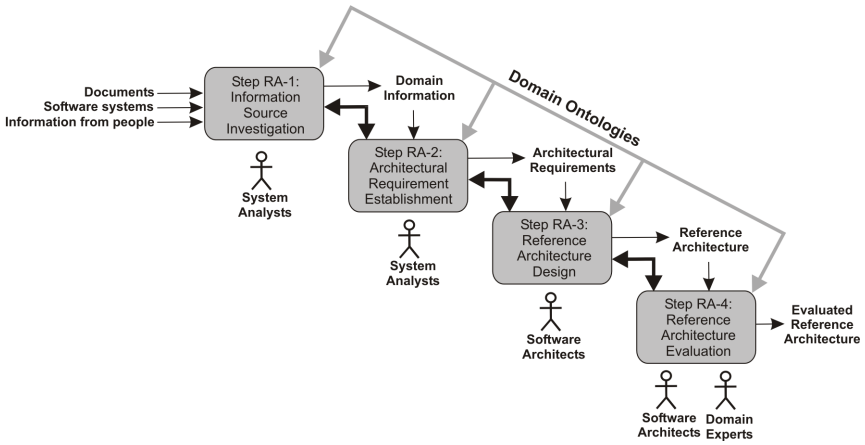

Fig. 1. Outline Structure of ProSA-RA

[2] An architectural requirement refers to requirement of a reference architecture of a given domain and describes common functionalities and configurations presented in systems of that domain.

Step RA-1: Information Source Investigation: In this step, the main sources of information are selected. These sources must provide information about processes, activities and tasks that must be automated to that domain. In general, these sources are more comprehensive, since reference architectures are basis of a set of systems of a specific domain. We have selected those more relevant: (i) people: customers, users, researchers, domain experts and developers of systems of the domain are important information sources. Interviews, questionnaires and other requirements elicitation techniques can be used. Needs and limitations must be identified; (ii) systems: main systems of the domain are investigated through their use (when available) and related documentation. Their architectures are also investigated, observing the evolution and adaptation capability; (iii) publications/documents: information related to processes/activities/tasks and system architectural model of the given domain are identified; and (iv) ontologies: they are particularly important to support this step, since they represent the knowledge domain — i.e. concepts/terms of the domain and relationships among them — in a well-structured format. Furthermore, Systematic Reviews can be applied as a means of evaluating and interpreting all available research relevant to a particular question, topic area, or phenomenon of interest, using a trustworthy, rigorous, and auditable methodology [20]. As a result, a set of information sources are arisen. In this step, there is not a concern if architecture is aspect-based or not; this concern is treated in the next steps.

Step RA-2: Architectural Requirement Establishment: Based on selected sources, information related to domain are arisen, resulting in a set of requirements of the reference architecture, as well as related concepts that must be considered in the architecture. For this, four main tasks are conducted: (i) requirements of systems of the domain are identified. These requirements reflect the processes/activities/tasks that must be automated by systems; (ii) based on those system requirements, a set of architectural requirements is established. Probably more than one system requirement will be aggregated in an architectural requirement. It is observed that architectural requirements are more comprehensive than the system requirements, since they describe the requirements of a set of systems of the domain; (iii) architectural requirements are mapped in domain concepts, aiming at going toward an architectural design. Ontologies, when available, can be used to perform this task, since they usually embody all domain concepts; and (iv) concepts are classified in crosscutting or non-crosscutting, intending to identify architectural aspects. In general, concepts related to many requirements or related to non-functional requirement have crosscutting characteristic. This previous categorization makes easy the posterior task of designing the aspect-oriented reference architecture.

Step RA-3: Reference Architecture Design: The effective reuse of knowledge contained in reference architectures depends not only on raising the domain knowledge, but also documenting efficiently this knowledge through an adequate architectural description. To build this description, well known architectural styles, software patterns and approaches are investigated and

used as basis to organize the concepts identified on the previous step. For instance, if three-tier architecture[3] is used, the identified concepts are organized in the application layer. A special treatment must be taken to the concepts marked as crosscutting. They will be considered architectural aspects and, therefore, must be represented using an adequate notation.
In the previous work [21], we selected architectural views — module, runtime, deployment and conceptual views — and UML 2.0 to describe reference architectures. Besides that, when designing an aspect-oriented reference architecture, aspects must be adequately represented, as presented in another previous work [22]. Thus, architectural views, UML techniques and required extensions to represent aspect-oriented reference architectures are:

