Extraction of Component-Based Architecture from Object-Oriented Systems

Conference Paper · March 2008

DOI: 10.1109/WICSA.2008.44 · Source: IEEE Xplore

CITATIONS
35

READS
221

4 authors:

Sylvain Chardigny
MGPS
17 PUBLICATIONS **148** CITATIONS
SEE PROFILE

Abdelhak Seriai
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LI…
113 PUBLICATIONS **811** CITATIONS
SEE PROFILE

Mourad Oussalah
University of Nantes
316 PUBLICATIONS **1,189** CITATIONS
SEE PROFILE

Dalila Tamzalit
University of Nantes
102 PUBLICATIONS **380** CITATIONS
SEE PROFILE

Extraction of Component-Based Architecture From Object-Oriented Systems

Sylvain Chardigny, Abdelhak Seriai
Ecole des Mines de Douai
941 rue Charles Bourseul
59508 Douai France
{chardigny,seriai}@ensm-douai.fr

Mourad Oussalah, Dalila Tamzalit
LINA, universit de Nantes
2 rue de la Houssiniere
44322 Nantes France
{mourad.Oussalah,dalila.Tamzalit}@univ-nantes.fr

Abstract

Software architecture modeling and representation became a main phase of the development process of complex systems. In fact, software architecture representation provides many advantages during all phases of software life cycle. Nevertheless, for many systems, like legacy or eroded ones, there is no available representation of their architectures. In order to benefit from this representation, we propose, in this paper, an approach called ROMANTIC which focuses on extracting a component-based architecture of an existing object-oriented system. The main idea of this approach is to propose a quasi-automatic process of architecture recovery based on semantic and structural characteristics of software architecture concepts.

1 Introduction

Software architecture modeling and representation are a main phase of the development process of complex systems [2]. Architecture representation is based on component and connector concepts in order to describe system structures at a high abstraction level. This representation provides many advantages during the software life cycle [9]. Indeed having a representation of software architecture makes exchanges between software architects and programmers easier. Then during maintenance and evolution phases, this representation helps to localize software defects and to reduce the risk of misplacing new system functionalities. Moreover the distinction which exists, in this representation, between components and connectors makes the separation between functional and communication aspects explicit and consequently makes the system comprehension and evolution easier. Finally, component-based architecture is also useful in order to facilitate the reuse of some system parts represented as components.

However most existing systems do not have a reliable architecture representation. Indeed these systems could have been designed without an architecture design phase, as it is the case for most legacy systems. In other systems, the available representation can diverge from the system implementation. This divergence, between the representation and the reality of the system is the result of the erosion phenomenon. This appears, first, during the implementation phase due to gaps between the expected architecture and the implemented one. These gaps become greater because of lack of synchronization between software documentation and implementation.

Taking into account the previous considerations, we propose an approach called ROMANTIC[1] which focuses on extracting a component-based architecture from object-oriented systems. Our extraction approach is different from existing ones by the fact that it uses several guides, like semantic of architecture elements or architectural quality properties, in order to drive the extraction process and decrease the need of human expertise which is expensive and not always available.

The remainder of this paper is structured as follows. Section 2 presents related work and the main advantage of our approach. In section 3 we introduces the principles of our extraction process. Section 4 exposes our measurement model of the semantic correctness of the architecture. A case study is presented in section 5. Conclusion and future works are given in section 6.

2 Related work

Various works are proposed in literature in order to extract architecture from an object-oriented system [8]. We distinguish these works according to two criteria: the process input and techniques used to extract architecture.

The inputs of the extraction approaches are various. Most often it works from source code representations, but it also considers other kinds of information. Most of them are

[1] ROMANTIC: Re-engineering of Object-oriented systeMs by Architecture extractioN and migraTIon to Component based ones.

0-7695-3092-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WICSA.2008.44

non-architectural. We can cite, for example, human expertise, which is used in an interactive way in order to guide the process [7, 6]. Some works use architectural input like styles for example. Medvidovic [7] uses styles in Focus in order to infer a conceptual architecture that will be mapped to a concrete architecture extracted from source code. Finally most works are based on the human expertise: some use the expertise of the architect which uses the tools as an input whereas others use the expertise of the one which proposed this approach. In our approach we propose to use architectural semantic in order to decrease this need of human expertise. This input will be completed by using other guides like the documentation which is used, in existing works, through the human expertise. Moreover ROMANTIC will use several inputs, using several guides whereas most works use only one or two inputs.

