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

ARTICLE INFO

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

ABSTRACT

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.

© 2016 Elsevier B.V. All rights reserved.

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.

Table 1
Main characteristics of related works.

	Used technique	Focus on pattern selection	Required Information			
			Pre-defined design concepts or experiences	Initial architecture design	Functional requirements	Quality attribute requirements
DesignBots [17]	multi-agent planning	no	yes	yes	yes	yes
Jabali et al. [19]	data-driven decision-making	no	yes	no	yes	yes
Hadaytullah et al. [20]	genetic algorithms	no	yes	yes	yes	yes
Charmy [21]	model-checking	no	yes	yes	yes	no
ArchE [22]	rule-based reasoning	no	yes	yes	yes	yes
Archpad [23]	decision models	yes	yes	no	yes	yes
Weiss and Birukou [24]	wiki	yes	yes	no	yes	yes
SYSAS [25]	knowledge-driven decision-making	yes	yes	yes	yes	yes
Kampffmeyer and Zschaler [26]	ontology	yes	yes	no	yes	yes
Hsueh et al. [27]	ontology	yes	yes	no	yes	yes
Kim and Khawand [28]	ontology	yes	yes	no	yes	yes
Hasheminejad and Jalili [29]	ontology	yes	yes	no	yes	yes
Pearson and Shen [30]	rule-based reasoning	yes	yes	no	yes	yes
Weiss and Mouratidis [31]	goal-based reasoning	yes	yes	no	yes	yes
Our approach	information extraction	yes	no	no	no	yes

ArchE [22] is a prototype tool that, using knowledge of quality attributes theories, helps architects by suggesting possible design decisions as well as predicting quality attribute responses of the resulting architecture in specific situations. Based on a pre-defined set of design concepts, ArchE uses a rule-based engine and reasoning framework to analyse an architecture for performance and modifiability. ArchE requires three types of inputs: a set of quality attribute requirements, a set of high-level functional requirements, and an architectural design (if available).

ArchPad [23] is a design method, proposed by Zimmermann et al., based on the use of domain-specific architectural patterns and decision models. A decision model stores architectural decisions, to promote functional and quality attribute requirements, harvested from experiences in previous projects. ArchPad identifies the required decisions in requirements models, gives domain-specific pattern selection advice, and provides traceability from platform-independent patterns to platform-specific decisions. In ArchPad models are created manually by experts and contain patterns proved to be applicable in the past projects in the domain.

In [24] Weiss and Birukou present a wiki-based pattern repository. The repository organizes short descriptions of patterns in collections and allows for their navigation, linking, tagging, commenting and searching. The repository supports any kind of pattern as long as a it can be described attending a proposed template. These templates are filled manually by design experts.

SYSAS [25] is a method to select architectural patterns for defining the overall structure of a software system. SYSAS is based on a mathematical decision-making technique that operates on a set of a pre-defined and closed pattern repository. The method requires ratings about the importance of functional and quality attribute requirements related to the architectural elements that are relevant for the developer and a description of the target system. The expected output is the total score, including satisfaction value, of each candidate pattern.

In [26] Kampffmeyer and Zschaler introduce an approach that automates pattern selection. The approach uses an ontology-based specification of the intent of a set of patterns from which a user executes queries focusing on the design problem she or he is trying to solve. Some other works in this vein are [27–29], proposed by Hsueh et al., Kim and Khawand, and Hasheminejad and Jalili respectively. A common aspect in all these approaches is that they characterise existing patterns considering patterns' descriptive elements such as intent or structure. Note, however, that the descriptions of these elements do not always focus on discussing the impact of using the pattern on a system's quality attributes. Therefore, pattern selection is not always quality attribute oriented.

In [30] Pearson and Shen present a decision support system that recommends design patterns satisfying privacy and security requirements. The system uses a rule-based engine to trigger decisions about appropriate patterns to use for the specified requirements. Rules and patterns must be created manually by design experts based on their experiences in a domain. Another similar work is the one presented in [31], by Weiss and Mouratidis, which is based on a goal-oriented reasoner supporting the selection of the security patterns. A user submits security and other non-functional requirements as a query to the reasoner, which returns the list of patterns fulfilling the specified requirements.

Table 1 shows a summary of the main characteristics of the above related works. These include *used technique* – e.g. multi-agent planning, *focus on pattern selection*, and *required information*. Required information includes whether a related work considers the following factors necessary: a set of *pre-defined design concepts or experiences* – e.g. a repository of pattern descriptions or experiences in previous projects, an *initial architecture design* of the prospective system, as well as specifications of *functional* and quality attribute requirements.

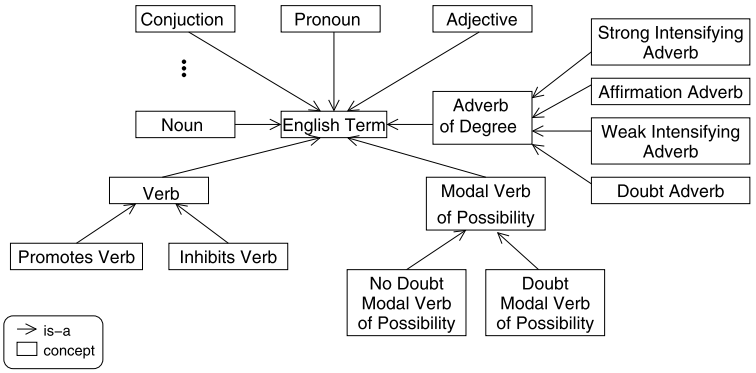

Fig. 1. An excerpt of the English ontology.

As can be noted, the approach presented in this paper (listed at the bottom of Table 1) relates to all of the above works and shares some characteristics with them. However, this work is different from the aforementioned in that (i) it focuses on quality attribute-oriented pattern selection; (ii) it does not require a pre-defined set of design concepts; (iii) it does not require specification of the design problem and (iv) it uses information extraction as its main technique. To the best of our knowledge, there are currently no proposals using information extraction to support the selection of patterns, or other design concepts, during architectural design.

3. The proposed approach

Knowledge representation methods provide a basis for designing and implementing mechanisms for representing information in computers so that programs can use this information to solve problems in areas that normally require human expertise [14]. Knowledge representation methods apply theories and techniques from three main fields: (i) *logic*, to provide a formal structure for information; (ii) *ontology*, to define the concepts that exist in a specific application domain and (iii) *computation*, to allow the logic and ontology to be implemented in computable models to solve problems in some domain.