- Module view: it shows the structure of the software in terms of code units. Packages, classes, containment, specialization/generalization, and dependency relations can be used to represent this view. Thus, UML class diagram is an adequate technique. In this diagram, when aspects are used to encompass architectural aspects, we have used the stereotype <<crosscuts>> to represent the dependency relation between the aspect and the elements affected by this aspect;
- Runtime view: it shows the structure of the system when it is executing. Components, provided and required interfaces, packages, ports, and connectors are used. Interfaces of modules that encapsulate architectural aspects are obviously different from the provided and required interfaces of modules composed only by objects. We proposed a filled circle, labelled with the name of the interface, attached by a solid line to the modules that encapsulate an architectural aspect in order to represent the interface of them. We named this type of interface as Interface Made by Aspects (IMA) [22]. We have also proposed a half-square, attached by a solid line to the module that is affected by modules that encapsulate an architectural aspect. Differently from provided interface, IMA represents the characteristics that modules must have to use the modules that encapsulate an architectural aspect. UML component diagram can be used to represent this view;
- Deployment view: it describes the machines, software that is installed on that machines and network connections. An adequate technique to represent this view is UML deployment diagram. Considering its higher abstraction level, architectural aspects are not required to be explicitly represented in this view; and
- Conceptual view: it aims at describing and supporting understanding of each concept of the domain that is used in the reference architecture, since other views do not present this property. For describing this view, ontologies, controlled vocabularies, taxonomies, thesauri, concept maps, among others can be used. Regarding to ontology, it basically consists of concepts and relations, as well as their definitions, properties and

[3] http://www.sei.cmu.edu/str/descriptions/threetier.html

constraints expressed by means of axioms [23]. In our work, ontologies have been explored to represent this view [18].

As a result of this step, a set of architectural views composing the description of the aspect-oriented reference architecture is created.

Step RA-4: Reference Architecture Evaluation: Checklist inspection approach is used in order to evaluate the quality of the reference architectures. A checklist corresponds to a list of questions that guides reviewers on detecting defects in documents and, specifically in our work, defects in documents related to reference architecture design. Ontologies are important to guide the evaluation, since this step involves, besides domain experts, software architects that do not have deep knowledge about the application domain. By using this checklist, we have evaluated quality characteristics: maintainability, performance, security, usability, portability and reuse. We also evaluate the architectural description itself, through identification and elimination of defects related to omission, ambiguity, inconsistency, as well as strange and incorrect information. As a result, a more reliable architecture can be achieved.

4 Case Study on the Visual Mining Domain

Nowadays, a large quantity of information — images, videos and text, for example — is available digitally. In their original form, it is impossible to effectively analyze such a quantity of data; that is why information extraction has been a challenge for many research areas, such as data mining, visualization and others. On Visual Mining, the original data is processed and transformed with the use of mining algorithms and then presented to the viewers; viewers can then interact with it in order to create and validate a hypothesis about the data [24]. Even though software tools are very important for the effective application of visual mining techniques, as they rely heavily on good graphical representations and proper user interaction, these tools have almost always been individually and independently implemented, relying on tailor-made architectures. In this context, a reference architecture could be used to aggregate knowledge about the Visual Mining domain aiming at facilitating the building of new tools, as well as of a product family. We intended then to establish a reference architecture for that domain, exploring the possible advantages that aspects can provide to architectural level and, as a consequence, to target systems. Thus, we conducted the steps of ProSA-RA:

Step RA-1: Information Source Investigation: This first step consisted on putting together a list of the most relevant information sources among all that was available. This resulted in: (i) three visual mining researchers and five tool developers; (ii) six visual mining tools, with available documentation and source code; and (iii) 35 important publications: three books, six technical reports and 26 scientific papers. These sources seemed sufficient to extract domain knowledge, although ontologies for the Visual Mining domain are not available yet.

Step RA-2: Architectural Requirement Establishment: Each source listed on the previous step was consulted in the following way: (i) the researchers and developers were interviewed and asked about features and attributes they know and also about what they would desire on a visual mining tool; (ii) by analyzing the tools' user interfaces — menus, buttons and features provided — it was possible to extract information about what they do and what they should do; (iii) documentation and source code did not help sufficiently to the requirements extraction, since the best practices were often not used; and (iv) publications were important to gather requirements, specially those that presented descriptions of visual mining tools and domain reference models. During this first stage, many fine-grained *System Requirements* were gathered, which were then analyzed to derive the more coarse-grained *Architectural Requirements*. This is partially illustrated in Table 1. We can see the relationship between *System Requirements*, which were extracted directly from the information sources, and *Architectural Requirements* derived from them. Many *System Requirements* were often grouped into a single *Architectural Requirement*.

Table 1. Deriving Architectural Requirements from System Requirements

N.	System Requirement	Source	Architectural Requirement
3	...	1	...
4	Accept Image Collections as Input	2	Accept Different Data Types as Input
5	Accept Text Collections as Input	2	Accept Different Data Types as Input
6	Accept Video Collections as Input	2	Accept Different Data Types as Input
7	Save Visualization Settings for Future Re-running	3	Persistence of the Pipeline's Intermediate Objects
8	...	3	...