The techniques used to extract architecture are various and can be classified according to their automation level. First, some approaches are quasi manual. For example, Focus [7] proposes a guideline to a hybrid process which regroup classes and maps the extracted entities to an idealized architecture obtained from an architectural style according to the human expertise. Second most approaches propose semi-automatic techniques. It automates repetitive aspects of the extraction process but the reverse engineer steers the iterative refinement or abstraction, leading to the identification of architectural elements. For example, in Dali [6] reverse engineer specifies reusable abstraction rules and executes them automatically using SQL. Third some techniques are quasi-automatic. For example, the clustering algorithms identify groups of objects whose member are similar in some way and are used to identify subsystems [1]. ROMANTIC approach is quasi-automatic. The main difference with other quasi-automatic approaches is that we use several guides, like semantic of architecture elements or architectural quality properties, in order to drive the extraction process and decrease the need of human expertise whereas others works use the expertise of the authors in order to define rules driving the process.

3 ROMANTIC principles for software architecture extraction

During the design process of an architecture, an architect mainly uses requirement specifications and his skill knowledge in order to determine architectural elements : components which describe functional computing, connectors which describe interactions and configuration which represents the topology of connections between components.

Architecture extraction is the reverse process of the design one. Indeed the extraction process uses existing implementation code and the architect skills to obtain a system abstraction. The extraction process can be splited in two steps. On the one hand we must define a correspondence model between code elements and the architectural concepts. This correspondence model is our first principle. On the other hand we must instantiate the previous correspondence model and extract the architectural elements from the software. This step is driven by our second principle which defines a set of guides for our process.

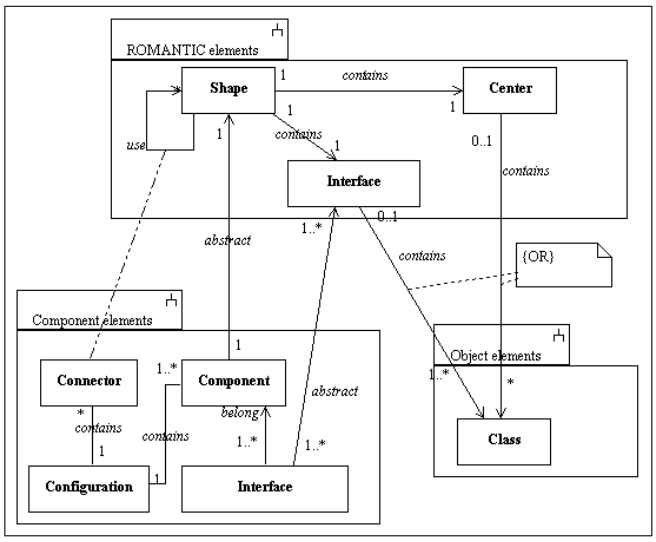

Figure 1. Our model of object-component correspondence

3.1 Principle 1: extraction of architectures from object-oriented systems

We focus in the ROMANTIC approach on the architecture extraction from an object-oriented system. Consequently, the first step of the extraction consists of defining a correspondence model between object concepts (*i.e.* classes, interfaces, packages, etc.) and architectural ones (*i.e.* components, connectors, interfaces, etc.)(*cf.* Fig.1).

Following this correspondence model, we define an architecture as a partition of the system classes. Each element of this partition represents a component. These elements are named "shape" and contain classes which can belong to different object-oriented packages. Each shape is composed of two sets of classes: the "shape interface" which is the set of classes which have a link with some classes from the outside of the shape, *e.g.* a method call to the outside; and the "center" which is the remainder of shape.

As shown in Fig.1, we assimilate component interface set to "shape interface" and component to shape. We focus in this paper on component extraction. For simplification reasons, we consider that connectors are all links existing between components. Consequently, the architecture con-

figuration is the set of shapes constituting a partition of the system classes.

3.2 Principle 2: guides of the architecture extraction process

The first ROMANTIC principle define a correspondence model between architectural elements and object-oriented ones. However we must instantiate this model in order to extract a relevant architecture. This second principle defines what a relevant architecture is and how to instantiate the previous model. Indeed, we consider that an architecture is relevant if it respects four guides (*cf.* Fig.2). Firstly, it must be semantically correct. Secondly, it must have good quality properties. Thirdly, it must respect precisely the recommendation of the architect and, as far as possible, specifications and constraints defined in documentations. Finally, it must be able to be adapted to the specificity of the deployment hardware architecture. We think that, using all these guides, our process provides a relevant component-based architecture representation.

