Science of Computer Programming 121 (2016) 176–189

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Knowledge representation and information extraction for analysing architectural patterns

Perla Velasco-Elizondo [a,*], Rosario Marín-Piña [b], Sodel Vazquez-Reyes [a], Arturo Mora-Soto [b], Jezreel Mejia [b]

[a] *Autonomous University of Zacatecas, Zacatecas, ZAC., 98160, Mexico*
[b] *Centre for Mathematical Research, Zacatecas, ZAC., 98060, Mexico*

A R T I C L E   I N F O

Article history:
Received 15 April 2015
Received in revised form 16 December 2015
Accepted 30 December 2015
Available online 21 January 2016

Keywords:
Architectural design
Architectural patterns
Quality attribute
Ontology
Information extraction

A B S T R A C T

Today, many software architecture design methods consider the use of architectural patterns as a fundamental design concept. When making an effective pattern selection, software architects must consider, among other aspects, its impact on promoting or inhibiting quality attributes. However, for inexperienced architects, this task often requires significant time and effort. Some reasons of the former include: the number of existing patterns, the emergence of new patterns, the heterogeneity in the natural language descriptions used to define them and the lack of tools for automatic pattern analysis. In this paper we describe an approach, based on knowledge representation and information extraction, for analysing architectural pattern descriptions with respect to specific quality attributes. The approach is automated by computable model that works as a prototype tool. We focus on the performance quality attribute and, by performing experiments on a corpus of patterns with forty-five architects of varying levels of experience, demonstrate that the proposed approach increases recall and reduces analysis time compared to manual analysis.

1. Introduction

When the architecture of a software system is designed, one key task is the selection of *design concepts* in order to satisfy a set of *architectural drivers* [1]. Architectural drivers are requirements that shape a software system and consist of *high-level functional requirements*, *constraints*, and *quality attribute* requirements [2]. Many software architecture design methods consider patterns as a fundamental design concept, e.g., Rozanski and Woods' [3], ADD [1], Microsoft's Technique for Architecture and Design [4]. The ones of our interest are *architectural patterns*, which denote a reusable named solution applicable to a commonly occurring problem in software architecture design. There are a number of architectural pattern catalogues that software architects have been using for years, e.g., Pattern-Oriented Software Architecture [5] and Patterns of Enterprise Application Architecture [6].

When making an effective pattern selection, software architects must consider, among other aspects, its impact on promoting or inhibiting quality attributes. However, for inexperienced architects, this task requires significant effort and can be time consuming for reasons including:

* Corresponding author.
E-mail address: pvelasco@uaz.edu.mx (P. Velasco-Elizondo).

http://dx.doi.org/10.1016/j.scico.2015.12.007
0167-6423/© 2016 Elsevier B.V. All rights reserved.

(i) *The number of existing patterns*. Nowadays there are plenty of architectural patterns' catalogues, e.g., Pattern-Oriented Software Architecture [5], Patterns of Enterprise Application Architecture [6], Service Oriented Architecture (SOA) Design Patterns [7], Service Design Patterns [8], Big Data Application Architecture [9]. Most of these catalogues describe more than fifty patterns; each pattern description is about two pages. Some reading time estimations state that reading and understanding one page of text takes from two to six minutes depending on the reader's experience on the subject [10]. Based on an average of four minutes per page, if a two-page pattern takes a total of eight minutes to read and understand, one hundred pattern descriptions would require approximately thirteen hours.

(ii) *The emergence of new patterns*. Since architectural patterns are fundamental design concepts, every time a new software development paradigm appears, new patterns related to it also arise. For example, the popularization of SOA promoted the definition of architectural patterns as the ones in SOA Design Patterns [7] and Service Design Patterns [8] catalogues. Similarly, cloud and big data software systems have contributed to the emergence of architectural patterns to tackle specific architectural drivers in these contexts, e.g., MapReduce Design Patterns [11], Big Data Application Architecture [9], Cloud Design Patterns [12]. Thus, the number of pattern descriptions an architect must read and consider at any given time is always increasing, adding to the time and effort required to evaluate them.

(iii) *The heterogeneity of pattern descriptions*. Although most patterns are defined in terms of a common set of elements, e.g., name, intent, context, participants, the description of these elements is written in natural language without standardization. For example, in describing quality attributes, a variety of concepts could be used to describe whether a pattern promotes or inhibits performance quality, including 'concurrency', 'overhead', 'speed', 'latency' or 'capacity'. This lack of standardized terminology could have an impact on how the elements are understood and evaluated by inexperienced architects [13].

(iv) *The lack of tools for automatic pattern analysis*. There are some tools that an architect could use to automatically identify the most suitable patterns for a software architecture design. However, the lack of a standard mechanism for indexing pattern catalogues as well as more efficient search engines makes these tools limited [13]. As it will be explained in Section 2, in most of the tools pattern selection is made from a pre-defined pattern repository. Most of the time, this repository is static and cannot be extended to include new patterns. When it is possible, the analysis and classification of new patterns are performed manually, becoming a time consuming task.

In this work we describe an approach, based on knowledge representation and information extraction, to automate the analysis of architectural pattern descriptions and help inexperienced software architects with determining whether specific quality attributes are promoted or inhibited. Knowledge representation methods provide a basis on which to design and implement mechanisms for representing information in computers so that programs can use this information to solve problems in areas that normally require human expertise [14]. On the other hand information extraction allows extraction from text documents salient facts about pre-specified types, entities or relationships [15,16]. The approach is automated by computable model that works as a prototype tool. In this paper we focus on the performance quality attribute and, by performing experiments on a corpus of patterns with forty-five architects of varying levels of experience, demonstrate that the proposed approach increases recall and reduces analysis time compared to manual analysis.

This paper is organised as follows: in Section 2 we describe relevant related work; in Section 3, we describe the proposed approach to analysing architectural pattern descriptions and in Section 4 we discuss and evaluate this approach. Finally, in Section 5, we state the conclusions and describe some lines of future work.

2. Related work

There have been several attempts to provide tools and frameworks to assist architects during architectural design. In this section we relate our work to other literature in this context.

DesignBots [17] is a planning-based design framework that assigns architectural knowledge to agents that compete in different quality attributes. The framework requires the architect to provide an initial architecture supporting functional requirements and a weighted set of related quality attributes scenarios [18]. Thus, using a pre-defined set of design concepts as well as information provided by the architect, a set of design alternatives to improve the initial architecture are automatically generated from the cooperative work of the agents.

Jabali et al. [19] propose a method, and the corresponding tool support, for choosing a suitable software architecture design that satisfies multiple required quality attributes [19]. The method requires a set of weighted quality attribute requirements and a set of high-level functional requirements. Using a pre-defined set of design concepts, the method applies data-driven decision making to generate a proposal for the required design.

Hadaytullah et al. [20] present a tool for producing potential architecture proposals using genetic algorithms. The tool requires a basic functional decomposition of the system, a set of high-level functional requirements and the specification of the quality requirements. As in previous approaches, the method uses a pre-defined set of design concepts for producing potential architecture proposals.

Charmy [21] is a framework whose goal is to apply model-checking techniques to discover potential inconsistencies of an architectural design and allow architects to fix them by applying suggested design decisions. The specification of an architectural design is given in terms of components, connectors, their internal functional behaviour and relations. When an adequate design is reached, Java code conforming it can be automatically generated through suitable transformations.