In this section we will describe a computable model for analysing architectural pattern descriptions to help software architects determine whether specific quality attributes are promoted or inhibited. For simplicity, but without significant loss of generality, the focus is given on the performance quality attribute. Performance is related to the software system's ability to meet timing requirements.

3.1. Defining the domain concepts

To support the analysis of architectural pattern descriptions regarding the performance quality attribute we defined an ontology. This ontology consists of a body of concepts that are expected to be discovered in textual pattern descriptions. The ontology was modelled in OWL-DL using the OWL extension of Protégé [32]. The defined ontology consists of two sub-ontologies: the *English* and *Performance*. The identification of the concepts in the ontology was performed manually by a domain expert, in this case a senior software architect with more than ten years' experience.

The English ontology defines generic English grammar-based concepts often used in architectural pattern descriptions. Fig. 1 shows an excerpt of this ontology. As can be seen, this ontology includes concepts that are the common English parts of speech. However, we have further categorised verbs, modal verbs and adverbs considering part of the rationale in some opinion mining approaches, e.g., [33,34].

Although several classifications exist for English verbs, adverbs and adjectives, e.g., VerbNet [35], Jassem's adjective classification [36], Levin's verb taxonomy [37] and WordNet [38], they are all restricted to certain general domain classes and often come with few class instance exemplars. Thus, these classifications are not effective for architectural pattern analysis. This motivated the definition of the sub-classes shown in Fig. 1. Next, these sub-classes are briefly described.

Regarding verbs, a *promotes verb* positively reinforces a performance statement expressed in a sentence, e.g., "increase"; while an *inhibits verb* negatively reinforces it, e.g., "decrease". As modal verbs of possibility are used to denote degrees of certainty, a *no doubt modal verb of possibility* expresses certainty, e.g., "can". A *doubt modal verb of possibility* expresses probability or possibility, e.g., "might". Adverbs of degree tell about the intensity with which something happens. Thus, we consider *affirmation adverbs*, e.g., "totally"; *doubt adverbs*, e.g., "roughly"; *strong intensifying adverbs*, e.g., "extremely"; and *weak intensifying adverbs*, e.g., "slightly".

To populate the English ontology, instances of candidate concepts were extracted from a pattern description corpus, which we created from classical books of architectural patterns, e.g. Pattern-Oriented Software Architecture [5], Patterns of Enterprise Application Architecture [6]. The selected instances were then ranked according to the frequency of individual

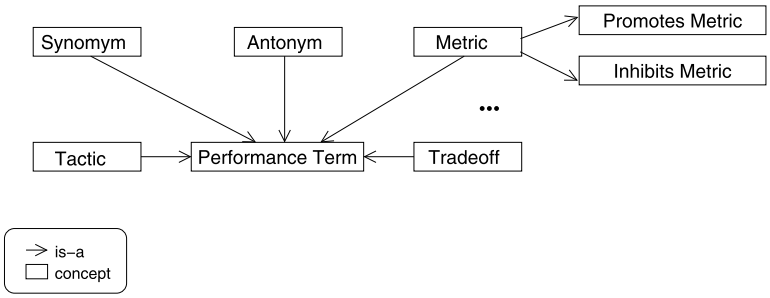

Fig. 2. An excerpt of the Performance ontology.

occurrence and co-occurrence with other concepts' instances. A stop list of general verbs, modal verbs, adverbs, and adjectives frequently mentioned in a corpus of software engineering articles was used to filter out the extracted instances. The top ranked instances were selected and considered as domain-specific.[1]

On the other hand, the Performance ontology defines performance-specific concepts. Many of these concepts and their instances come from (*i*) software quality models, e.g., Dromey's quality model [39], Mc Call's quality model [40], ISO/IEC [41]; (*ii*) the performance tactics in [42,43], e.g. "concurrency"; (*iii*) performance metrics, e.g., "throughput" and (*iv*) well-known performance trade-offs, e.g., "security" (as it inhibits performance). Fig. 2 shows an excerpt of this ontology.[2]

Note that the metric concept in this ontology was further categorised into *promotes* and *inhibits metric*. Promotes metric denotes a metric that should be maximised when promoting performance, e.g.,"throughput". Inhibits metric denotes a metric that should be minimised when promoting performance, e.g., "overhead time".

3.2. Analysing architectural pattern descriptions

Information Extraction deals with the identification and selection of relevant entities and the relationships between them in order to make them more accessible for further manipulations [15,16]. Unlike Information Retrieval, which concerns how to identify relevant documents from a document collection, Information Extraction produces structured data ready for post-processing, which is crucial to many applications of Web mining and searching tools. In most cases these activities concern processing human language texts by means of natural language processing techniques.

We constructed a computable model in Java to assist software architects to analyse a set of architectural patterns descriptions based on the GATE (General Architecture for Text Engineering) framework [44]. GATE is a well-known suite of tools used for all sorts of natural language processing tasks including Information Extraction from textual data.

GATE includes a family of processing components for performing various Information Extraction processing tasks such as tokenisation, semantic annotation or verb phrase chunking. Some of these components are depicted, within the context of the computable model we built, in Fig. 3 (see the GATE API box).

The computable model implements a pattern analysis process composed of three sequential phases, namely, (*i*) Corpus preparation, (*ii*) the Named entities recognition and (*iii*) Analysis resolution. Fig. 3 also shows how some of these phases relate to the GATE processing components. Next these phases are described.

Corpus preparation. The objective of the Corpus preparation phase is to obtain from each pattern description, in a corpus of architectural patterns description, the morphologic and syntactic structure of each one of its sentences. A sentence is a sequence of tokens; its beginning is denoted by a token starting with a capital letter and the end by a newline character or a dot. Tokens are typically complete words, but can also be symbols. As Fig. 3 shows, this phase is supported by the GATE Tokeniser and Sentence Splitter processing components. The Tokeniser splits text documents into tokens, annotating the tokens in different categories and storing the length of the token and other morphologic information such as capitalization. Tokens' annotations specify what a token of extracted text represents, e.g., a noun, a verb, an adjective. On the other hand, the Sentence Splitter splits the text into sentences according to punctuation.

Named entities recognition. The main purpose of this phase is to locate and classify (complex) named entities of interest. Generally speaking, named entities are elements in text belonging to pre-defined categories or pre-defined linguistic patterns. As shown in Fig. 3, this phase is supported by the OntoGazetteer and the JAPE Transducer processing components.

