Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Combining lexical and structural information to reconstruct software layers

Alvine Boaye Belle*, Ghizlane El Boussaidi, Sègla Kpodjedo

Department of Software and IT engineering, École de technologie supérieure, Montreal, Canada

ARTICLE INFO

Article history:
Received 26 July 2015
Revised 22 December 2015
Accepted 21 January 2016
Available online 15 February 2016

Keywords:
Software maintenance
Reverse engineering
Architecture recovery
Layering style
Hill climbing
Latent Dirichlet Allocation

ABSTRACT

Context: The architectures of existing software systems generally lack documentation or have often drifted from their initial design due to repetitive maintenance operations. To evolve such systems, it is mandatory to reconstruct and document their architectures. Many approaches were proposed to support the architecture recovery process but few of these consider the architectural style of the system under analysis. Moreover, most of existing approaches rely on structural dependencies between entities of the system and do not exploit the semantic information hidden in the source code of these entities.
Objective: We propose an approach that exploits both linguistic and structural information to recover the software architecture of Object Oriented (OO) systems. The focus of this paper is the recovery of architectures that comply with the layered style, which is widely used in software systems.
Method: In this work, we (i) recover the responsibilities of the system under study and (ii) assign these responsibilities to different abstraction layers. To do so, we use the linguistic information extracted from the source code to recover clusters corresponding to the responsibilities of the system. Then we assign these clusters to layers using the system's structural information and the layered style constraints. We formulate the recovery of the responsibilities and their assignment to layers as optimization problems that we solve using search-based algorithms.
Results: To assess the effectiveness of our approach we conducted experiments on four open source systems. The so-obtained layering results yielded higher precision and recall than those generated using a structural-based layering approach.
Conclusion: Our hybrid lexical–structural approach is effective and shows potential for significant improvement over techniques based only on structural information.

1. Introduction

Recovering architectures of existing software systems remains a challenge, especially in the context of large and complex systems. Architecture recovery may be achieved using a bottom-up process that starts from source code and progressively constructs a more abstract representation of the system [1]. Such representation helps supporting the designer in understanding and properly evolving an existing software system. Thus, many approaches were proposed to support the architecture recovery process. Most of these approaches (e.g., [2,4]) rely on clustering techniques to find a division of the system that optimizes the modularity of resulting clusters in terms of high-cohesion and low-coupling. However, these approaches do not consider the architectural style of the system under analysis.

In this paper, we focus on the recovery of architectures that comply with the layered style, which is widely used in software systems. The layered style is a technique for structuring software as an organized hierarchy of layers where each layer provides services to the layer above it and serves as a client to the layer below [5,6]. Each layer comprises a set of modules which are cohesive with respect to their responsibilities [6]. In a "strict" [5] (or "closed" [7]) layering, the layers should interact according to a strict ordering relation, i.e. a layer may only use services of the next lower layer.

The layered style promotes many quality attributes such as reuse, portability and maintainability but also comes with some liabilities such as a lack of flexibility and weak performance [5]. To address these liabilities, current practice involves some deviations from a strict layering: (i) skip-calls [8] (also called "layer bridging" [6]) and (ii) back calls [8] (also called "upward usage" [6]). Skip calls refer to situations in which a layer uses services of a layer

* Corresponding author. Tel.: +15145858531.
E-mail addresses: ak20180@ens.etsmtl.ca, nebliva@gmail.com (A.B. Belle), Ghizlane.ElBoussaidi@etsmtl.ca (G.E. Boussaidi), Segla.Kpodjedo@etsmtl.ca (S. Kpodjedo).

http://dx.doi.org/10.1016/j.infsof.2016.01.008
0950-5849/© 2016 Elsevier B.V. All rights reserved.

that is not immediately below it [6]. Back calls refer to (normally exceptional) situations in which a layer needs to rely on a service offered by an upper layer.

Thus the structure of an architecture which complies strictly with the layered style is a directed acyclic graph. This property led to several layering recovery approaches that are based on a depth traversal of dependency graphs built from the analyzed system (e.g., [3,8–11]), often coupled with some heuristics to deal with the violations of this property. For instance, [8,9,11] rely on heuristics to resolve cyclic dependencies while [9,10] propose heuristics based on the number of fan-out and fan-in dependencies of an entity (i.e., class or package) to assign it to the lowest or highest-level layer. These heuristics may result in architectures with very few layers (e.g., in case of a heuristic based on highly connected modules [8,11]) or too many layers (e.g., in case of heuristics to resolve cyclic dependencies [9]). Besides, most of these approaches rely on structural dependencies between entities of the system and do not exploit the semantic information hidden in the source code of these entities.