In Table 2, it is shown the relationship between *Architectural Requirements* and *Concepts* derived from them. Similarly to the previous table, often many *Architectural Requirements* related to the same *Concept*. *Concepts* are also classified as crosscutting or not; one example is the *Persistence* concept, which is important so that multiple visualizations can be created from the same — or slightly altered — settings.

Table 2. Deriving Concepts from Architectural Requirements

N.	Architectural Requirement	Concept	Crosscutting (Yes/No)
1	Accept Different Data Types as Input	Data Acquisition	No
2	Accept Data from Different Sources	Data Acquisition	No
3	Persistence of the Pipeline's Intermediate Objects	Persistence	Yes
4	...	...	...

Step RA-3: Reference Architecture Design: From the *Architectural Requirements* and the *Concepts* defined on the previous step, the architectural design was a quite straight-forward task. Architectural styles that matched the

requirements were studied and selected, and the four proposed architectural views were built. For the sake of space, only the module view is presented herein and showed in Fig. 2. Model-View-Controller (MVC) was used to ensure that derived systems will be able to deliver diverse rich and powerful graphical user interfaces. Common concepts were encapsulated in modules. For the sake of space, only some decisions are explained. For instance, `Data Acquisition` (in Table 2) was encapsulated in the `DataAcquisition` module (in Figure 2). On the other hand, crosscutting concepts were encapsulated in architectural aspects and <<crosscuts>> relations were used to indicate the relationship between these aspects and other modules. For example, `Persistence` concept in Table 2 is represented as an architectural aspects in Figure 2. Another example is `Documentation` (in Figure 2) related directly to domain functional requirements that was also found as an architectural aspects. Currently, the evaluation of this architecture is in progress. RefVM will be also used to design applications on the Visual Mining domain.

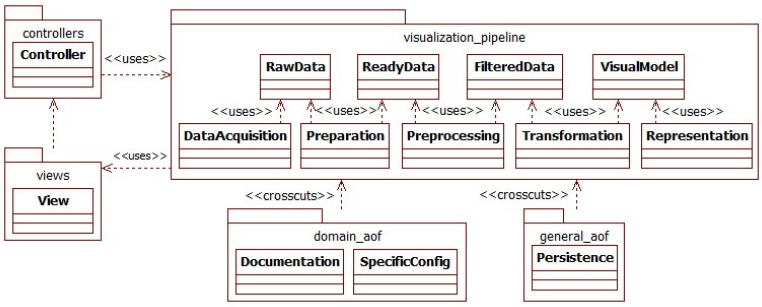

Fig. 2. Module View of the Visual Mining Reference Architecture

5 Discussion

The design of aspect-oriented reference architecture using ProSA-RA has showed to be useful in two main perspectives: (i) ProSA-RA supports organization of domain knowledge, systematizing the architecture establishment. It encompasses steps typically found in approaches to design software architectures: requirement identification, creation, representation and evaluation of the architecture. Moreover, differently from these approaches, ProSA-RA has the concern of dealing with aspects along all steps; (ii) An informal qualitative analysis provides evidences about a better modularization in the reference architecture, since ProSA-RA supports identification of architectural aspects. In other words, a better SoC seems to be achieved. SoC improves software reuse, comprehensibility, component integration and high impact of change in software systems and, therefore, it

is also desired in reference architectures. The case study presented in this paper has pointed out to these perspectives. We have also found these same observation in two other aspect-oriented reference architectures that we have built: RefASSET [19] and RefTEST [18].

However, even using ProSA-RA, the establishment of aspect-oriented reference architectures is a complex activity, requiring a deep knowledge on diverse subjects related to software development. This difficulty occurs mainly due to the fact that the understanding of the domain and generalization of domain elements are hard to be conducted. Another difficulty is to identify the architectural aspects. Some traditional architectural aspects, such as persistence, are easy to identify; however, experience is required to identify aspects related to application domain. Furthermore, there is still a gap between the reference architecture and implementation. Thus, activities to instantiate the reference architecture in order to obtain architectural instances must be conducted. For this, specific information about the system to be developed must be arisen, conducting, for instance, requirement identification and analysis/design.

6 Conclusion

Considering the relevance of reference architectures to the productivity of software systems, the main contribution of this work is ProSA-RA, a process that systematizes the building of aspect-oriented reference architectures. Special attention has been taken to the identification and representation of aspects, aiming at achieving a full potential of early aspects in the architecture level. Furthermore, ProSA-RA provides mechanisms to the representation and evaluation of reference architectures, considering readability and understandability of these architectures as fundamental to their dissemination, reuse and to posterior evolution. Results of our investigations, such as the case study presented in this paper, have pointed out that ProSA-RA supports organization of domain knowledge and seems to provide a better SoC in architectural level. As future perspectives, we intend to apply ProSA-RA in different domains. Furthermore, there is a number of interesting directions for future work. One of them is to investigate how this process can contribute in a wider software development approach, such as software product line. Another research line is to investigate the difficulties to automate ProSA-RA, aiming at achieving more productivity and reliability in the resulting architectures.