In general terms, a Gazetteer is a processing component that finds occurrences of named entities of interest in text by using a set of lists containing these entities. An OntoGazetteer, or an specialised gazetteer that finds occurrences of named entities of interest and annotates them with the corresponding class in the defined ontologies,

[1] More information about the domain-specific verbs, modal verbs, adverbs, and adjectives used in this work is available in https://goo.gl/IcTFeH.
[2] More information about the performance-specific concepts used in this work is available in https://goo.gl/IcTFeH.

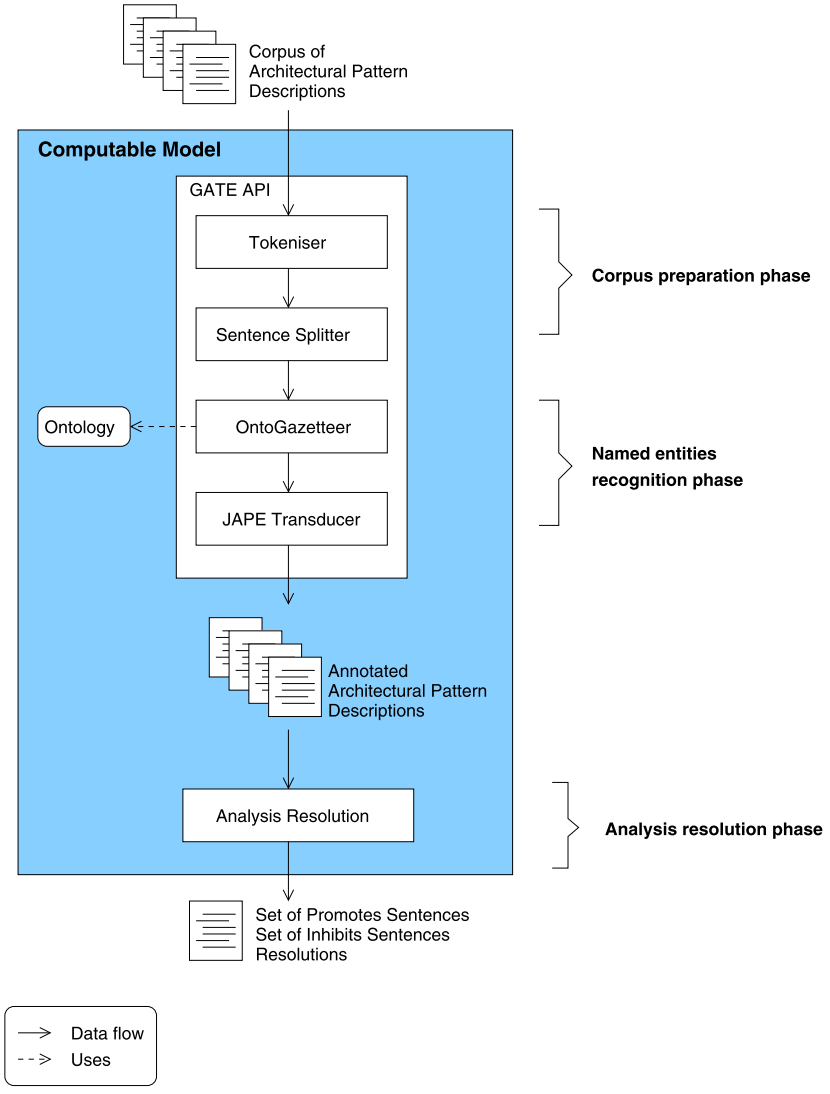

Fig. 3. Elements of the computable model for analysing pattern descriptions and their relation to the phases of the pattern analysis process.

is used in this work. For example, the named entity "concurrency" could be annotated with the "tactic" class in the performance ontology since "concurrency" *is-a* "tactic" in this ontology. Similarly, the named entity "increase" should have been annotated with the "promotes verb" class in the English ontology since "increase" *is-a* "verb" and this verb *is-a* "promotes verb" in this ontology.

In this work, complex entities of interest denote complete sentences, or sentence fragments, attending some specific structure. Thus, the recognition of these sentences is based on linguistic patterns and performed by the JAPE Transducer, which is a processing component that executes JAPE (Java Annotation Patterns Engine) grammars. JAPE is a pattern matching language that uses regular expressions to match linguistic patterns found in annotations on text documents.

Two sets of custom JAPE grammar rules have been defined: the *promotes* and the *inhibits* sets. These sets contain rules whose objective is to analyse architectural pattern descriptions with regard to whether they contain sentences about performance being promoted or inhibited, respectively. The promotes set contains eight rules and the inhibits set contains five rules. All the rules in these sets refer to the annotations generated by the OntoGazetteer, and evaluate the English and Performance ontologies.

A JAPE rule consists of a left-hand-side (LHS) and a right-hand-side (RHS) separated by an arrow ($--> $). The LHS specifies the pattern to look for in the text and the RHS specifies statements to produce new annotations out of the detected patterns. To illustrate the former, consider the following excerpt of an architectural pattern description, which indicates that it promotes performance:

*"…such a configuration **can** further **increase** system **performance** and **throughput**, as some filter instances can …"*

Looking at the English grammar concepts in this sentence as well as the ones defined in the ontologies, this sentence could be rewritten as follows:

*"…such a configuration **no_doubt_modal_verb_of_possibility** conjunction **promotes_verb** system **performance** conjunction **promotes_metric**, as some filter instances can …"*

The following is an example of one of the JAPE rules contained in the promotes set. This rule detects whether a sentence in a pattern description contains the linguistic pattern in the rewritten sentence above:

```
1  Options : control = appelt
2  Rule : promotes_01
3  (
4     (
5        { No_Doubt_Modal_Verb_Possibility }
6        { Conjunction }?
7        )
8     { Promotes_Verb }
9        (
10       { Token.kind==word, Token.string == ''system'' } |
11       { Token.kind==word, Token.string == ''application'' }
12       ) ?
13    { Performance }
14       (
15       { Conjunction }
16       { Promotes_Metric }
17       ) ?
18 )
19  : prom_01
20 -->
21 : prom_01 . promotes = { kind = ''NDMVP-PV-P-PM'', rule = ''promotes_01'' }
```

