Multiple Viewpoints Architecture Extraction

Azadeh Razavizadeh
University of Savoie
LISTIC Lab
Annecy, France
azadeh.razavizadeh@univ-savoie.fr

Herve Verjus
University of Savoie
LISTIC Lab
Annecy, France
herve.verjus@univ-savoie.fr

Sorana Cimpan
University of Savoie
LISTIC Lab
Annecy, France
sorana.cimpan@univ-savoie.fr

Stephane Ducasse
INRIA Lille-Nord Europe
RmoD Team, LIFL
Lille, France
stephane.ducasse@inria.fr

Abstract—A software system’s architecture, its elements and the way they interact, constitute valuable assets for comprehending the system. Many approaches have been developed to help comprehending software systems in different manners. Most of them focus on structural aspects. We believe offering multiple views of the same system, using domain knowledge helps understanding a software system as whole.

To correlate domain information and existing software systems, different viewpoints are considered and modelled. Viewpoints guide the extraction of architectural views, the later representing different system facets. We propose a recursive framework, an approach that expresses domain knowledge as viewpoints to guide the extraction process. It provides multiple architectural views according to multiple given viewpoints.

I. INTRODUCTION

Software systems need to evolve over time [12]. They are modified to improve their performance or change their functionality in response to new requirements, detected bugs, etc. Changes are thus part of the system maintenance; they preserve the system functionality and ensure it performs well. Other changes evolve the system, generally by adding new functionalities, modifying its architecture, etc. Thus, there are several evolution phases for which different processes may be employed. The continuous evolution increases the system complexity [12].

The evolution process ideally begins with the system comprehension and continues with finding a suitable set of system modifications. It has been measured that in the maintenance and evolution phases, at least half of the engineers’ time is spent to the system comprehension [3]. To successfully evolve a complex system, it is essential to understand it. The understanding phase is time and effort consuming, due to several reasons, among which: the system size (large systems consist of millions lines of code), inappropriate documentation, lack of overall views of the system, its previous evolutions (not necessarily documented), etc. This motivates our work on supporting the understanding phase of software system evolution by extracting higher level system views.

Architectures serve thus as education means [1] and guide the software evolution, by providing high-level abstract models of existing software systems. Software architecture reconstruction is a reverse engineering approach that aims at reconstructing viable architectural views of a software application [3]. Some researchers have highlighted the importance of considering different knowledge sources when extracting a system architecture. Reflection models propose a recovery process by considering developer knowledge [16][8]. The most reliable source of information is the system’s source code, therefore we consider it as the main source of information. We limit our proposition to object oriented systems. The approach proposed can nevertheless be extended to other information sources.

We focus on extracting architectural views of existing software systems. It is widely accepted that multiple architectural views are needed to describe the software architecture [9][7][1]. Architecture relevant information can be found at different granularity levels of given systems and needs to be studied from different viewpoints. A model providing the main concepts and their relationships defines a viewpoint. The extraction rules indicate how the architectural view elements map to the elements of the architectural viewpoint. Different viewpoints are considered such as business-based, software pattern-based, cohesion-based model, etc.

The main contributions of the proposal presented in this paper are:

- the construction of multiple architectural views of existing software systems according to several viewpoints;
- the improvement of software system understanding by presenting high level complementary views of the system, especially in a software evolution perspective;
- a model-based approach that is generic enough to be viewpoint independent.

Structure of the paper: In the next section, we present a recursive framework and its related concepts: architectural view, architectural viewpoint and the extraction process that employs and extracts architectural views using viewpoints. In section 3 we describe how architectural views (and corresponding viewpoints) can be cascaded using the recursive framework. Then, section 4 browse the concluding remarks.

II. A RECURSIVE FRAMEWORK

We adopt a generic and recursive framework (Figure 1) for extracting architectural views: at each recursion of the framework, a specific view is extracted (also results) from another view, under the influence of a guiding viewpoint. Our framework is reproducible by chaining horizontal and/or vertical recursions of the framework (Figure 4).

- horizontal recursion: starting from a given architectural
view, other architectural views can be elaborated, each based on a specific viewpoint:

- **vertical recursion**: architectural views are organized in a pipe-line style, such that the output (view) of an extraction recursion is used as an input (view) for another one (Figure 3).

Each horizontal or vertical recursion corresponds to an instance of the generic framework and supports a specific viewpoint.

![Figure 1. The recursive framework](image)

**A. Architectural Views**

We propose a simple **Architecture Meta Model** for representing architectural views: a system architectural view is represented as a set of interconnected **architectural elements**. An architectural element of an extracted architectural view is characterized as a group of another architectural elements of the architectural view obtained at the previous recursion of the framework (and given as input).

![Figure 2. Architectural Meta Model](image)

The **Architecture Meta Model** deals also with relationships among architectural elements. Relationships among architectural elements of an extracted architectural view are deduced from relationships among architectural elements of the architectural view given as input of the framework. Architectural views are thus generated using an extraction process, which supports both the identification of the architectural elements and their relationships.

We call **implementation view** the result of the first instantiation of the framework: it consists in generating an (architectural) implementation view of the source code (i.e., flat files containing the source code expressed in a object-oriented programming language). This implementation view is composed of architectural elements such as classes, methods, and packages. We note here that this implementation view is the result issued from a re-engineering approach like [17][15] etc.