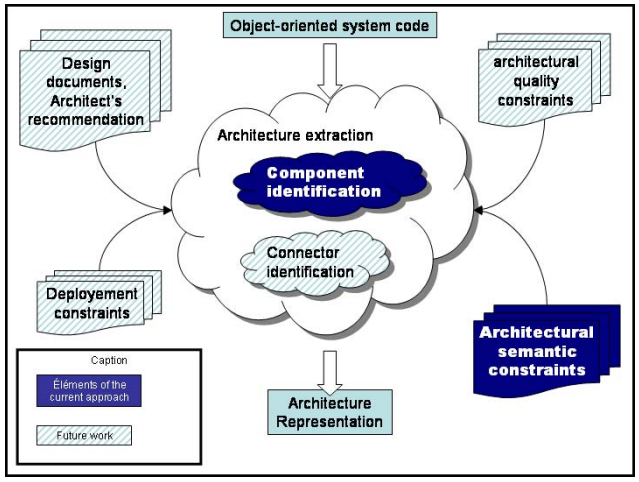

Figure 2. Extraction guides

ROMANTIC aims at using these guides in order to define an evaluation function of the relevance of an architecture. This function will be used to select the best architectures among all the possible ones. However, among all these guides, the one which uses the architectural element semantic is preponderant. That is why our first work was to use this guide. To do this we defined a function evaluating the software architecture semantic. Then we used a hierarchical clustering algorithm which uses this function and the model defined in the first principle to extract the best architecture.

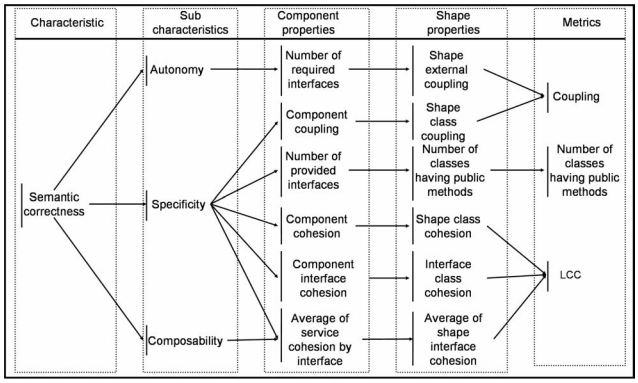

Figure 3. Our measurement model of the component semantic correctness

4 Evaluating the software architecture semantic

In order to evaluate the architecture semantic correctness, we use the refinement model given by the norm ISO-9126 [5]. According to this model we need to refine the semantic correctness characteristics in sub-characteristics. This refinement is done using the semantic which is associated to the architecture concept. However an architecture is semantically correct if its element are semantically correct. Consequently, focusing on components, we use the most commonly admitted definition of components in order to define the component semantic sub-characteristics.

Using the definition of Szyperski [10] and others [4], we identify three semantic sub-characteristics of software components: composability, autonomy and specificity which means that it must contain a limited number of functionalities.

Then according to the norm ISO-9126, we refine these sub-characteristics in component properties as the component cohesion for example. Then in order to define metrics for these properties we need to link them to the shape properties. These refinement and links are based on our analyze of these properties and are summarized in Fig.3.

The shape properties are measured by three metrics (*cf.* Fig.3). The "Loose Class Cohesion" (LCC), proposed by Bieman and Kang [3], measures the cohesion of a set of classes using the method calls and variable access. The "Coupling" measures the coupling between two set of classes using the number of dependences between these sets.