This rule uses the matching style "`appelt`" (line 1), which defines how to deal with annotations that overlap or where multiple matches are possible for a particular sequence, e.g., from all the rules that match a region of the description starting at some point, the one which matches the longest region is fired. The rule is entitled "`promotes_01`" (line 2), and it will match text starting with a modal verb with a "`No_Doubt_Modal_Verb_Possibility`" annotation (line 5), followed optionally by a token with a "`Conjunction`" annotation (line 6). Then, a verb with a "`Promotes_Verb`" annotation is required (line 8), followed optionally by a "`system`" or "`application`" token (lines 9–12). A token with a "`Performance`" occurring after it is also required (line 13). The rule also specifies that the "`Performance`" annotation be followed by optional text annotated as "`Conjunction`" and "`Promotes_Metric`" (lines 14–17). Once this rule has matched a sequence of text, the entire sequence is allocated a label by the rule. In this case, the label is "`prom_01`" (line 19). On the RHS (line 21), this span of text is referred using the label given in the LHS, "`prom_01`". It is said that the pattern specified by this rule will be awarded an annotation of type "`promotes`". This annotation allows, for example, the recovery of all the sentences containing a promotes pattern. "`kind`" is an attribute of this rule with the value set to "`NDMVP-PV-P-PM`" which denotes a specific combination of ontology concepts, i.e., no doubt modal verb of possibility (NDMVP), promotes verb (PV), performance (P) and promotes metric (PM). This attribute is particularly useful in the analysis resolution phase (described next). "`rule`" is another attribute of this rule with the value set to "`promotes_01`"; the purpose of the "`rule`" attribute is simply to ease the process of manual rule validation.

Analysis resolution. The main objective of the Analysis resolution phase is to deliver, for each analysed architectural pattern in the corpus, the promotes and inhibits sentences sets identified in the Named entities recognition phase. A resolution on whether each one of these patterns promotes or inhibits performance is also given. As shown in Fig. 3, this phase is supported by the Analysis Resolution component. This is a built-in component implementing the computational logic required in this phase.

For each pattern, the resolution on whether it promotes or inhibits performance is supported by comparing the values of two aggregate scores obtained from applying a scoring function to the promotes sentences and inhibits sentences sets respectively. For example, if the resolution is that the analysed architectural pattern promotes performance, then the value of the aggregate score of the promotes sentences set should be higher than the value of the aggregate score of the inhibits sentences set. As explained in Section 3.2 all these sentences can be easily identified via the annotation types "`promotes`" or "`inhibits`".

The defined scoring function takes as input the *concept combination* of a sentence and returns a value between 0 and 1. This value denotes the degree of certainty about the fact that the sentence is telling performance is being promoted (0 denotes a minimum degree of certainty and 1 denotes a maximum degree of certainty). Considering the former, the aggregate score value *aggScore* obtained from applying a scoring function *score* to, for example, the set of promotes sentences can be defined as follows:

$$aggScore = \sum_{i=1}^{n} score(concept\_combination\_kind_i)$$

where: *concept_combination_kind* is the value of the "`kind`" attribute of the sentence i in the promotes set.

In the language of architectural patterns, promotes or inhibits verbs like "increase" or "decrease" by themselves are meaningless. In this case, what these verbs mean depends on whether they precede or go after certain other concepts. Consider, for example, the following sentence fragments:

Sentence 1: ...***can*** *increase* ***throughput*** ...
Sentence 2: ...***might*** *increase* ***throughput*** ...
Sentence 3: ...***can*** *increase* ***overhead time*** ...
Sentence 4: ...***might*** *increase* ***overhead time*** ...

Regarding promoting performance, sentences 1 and 2 should be scored differently as, despite the fact that both include the promotes verb "*increase*", the "***might***" modal verb of possibility in the latter sentence makes it denote a slight possibility of promoting rather than inhibiting. Note also that, despite the fact that both sentences 3 and 4 include the promotes verb "*increase*", they actually describe performance being inhibited because of the occurrence of the inhibits metric "***overhead time***". As explained in Section 3.1, an inhibits metric should be minimised when promoting performance.

The following are examples of score functions for the concept combinations appearing in the sentences lines 1–4:

Sentence 1: $score(NDMVP - PV - PM)$
Sentence 2: $score(DMVP - PV - PM)$
Sentence 3: $score(NDMVP - PV - IM)$
Sentence 4: $score(DMVP - PV - IM)$

where:
NDMVP and *NDMVP* are a no doubt and a doubt modal verb of possibility, respectively.
PV is a promotes verb.
PM and *IM* are a promotes and an inhibits metric, respectively.

The rationale for assigning scores to a concept combination is based on the approaches described in [33,34]. In general terms, the score of a concept combination is calculated by aggregating the scores of its individual concepts. In our work, the scores of individual concepts are fixed pre-defined values obtained from three senior software architects. The average is used to obtain the score of a concept. Some example scores we obtained are $score(NDMVP) = .84$ and $score(DMVP) = .22$.

4. Evaluation and discussion of results

Having created a computable model for analysing architectural pattern descriptions, an evaluation to assess it was applied. In order to do that, we adopted a classical evaluation approach for information extraction systems [16,15]. In general in this evaluation a system is assessed in order to see how it behaves with regard to an exemplar of the ideal analysis's output and how it compares to humans performing the same task. Thus, we assessed the defined computational model to see how it behaved with regard to an exemplar of the ideal analysis' outputs and how it compared to both inexperienced and experienced software architects performing the same analysis task.

In what follows, we provide the details of this evaluation and discuss the obtained results. For clarity and length, the focus in this section is on the performance quality attribute.

4.1. Evaluation

Participants Three groups of fifteen people each participated in this evaluation (forty-five participants in total). The first group, called *Undergraduates*, consisted of fourth-year students of a Software Engineering undergraduate program. The second group, called *Masters*, consisted of second-year students in a Software Engineering master's degree program. The third group, which is called *Professionals*, consisted of junior software developers in a software development company graduated mostly from Computer Science programs.

The participants inclusion criteria capture the attributes that we expect the inexperienced software architects that are potential users of the proposed approach should have, see Table 2. The reasons of having three groups are: (i) we have access to both, students of these two academic programs and junior software developers in a software company and (ii) in these communities are people meeting the inclusion criteria. The size of these groups was meanly driven by the availability of junior software developers in the software company. It was hard to get more than fifteen software developers and we wanted to keep the groups' size the same. While fifteen participants per condition is not a large enough sample to overcome variation in measurements, we believe it is a reasonable number to start with, provided the types of metrics we estimated.