In this paper, we propose an approach that combines lexical and structural information of a given system to recover its layered architecture. We use the lexical information to recover the responsibilities of the system. To do so, we identify meaningful topics in entities of the system using the Latent Dirichlet Allocation (LDA) statistical model [12]. Assuming that the topics of an entity convey some evidence about its responsibilities, we use these topics to group similar entities into clusters corresponding to responsibilities of the system. We then assign the resulting clusters to layers using the structural information and exploiting layering properties and constraints that we defined in previous work [13]. Both the processes of recovering responsibilities of the system and assigning them to layers are formalized as optimization problems that are solved with a search-based algorithm. The effectiveness of our approach is evaluated through an experiment on four open source systems and a comparative study between our approach and two other approaches that use structural information only. The results of these experiments are promising and they show the potential of the proposed approach for significant improvements over purely structural-based approaches.

The paper is organized as follows. Section 2 is dedicated to some necessary background on LDA and to related work. An overview of the proposed approach is given in Section 3. Section 4 presents the process of recovering responsibilities of the system while section 5 presents the process of assigning them to layers. Section 6 describes the experimental setting. Section 7 reports and discusses the experimental results. Section 8 concludes the paper and outlines some future directions for our work.

2. Background and related works

2.1. Latent Dirichlet Allocation

Extracting lexical information such as identifier names and comments allows enriching software analysis [14]. To extract the lexical information, a current practice is to resort to topic modeling. A topic model is a statistical method that analyzes the words contained in a corpus of documents in order to extract the themes that run through these documents, the links between these themes and the evolution of these themes over time [12]. One of the most popular topic modeling technique is LDA (Latent Dirichlet Allocation), which is a probabilistic model used in natural language processing to extract a set of latent topics from a corpus of text documents [15]. LDA models each document as a probability distribution over topics and each topic as a probability distribution over the words in the vocabulary [16]. The estimation of these distributions allows generating the set of T topics used in the corpus of documents as well as the distribution of these topics in each document [15]. Relying on topic distributions instead of bags of words to represent documents not only decreases the effect of lexical variability but also preserves the semantic structure of the corpus of documents [18].

LDA takes as input a word-by-document matrix M where m_{ij} represents the importance of the word w_i in the document d_j. It also requires a set of hyper-parameters that affect the resulting distributions of topics per document and words per topic. Simply explained, these hyper-parameters are ([15] and [17]):

(1) α which impacts the topic distribution per document: the higher α, the higher the likeliness for every document to be composed of every topic in significant proportions.
(2) β which impacts the word distribution per topic: the higher β, the bigger the set of words per topic.
(3) T which is the number of topics to be identified from the corpus. A small value of T results in too broad topics i.e. topics which are constituted by keywords coming from multiple concepts and that are hard to discriminate [19,16]. On the other hand, with a too high value of T, the same concept can be spread over numerous topics and these ones are then diluted and meaningless [16]. As these topics are made of idiosyncratic words, they can become uninterpretable [19].

To estimate topic per document and word per topic distributions, many approximate algorithms can be used. In this paper, we use the Gibbs sampling method [20] with the speed-up enhancements introduced by Yao et al. [18]. The Gibbs sampling method uses a Markov Chain Monte Carlo method to converge to its target distributions after N iterations, each iteration consisting in sampling a topic for each word.

2.2. Related works

Several approaches have been proposed to recover software architectures (e.g., [2–4, 8–11,13,21–26]). These approaches use various techniques and exploit various information of the analyzed system [1,27]. Most of these approaches are tailored for specific languages and platforms and do not rely on a standard data representation of the analyzed system [3]. Besides, the lack of ground-truth architectures or of experts able to produce these architectures impedes the comparison or the assessment of the techniques used by these approaches [27,4]. In the following we classify existing approaches into two categories: (i) approaches that target the architecture recovery in general and (ii) approaches that target the recovery of layered architectures.

Approaches that target architecture recovery in general: A popular technique used by these approaches is clustering (e.g., [2,4,28,29]. The latter is generally performed using properties, such as high-cohesion and low-coupling, which are mostly derived from structural information. In particular, Mancoridis et al. [28] presented a modularization tool called Bunch which uses a family of search-based algorithms such as hill climbing (HC) and genetic algorithms (GA) to produce a high level view of an analyzed system. Bunch uses a fitness function called MQ (Modularization Quality) which aims at minimizing the coupling between resulting clusters and maximizing their cohesion. In follow-up papers [2,55], the authors extended [28] by notably proposing a new formulation of MQ. Praditwong et al. [29] formulated software module clustering as a multi-objective problem that they solved using an evolutionary algorithm.

Important knowledge is embedded in the identifiers and comments found in the source code [14,31]. Some approaches (e.g., [14,32,33]) propose a lexical-based clustering approach that ignores structural dependencies when partitioning a system into subsystems. In particular, Anquetil and Lethbridge [34] used file names