Finally we use our measurement model in order to define the functions Spe, A, C respectively evaluating specificity, autonomy and composability. We define the function evaluating the software architecture semantic as linear combination of each sub-characteristic evaluation function (Spe, A,

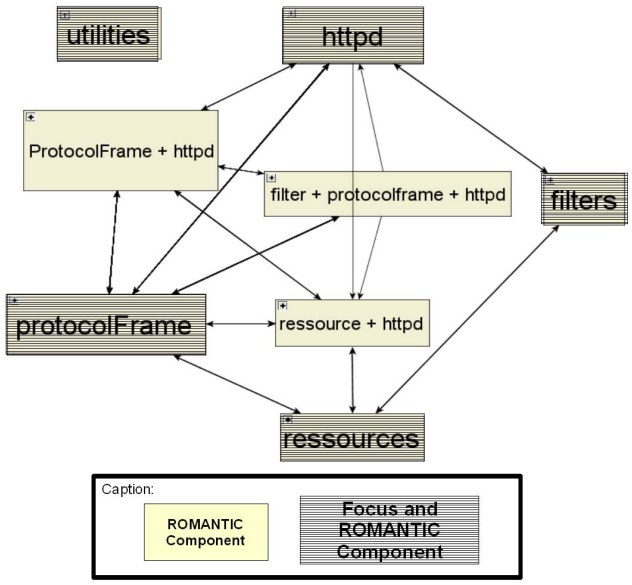

Figure 4. ROMANTIC architecture of Jigsaw

and C) where λ_i are chosen by the architect:

$$S(E) = \frac{1}{\sum_i \lambda_i}(\lambda_1 \cdot C(E) + \lambda_2 \cdot A(E) + \lambda_3 \cdot Spe(E)) \quad (1)$$

Then we can use this function as similarity function in a hierarchical clustering algorithm in order to obtain a partition of the system classes and consequently an architecture semantically correct.

5 Case study: Jigsaw

We have experimented our extraction process on the Jigsaw software. Jigsaw is a Java based web server. Its architecture is well-known and it has been used to present the architecture extraction process in existing works [7].

Jigsaw contains around 300 classes. We apply on the system classes a hierarchical clustering algorithm where the similarity is measured by our function evaluating the software architecture semantic. From this process, we obtain an architecture whose result for our evaluation function is 80% and the average size of the components is 14.

Fig.4 presents the superposition of Jigsaw architecture as thought and the architecture extracted by ROMANTIC. The comparison of these architectures shows that most of our components are sub-components of envisaged ones. Few of them are included, at the same time, in several envisaged components.

The study of these results shows that the ROMANTIC components included in expected ones define a functionality which is a sub-functionality of expected components. The ROMANTIC components which are not included in the expected ones seem to contain classes of several expected components These components seem to contains a part of the connector between the expected components.

6 Discussion and conclusion

We propose, in this paper, an approach for the extraction of a component-based architecture from an object-oriented system. This approach is based on the use of the component semantic characteristics. These characteristics guide the partitioning of the system classes and the abstraction of each shape in a component. The main difference with existing works on architecture extraction is that we refine the commonly used definitions of architecture into semantic characteristics and measurement models whereas other works use the expertise of the corresponding authors in order to define rules driving the process.

Our first future work is to study the algorithms which can be used in this approach and their comparison. The use of other guides is our second future work. We have started the study of the architecture quality properties and we also consider the use of the design documents and all additional information provided by the architect.

References

[1] N. Anquetil and T. C. Lethbridge. Recovering software architecture from the names of source files. *Journal of Software Maintenance*, 11(3):201–221, 1999.

[2] A. Bertolino, A. Bucchiarone, S. Gnesi, and H. Muccini. An architecture-centric approach for producing quality systems. In *QoSA/SOQUA*, pages 21–37, 2005.

[3] J. M. Bieman and B.-K. Kang. Cohesion and reuse in an object-oriented system. In *Proc. of the Symp. on Software reusability,SSR '95*, pages 259–262, 1995.

[4] G. Heinemann and W. Councill. *Component-based software engineering*. Addison-Wesley, 2001.

[5] ISO/IEC-9126-1. In *Software engineering - Product quality - Part 1: Quality Model*. ISO-IEC, 2001.

[6] R. Kazman, L. O'Brien, and C. Verhoef. Architecture reconstruction guidelines. Technical report, 2001.

[7] N. Medvidovic and V. Jakobac. Using software evolution to focus architectural recovery. *Automated Software Engineering*, 13:225–256, 2006.

[8] D. Pollet, S. Ducasse, L. Poyet, I. Alloui, S. Cimpan, and H. Verjus. Towards a process-oriented software architecture reconstruction taxonomy. In *Proc. of the 11th CSMR*, pages 137–148, 2007.

[9] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, USA, 1996.

[10] C. Szyperski. *Component Software*. ISBN: 0-201-17888-5. Addison-Wesley, 1998.