Table 2
Participants profiles.

	Years of design experience	Number and type of projects	Knowledge and Activities
Undergraduates	1 to 3	1 to 4 undergraduate level projects.	Know some design patterns. Have contributed to define the design of these projects.
Masters	1 to 3	1 to 4 graduate level projects.	Know some design patterns. Have contributed to define the design of these projects.
Professionals	1 to 3	1 to 4 industry projects.	Know some design patterns. Have contributed to define the design of these projects.

Data set Some of the metrics used in this evaluation only can be calculated if a *gold standard* is available against which the analyses' outputs are compared. A gold standard is an exemplar of the ideal analysis' outputs. In this work, the gold standard was created by a senior architect with more than ten years' experience. The gold standard included ten architectural pattern descriptions. The length of each pattern ranged from two to three pages. Three of them were written by different authors; seven corresponded to patterns promoting performance.

The definition of the gold standard' size was driven by considering the length of each pattern as well as existing guidelines on task duration in controlled experiments of software engineering tools with human participants, e.g., [45]. Based on an average of four minutes per page, if a two-page pattern takes a total of eight minutes to read and understand, ten pattern descriptions would require less than two hours.

Analyses In this evaluation, two types of analyses were performed: a *manual analysis* and an *automated analysis*. The manual analysis was performed by the three groups of people described in Section 4.1. The automated analysis was performed by the constructed computational model described in Section 3.2.

In the manual analysis we asked the participants to read the set of 10 pattern descriptions, contained in the gold standard, and provide a resolution for each one on whether it promotes or inhibits performance. Reading the pattern and providing a resolution are tasks that inexperienced architects will do in practice for pattern selection. These tasks were performed in three stages that correspond to scenarios with different levels of difficulty.

Stage 1. Reading one pattern description. This pattern promoted performance.
Stage 2. Reading three patterns descriptions, of the same pattern, written by different authors. The pattern promoted performance.
Stage 3. Reading six patterns descriptions, of different patterns, written by the same author. Four of these patterns promoted performance; two patterns did not.

We also asked the participants to underline the sentences that they considered to be indicating whether performance was promoted or inhibited.

For the automatic analysis the computational model was executed on the same set of ten pattern descriptions used in the manual analysis.

Employed metrics In order to evaluate the outputs of both the manual analysis and the automated analysis, two metrics were employed: *time* and *recall*.

The time metric was used to measure the time spent performing the analyses of the architectural pattern descriptions. Recall is a well-known information retrieval/extraction metric [46]. Recall represents the ratio of correctly identified information items to the total number of correct information items in a data set. In the context of this work, recall measures how many of the correct promotes and inhibits sentences in the data set are identified. A high recall value is specific to an analysis capable of identifying most of the correct sentences in the data set. Any correct sentence in the data set that is not identified will decrease the recall value. In our context, recall important because we want to find all correct promotes and inhibits sentences in pattern descriptions.

4.2. Discussion of results

In this section we discuss the metrics values obtained by three groups, namely Undergraduates, Masters and Professionals, when performing manual analyses of patterns. The results of the automated analyses are also discussed.

Fig. 4 shows the time spent in the Stage 1 of the manual analysis. As can be observed, medians are about the same in the three groups, i.e., 9 and 10 minutes. The minimum recorded time for analysing one pattern was 6 minutes for the Undergraduates group and 3 minutes for both the Masters and Professionals groups. On the other hand, the maximum time for analysing one pattern was 18 minutes for the Undergraduates group, 14 minutes for the Masters group and 12 minutes for the Professionals group. In the figure it is also observed that the time data distribution of the Undergraduates group is notoriously positively skewed, meaning that the time data constitute higher frequency of high value times. On the contrary, the time data distribution of the Professionals group is negatively skewed, meaning that the time data constitute higher frequency of low value times.

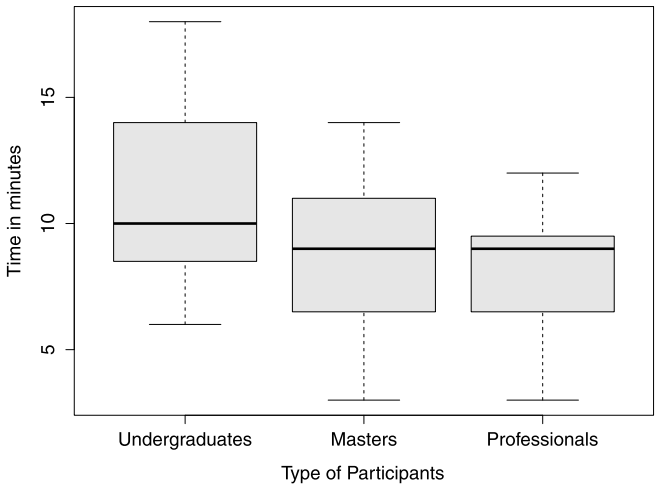

Fig. 4. Analysis time values obtained in Stage 1.

Table 3
Recall values obtained in Stage 1.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Undergraduates	.86	.57	.29	.86	.71	.57	.71	1	.57	.86	.57	.86	.71	1	1
Masters	1	.60	.53	.33	.40	.80	.60	.93	1	.20	1	.67	.47	.73	1
Professionals	.75	.5	.65	.85	.60	.60	.60	1	.85	.45	1	1	.75	.40	1

Table 4
Average and standard deviation values for analysis time and recall metrics obtained in Stage 1, for both manual and automated analyses.

	Manual Analysis (average)	Manual Analysis (standard deviation)	Automated Analysis
Analysis Time Undergraduates	660 sec	231 sec	40 sec
Analysis Time Masters	480 sec	197 sec	
Analysis Time Professionals	420 sec	140 sec	
Recall Undergraduates	.74	.20	.96
Recall Masters	.68	.27	
Recall Professionals	.73	.21	

Table 3 shows the recall values obtained in Stage 1. As can be seen, eleven participants obtained perfect values for recall: P8, P14 and P15 from the Undergraduates group; P8, P11, P12 and P15 from the Masters group; P1, P9, P11, P15 from the Professionals group. That is, eleven out of forty five participants recovered all existing relevant sentences.