Acknowledgments: This work is supported by Brazilian funding agencies (FAPESP, Capes and CNPq), QualiPSo European research project (Grant: IST- FP6-IP-034763) and INCT-SEC Project (Processes: 573963/2008-8 and 08/57870-9).

References

1. Kruchten, P., Obbink, H., Stafford, J.: The past, present, and future for software architecture. IEEE Software **23** (2006) 22–30

2. Shaw, M., Clements, P.: The golden age of software architecture. IEEE Software **23** (2006) 31–39
3. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley (2003)
4. Bass, L., Kazman, R.: Architecture-based development. Technical Report Technical Report CMU/SEI-99-TR-007, SEI, Pittsburgh, USA (1999)
5. You-Sheng, Z., Yu-Yun, H.: Architecture-based software process model. ACM SIGSOFT Software Engineering Notes **28** (2003) 1–5
6. Bayer, J., Forster, T., Ganesan, D., Girard, J.F., John, I., Knodel, J., Kolb, R., Muthig, D.: Definition of reference architectures based on existing systems. Technical Report 034.04/E, Fraunhofer IESE (2004)
7. Muller, G.: A reference architecture primer. [*On-line*], *World Wide Web* (2008) Available: http://www.gaudisite.nl/ (Access: 06/19/2009).
8. Kiczales, G., Irwin, J., Lamping, J., Loingtier, J., Lopes, C., Maeda, C., Menhdhekar, A.: Aspect-oriented programming. In: Proc. of the 11th Eur. Conf. on Object-Oriented Programming, Jyväskylä, Finland (1997) 220–242
9. Laddad, R.: Aspect-oriented programming will improve quality. IEEE Software **20** (2003) 90–91
10. Ivers, J., Clements, P., Garlan, D., Nord, R., Schmerl, B., Silva, J.R.O.: Documenting component and connector views with UML 2.0. Technical report (2004) CMU/SEI-2004-TR-008.
11. Keuler, T.: An aspect-oriented approach for improving architecture design efficiency. In: Companion of the ICSE'08. (2008) 1007–1010
12. Navarro, E., Letelier, P., Ramos, I.: Requirements and scenarios: Running aspect-oriented software architectures. In: WICSA'07, Washington, USA (2007) 23
13. Baniassad, E., Clements, P.C., Araujo, J., Moreira, A., Rashid, A., Tekinerdogan, B.: Discovering early aspects. IEEE Software **23** (2006) 61–70
14. Greenwood, P., Surajbali, B., Coulson, G., Rashid, A., Lagaisse, B., Truyen, E., Sanen, F., Joosen, W.: Reference architecture for aspect-oriented middleware (version 3.0). Technical report, AOSD-Europe (2008) AOSD-Europe deliverable D103.
15. Kruchten, P.: An iterative software development process centered on architecture. In: Proc. 4ème Congrès de Génie Logiciel. (1991) 369–378
16. Kruchten, P.: The Rational Unified Process: An Introduction. 3 edn. The Addison-Wesley Object Technology Series. Addison-Wesley (2003)
17. Eklund, U., Örjan Askerdal, Granholm, J., Alminger, A., Axelsson, J.: Experience of introducing reference architectures in the development of automotive electronic systems. SIGSOFT Softw. Eng. Notes **30** (2005) 1–6
18. Nakagawa, E.Y., Simão, A.S., Ferrari, F., Maldonado, J.C.: Towards a reference architecture for software testing tools. In: SEKE'2007, Boston, USA (2007) 1–6
19. Nakagawa, E.Y., Maldonado, J.C.: Architectural requirements as basis to quality of software engineering environments. IEEE Latin America Trans. **6** (2008) 260–266
20. Kitchenham, B.: Procedures for performing systematic reviews. Technical Report TR/SE-0401, Keele University (2004)
21. Nakagawa, E.Y., Maldonado, J.C.: Reference architecture knowledge representation: An experience. In: SHARK'2008 at ICSE'2008, Germany (2008) 1–4
22. Nakagawa, E.Y., Maldonado, J.C.: Representing aspect-based architecture of software engineering environments. In: AArch'07 at AOSD'07, Canada (2007) 1–4
23. Uschold, M., Grüninger, M.: Ontologies: principles, methods, and applications. Knowledge Engineering Review **11** (1996) 93–155
24. Keim, D.: Information visualization and visual data mining. IEEE Transactions on Visualization and Computer Graphics **8** (2002) 1–8