**B. Architectural Viewpoints**

Architectural views and viewpoints have been considered by several research approaches [9][18][2]. Starting from an implementation view of the software system, we provide several architectural views according to different viewpoints. We consider the following viewpoints, but the framework can easily integrate other viewpoints:

- a **business domain-based viewpoint** which considers the principal business domain concepts and their relationships;
- a **pattern-based viewpoint** [6][19] which identifies architectural elements conform to a given pattern;
- a **cohesion-based viewpoint** [14][11] which identifies a set of related architectural elements with strong dependencies.

For each viewpoint, a specific **Viewpoint Model** (VptM) is defined. Each viewpoint reveals certain aspects of the software system. For example, the extracted architectural view of the business domain-based viewpoint presents the system architectural elements organization in accordance with the business domain concepts. Such a view mainly provides an overall view of the system in terms of the business concepts and helps different stakeholders in their system understanding. Let us consider a Banking software application example. The business domain concepts of such an application can be **Bank**, **Client**, **Account**, **Credit card**. The corresponding VptM of the business domain-based viewpoint comprises these concepts as entities and their relationships (i.e., a Bank may have more than one Client; a Client may have more than one Account; a Credit card concerns a specific Account, etc.). This VptM supports and guides the architectural view extraction process.

**C. Extraction Process**

The extraction process enacts the recursive framework presented before. An extraction process corresponds to the framework application: an architectural view that is extracted from a view given as input of the extraction process is considered as the view generated by the framework. Several views may be extracted from a given view, using different VptMs. In that sense, we propose a viewpoint-driven extraction process. Practically, the recursive framework entails at least two extraction recursions: the first one re-engines (using the Moose [4] re-engineering environment) the software system from its source code and produces a Famix model [5] as an implementation view; the second recursion extracts an architectural view from this implementation view. Other recursions
can generate additional architectural views (in a cascading way, cf. Figure 3).

![Diagram](image_url)

Figure 3. Reverse-engineering and Retrieving-views

For each specific viewpoint a set of extraction rules are defined supporting the extraction (and generation) of architectural views (as outputs of the framework). These rules consider two criteria: (i) the kind of viewpoint, (ii) the input architectural elements. An extraction algorithm entails these rules and indicates how architectural elements of the framework input architectural view are grouped according to the Viewpoint Model (VptM) concepts and their relationships.

Algorithm 1 Domain-Based Extraction Algorithm

```plaintext
Require: ImpV: Implementation View elements
VptM: Viewpoint Model elements
FOR i=1 to i = size of VptM
   FOR j=1 to j = size of ImpV
      IF (VptM[i]) match (ImpV[j] name) THEN
         CALL FIND-GROUP[i]
         ADD ImpV[j] in GROUP[i]
         INCREMENT j
      ELSE IF (j == size of ImpV)
         ADD ImpV[j] in GROUP[outside]
      END IF
   END FOR
END FOR
```

In the Banking software application example, the extraction process uses the above algorithm. The latter searches all classes (considered as architectural elements) of the implementation view which contain the concept of VptM in their name (i.e., Bank, Account, Credit card, etc.); when found, the classes are put in a group (i.e., architectural element) labeled with the corresponding concept. The classes that do not correspond to any concept of the VptM are grouped in another group labeled Outside domain. The process can also be considered at a finer grain level if needed: the extraction rules can target different architectural elements (i.e., packages, classes, methods) of the implementation view. Thus, different architectural views can be generated accordingly. Moreover, at the implementation view level, the extraction process can identify subsets of classes according to the VptM concepts. A subset of a class is a group of related attributes and methods and can be considered as traits [13]. As a consequence, an architectural element (i.e., a class) can be seen as a set of (potentially and partially) overlapping traits. The same trait can be found in different architectural elements.

The same principles presented before are used in employing software pattern-based viewpoints. The viewpoint model is then a Software Design Pattern Model. For instance the MVC pattern model contains Model, View, Controller as concepts of the MVC’s domain with their relationships. In this case, the extraction process groups architectural elements according to MVC’s concepts and their relationships.

### III. Cascading Extraction Processes

Our recursive approach enables one to use a previously extracted architectural view as an input for a new extraction process with a new (or even the same) architectural viewpoint. Therefore, each extracted architectural element (that is a group of elements of the architectural view given as input) may be also considered as an input of the recursive framework. Taking again the principle of our extraction algorithm, applying the framework recursively consists in defining and organizing architectural elements of the generated view according to the viewpoint model and the extraction rules application. As consequence, each architectural element of this generated view may be refined according to a new viewpoint model, and so on.

Considering again the Banking software application example, we are cascading extraction processes (i.e., several framework recursions): starting from the source code of the software application, the first extraction process produces the implementation view which serves as the architectural input view for another extraction process; this latter considers a business domain-based viewpoint and generates a business domain-based architectural view. This business domain-based architectural view is itself employed as input of a third extraction process using a MVC pattern viewpoint. This third extraction process identifies MVC viewpoint architectural elements (i.e., Model, View, Controller) of each architectural element of the business domain-based architectural view (given as input) and generates new architectural views accordingly.

As a result of cascading extraction processes, we obtain an abstract architectural view for which architectural elements reveal business domain concepts (second viewpoint used) and are composed of those of Model, View and Controller elements (third viewpoint used).

### IV. Concluding Remarks

The approach presented in this paper can be used at any object-oriented source code granularity level. The straightforward approach uses classes as the first architectural element kind, but any other slicing like method-based and/or package-based can be used.
This paper proposes a generic and recursive approach for considering some knowledge (as viewpoint) into reconstruction process of existing system. We use the extraction process to reveal the architectural views of system according to defined viewpoints.

ACKNOWLEDGEMENTS

This work has been partially funded by the French ANR JC05 42872 COOK Project.

REFERENCES