Table 4 shows the average and standard deviation values for analysis time and recall metrics obtained by the participants in Stage 1. The table also shows the values for analysis time and recall obtained by performing the automated analysis. It should be noted that the automated analysis increased recall and reduced analysis time in comparison to the manual analysis. In this stage, all the participants resolved correctly that the analysed pattern promoted performance. The resolution was the same for the automated analysis.

Fig. 5 shows the time spent in Stage 2 of the manual analysis. As can be observed, the median tends to decrease in each group, i.e., 31, 28 and 25 minutes. The minimum recorded time for analysing three descriptions of the same pattern written by different authors was 17 minutes for the Undergraduates group and 14 minutes for both the Masters and Professionals groups. On the other hand, the maximum analysis time in this stage was 55 minutes for the Undergraduates group, 38 minutes for both the Masters group and 35 minutes for the Professionals group. In the figure it is also observed that the time data distribution of the Undergraduates group is positively skewed, meaning that the time data constitute higher frequency of high value times. On the contrary, the time data distribution of the Masters and Professionals group is negatively skewed, meaning that the time data constitute higher frequency of low value times.

The values for recall in Stage 2 are shown in Table 5. Thirteen participants, out of forty five, recovered all existing relevant sentences in the analysed pattern descriptions: P1, P9, P11 and P15 from the Undergraduates group; P1, P6, P9, P11, P13, P14, P15 from the Masters group; P10, P15 from the Professionals group.

Table 6 shows the average and standard deviation values for analysis time and recall metrics obtained by the participants in Stage 2. The values of recall decreased compared to Stage 1, indicating that students performed less well when analysing heterogeneous pattern descriptions. The same situation happened with the automated analysis. However, the automated analysis achieves better recall and time values in comparison to the manual analysis. In this stage, thirteen participants

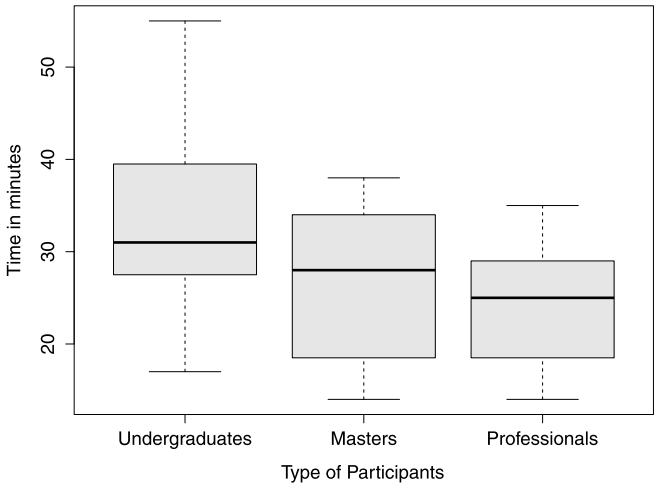

Fig. 5. Analysis time values obtained in Stage 2.

Table 5
Recall values obtained in Stage 2.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Undergraduates	1	.60	.53	.33	.40	.80	.60	.93	1	.20	1	.67	.47	.73	1
Masters	1	.67	.93	.93	.93	1	.47	.93	1	.40	1	.53	1	1	1
Professionals	.73	.53	.60	.53	.33	.47	.33	.20	.53	1	.47	.40	.93	.80	1

Table 6
Average and standard deviation values for analysis time and recall metrics obtained in Stage 2, for both manual and automated analyses.

	Manual Analysis (average)	Manual Analysis (standard deviation)	Automated Analysis
Analysis Time Undergraduates	33 min	10.7 min	56 sec
Analysis Time Masters	25 min	8.5 min	
Analysis Time Professionals	23 min	6.9 min	
Recall Undergraduates	.68	.27	.77
Recall Masters	.85	.22	
Recall Professionals	.59	.25	

Table 7
Recall values obtained in Stage 3.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Undergraduates	1	1	.71	.57	.57	.29	1	.71	1	.86	.86	.71	.86	.57	.71
Masters	.73	.53	.60	.53	.33	.47	.33	.20	.53	1	.47	.40	.93	.80	1
Professionals	.1	.15	.25	.25	.35	.50	.35	1	.40	.45	1	1	.70	.75	1

resolved that analysed pattern promoted performance and it was correct. One participant resolved that it did not. One participant resolved that he did not know. The automated analysis resolved that the pattern promoted performance.

Fig. 6 shows the time spent in Stage 3 of the manual analysis which consisted in analysing six different pattern descriptions written by the same author. As can be seen, the median tends to decrease in each group, i.e., 35, 32 and 27 minutes. The minimum recorded time for the analysis was 24 minutes for the Undergraduates group, 16 minutes for the Masters group and 21 minutes for the Professionals group. On the other hand, the maximum analysis time in this stage was 44 minutes for the Undergraduates group, and 35 minutes for the Masters group and 36 minutes for the Professionals group. Regarding the time data distribution, it can be seen that Undergraduates group's data is positively skewed, meaning that the time data constitute higher frequency of low value times. On the contrary, Masters group's data is negatively skewed, meaning that the time data constitute higher frequency of low value times

Regarding recall, Table 7 shows that ten participants recovered all existing relevant sentences: P1, P2, P7, P9 Undergraduates group; P10, P15 from the Masters group; P8, P11, P12, P15 Professionals group.

Table 8 shows the average and standard deviation values for analysis time and recall metrics obtained by the participants in Stage 3. In general, the values of recall decreased compared to Stage 2. The values for analysis time and recall obtained by the automated analysis were better in comparison to the manual analysis. However, recall is not a great deal better,

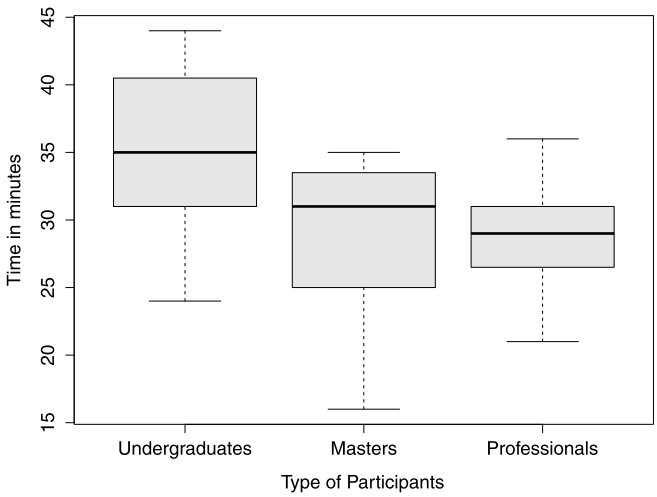

Fig. 6. Analysis time values obtained in Stage 3.

Table 8
Average and standard deviation values for analysis time and recall metrics obtained in Stage 3, for both manual and automated analyses.

	Manual Analysis (average)	Manual Analysis (standard deviation)	Automated Analysis
Analysis Time Undergraduates	35 min	6 min	75 sec
Analysis Time Masters	28 min	6.4 min	
Analysis Time Professionals	28 min	4.4 min	
Recall Undergraduates	.76	.20	.75
Recall Masters	.59	.25	
Recall Professionals	.55	.33	

which exposed some limitations in the defined JAPE grammar rules. However, in the automated analysis the resolution on whether the analysed patterns promoted or inhibited was correct for all of them. This was not the case for manual analysis.

4.3. Other aspects to consider

In previous sections we have shown that the proposed approach exposes good characteristics. However, the nature of the techniques utilised makes it having some intrinsic limitations that are important to discuss. Next we do it.

In this work we used an ontology to define relevant concepts that are expected to be discovered in textual pattern descriptions. Ontologies can be created adopting one (or more) of the following methods: manual, semi-automatic and automatic [47]. In this work the ontology was created manually by two domain experts. This process required a considerable amount of time and effort.

Similarly in this work we used JAPE rules to analyse architectural pattern descriptions with regard to whether they contain sentences about performance being promoted or inhibited. The definition of JAPE rules was also a time-consuming process as the rules were created manually by the same two domain experts. In this case, it does not seem possible to define these rules in a (semi-) automatic manner.

Both processes, ontology and JAPE rules definition, took the domain experts four months in total. However, this time was required only once and the resulting ontology and rules were reused many times to analyse a variety of architectural pattern descriptions; including domain-specific ones.

5. Conclusions and future work

In this paper we described an approach, based on knowledge representation and information extraction, to automate the analysis of architectural pattern descriptions with respect to specific quality attributes. The approach aims to help inexperienced software architects with determining whether specific quality attributes are promoted or inhibited, which is useful for pattern selection during architectural design. Through experimentation, it was shown that the approach exhibited good performance and recall.

Several efforts have been oriented towards providing tools and frameworks to help architects during architectural design. However, the proposed approach differs from them in that (i) it focuses on supporting quality attribute-oriented pattern selection during the design process; (ii) it is not limited to the use of a pre-defined and closed pattern repository and (iii) it does not require an extensive specification of the design problem. Because of the former, it can be easily integrated into existing architectural design methods such as Rozanski and Woods' [3], ADD [1] and Microsoft's Technique for Architecture

and Design [4]. Furthermore, our approach can deal with new pattern types without significant work, as the descriptions of these new patterns do not contain a significant number of new concepts.

Future work focuses on the following areas: (i) considering combinations of quality attributes; (ii) improving the JAPE rules; (iii) improving the sentence scoring approach, (iv) automating ontology population, (v) improving tool support. Further details are provided next.

In real life, architectural design requires considering a number of quality attribute requirements. Depending on the system, these attributes could have different meanings and importance. In its current version, the described approach considers quality attributes in an isolated manner. That is, it does not support pattern analysis with respect to multiple quality attribute requirements. For example, in some situations the architect could require a pattern promoting performance and security, but prefer maximising performance, perhaps at the expense of security. The notion of *utility preferences*, as in utility theory [48], to specify priorities between the quality attributes could allow a pattern analysis in a "utility-theoretic" way.

As shown in Table 8 the values for time and recall metrics obtained by the automated analysis in Stage 3 were not much better in comparison to the manual analysis, due to some limitations in the defined JAPE grammar rules. We are currently analysing these metrics in order to identify a set of general shortcomings and define a set of possible improvements.

Regarding improving the sentence scoring approach, future investigation might look into other ways of sentence scoring to cover other concepts in the ontology, and determine how best to scale up the speed of analysis without sacrificing recall.

In our approach, ontology building required a considerable amount of time and effort, as it was performed manually by domain experts. Despite having a "stable" version of the ontology, because of the incredible speed with which knowledge develops in the real world, having an "up to date" version is a never-ending goal. To overcome this problem, many methods to automatically or semi-automatically supplement extant concepts from corpus data have been developed, e.g., [49,47,50]. We are currently studying these approaches.

Finally, the approach presented in this paper is automated by computable model that works as a prototype tool. We have performed some user tests of our tool with junior software developers in a software development company. Future work includes performing more tests and attending the resulting feedback in order to improve the tool and make it publically available.

References

[1] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, W. Wood, Attribute-Driven Design (ADD), Version 2.0, Tech. Rep. CMU/SEI-2006-TR-023, Software Engineering Institute, Carnegie Mellon University, 2006, http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8147.
[2] A. Lattanze, Architecting Software Intensive Systems: A Practitioners Guide, 1st edition, Auerbach Publications, 2008.
[3] N. Rozanski, E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Addison-Wesley Professional, 2005.
[4] M. Patterns, Microsoft Application Architecture Guide, 2nd edition, Microsoft Press, 2009.
[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-oriented Software Architecture: A System of Patterns, John Wiley & Sons, Inc., 1996.
[6] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Longman Publishing Co., Inc., 2002.
[7] T. Erl, SOA Design Patterns, 1st edition, Prentice Hall PTR, 2009.
[8] R. Daigneau, Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services, 1st edition, Addison-Wesley Professional, 2011.
[9] N. Sawant, H. Shah, Big Data Application Architecture Q&A: A Problem – Solution Approach, 1st edition, Apress, 2013.
[10] S. R. International, Speed Reading Facts, http://www.execuread.com/facts/, 2015.
[11] D. Miner, A. Shook, MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems, 1st edition, O'Reilly Media, Inc., 2012.
[12] A. Homer, J. Sharp, L. Brader, M. Narumoto, T. Swanson, Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications, Microsoft Patterns and Practices, 2014.
[13] B. Aliaksandr, A survey of existing approaches for pattern search and selection, in: Proceedings of the 15th European Conference on Pattern Languages of Programs, ACM, New York, NY, USA, 2010, pp. 1–13.
[14] R. Brachman, H. Levesque (Eds.), Readings in Knowledge Representation, Morgan Kaufmann Publishers Inc., 1985.
[15] M. Pazienza, Information Extraction: Towards Scalable, Adaptable Systems, Information Extraction: Towards Scalable, Adaptable Systems, vol. 1714, Springer, 1999.
[16] J. Cowie, W. Lehnert, Information extraction, Commun. ACM 39 (1) (1996) 80–91.
[17] C. Dhaya, G. Zayaraz, Development of multiple architectural designs using ADUAK, in: Proceedings of the International Conference on Communications and Signal Processing, 2012, pp. 93–97.
[18] M. Barbacci, R. Ellison, A. Lattanze, J. Stafford, C. Weinstock, W. Wood, Quality attribute workshops (qaws), Tech. Rep. CMU/SEI-2003-TR-016, Software Engineering Institute, Carnegie Mellon University, 2003.
[19] F. Jabali, S. Sharafi, K. Zamanifar, A quantitative algorithm to select software architecture by tradeoff between quality attributes, Proc. Comput. Sci. 3 (2011) 1480–1484.
[20] H. Hadaytullah, S. Vathsavayi, O. Raiha, K. Koskimies, Tool support for software architecture design with genetic algorithms, in: Proceedings of the Fifth International Conference on Software Engineering Advances, IEEE Computer Society, 2010, pp. 359–366.
[21] P. Pelliccione, P. Inverardi, H. Muccini, CHARMY: A framework for designing and verifying architectural specifications, IEEE Trans. Softw. Eng. 35 (3) (2009) 325–346.
[22] S. E. Institute, ArchE-the Architecture Expert, http://www.sei.cmu.edu/library/abstracts/news-at-sei/architect200705.cfm, 2007.
[23] O. Zimmermann, U. Zdun, T. Gschwind, F. Leymann, Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method, in: Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, 2008, pp. 157–166.
[24] M. Weiss, A. Birukou, Building a pattern repository: benefitting from the open, lightweight, and participative nature of wikis, in: Proceedings of the Wiki4SE-Wikis for Software Engineering Workshop, 2008.

[25] M. Galster, A. Eberlein, M. Moussavi, Systematic selection of software architecture styles, IET Softw. 4 (5) (2010) 349–360.
[26] H. Kampffmeyer, S. Zschaler, Finding the pattern you need: the design pattern intent ontology, in: Model Driven Engineering Languages and Systems, in: Lecture Notes in Computer Science, vol. 4735, Springer, Berlin, Heidelberg, 2007, pp. 211–225.
[27] N.-L. Hsueh, J.-Y. Kuo, C.-C. Lin, Object-oriented design: a goal-driven and pattern-based approach, Softw. Syst. Model. 8 (1) (2009) 67–84.
[28] D.-K. Kim, C. El Khawand, An approach to precisely specifying the problem domain of design patterns, J. Vis. Lang. Comput. 18 (6) (2007) 560–591.
[29] S.M.H. Hasheminejad, S. Jalili, Design patterns selection: an automatic two-phase method, J. Syst. Softw. 85 (2) (2012) 408–424.
[30] S. Pearson, Y. Shen, Context-aware privacy design pattern selection, in: Proceedings of the 7th International Conference on Trust, Privacy and Security in Digital Business, Springer-Verlag, 2010, pp. 69–80.
[31] M. Weiss, H. Mouratidis, Selecting security patterns that fulfill security requirements, in: Proceedings of the 16th IEEE International Requirements Engineering Conference, IEEE Computer Society, 2008, pp. 169–172.
[32] S. C. for Biomedical Informatics Research, Protege, http://protege.stanford.edu/, 2014.
[33] F. Benamara, C. Cesarano, A. Picariello, D. Reforgiato, V.S. Subrahmanian, Sentiment analysis: adjectives and adverbs are better than adjectives alone, in: Proceedings of the International Conference on Weblogs and Social Media, ICWSM, AAAI Press, 2007, pp. 203–206.
[34] V.S. Subrahmanian, D. Reforgiato, AVA: Adjective-Verb-Adverb combinations for sentiment analysis, IEEE Intell. Syst. 23 (4) (2008) 43–50.
[35] K.S. Kipper, Verbnet: a broad-coverage, comprehensive verb lexicon, Ph.D. thesis, aAI3179808, 2005.
[36] K. Jassem, Semantic classification of adjectives on the basis of their syntactic features in Polish and English, Mach. Transl. 17 (1) (2002) 19–41, http://dx.doi.org/10.1023/A:1025512525185.
[37] B. Levin, English Verb Classes and Alternations: A Preliminary Investigation, University of Chicago Press, 1993.
[38] G. Miller, WordNet: a lexical database for English, in: Proceedings of the Workshop on Human Language Technology, HLT '93, Association for Computational Linguistics, 1993, p. 409.
[39] R. Dromey, A model for software product quality, IEEE Trans. Softw. Eng. 21 (2) (1995) 146–162.
[40] J. McCall, Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisition Manager, vol. 1, General Electric, 1977.
[41] ISO, ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model, Tech. rep., International Organization for Standardization, 2001.
[42] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 3rd edition, Addison-Wesley Professional, 2003.
[43] J. Ramachandran, Designing Security Architecture Solutions, John Wiley & Sons, Inc., 2002.
[44] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, GATE: An architecture for development of robust hlt applications, in: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02, Association for Computational Linguistics, 2002, pp. 168–175.
[45] A.J. Ko, T.D. Latoza, M.M. Burnett, A practical guide to controlled experiments of software engineering tools with human participants, Empir. Softw. Eng. 20 (1) (2015) 110–141.
[46] C.J.V. Rijsbergen, Information Retrieval, 2nd edition, Butterworth-Heinemann, 1979.
[47] Edward H.Y. Lim, James N.K. Liu, Raymond S.T. Lee, Knowledge Seeker – Ontology Modelling for Information Search and Management, Intelligent Systems Reference Library, Springer-Verlag, Berlin, Heidelberg, 2011.
[48] P. Fishburn, Utility Theory for Decision Making, Publications in Operations Research, Wiley, 1970, http://books.google.com/books?id=3CFwQgAACAAJ.
[49] W. Shen, J. Wang, P. Luo, M. Wang, A graph-based approach for ontology population with named entities, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, ACM, 2012, pp. 345–354.
[50] C. Giuliano, A. Gliozzo, Instance-based ontology population exploiting named-entity substitution, in: Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1, Association for Computational Linguistics, 2008, pp. 265–272.