Research on Identification and Refactoring Approach of Event-driven Architecture Based on Ontology

Li WANG[1,2]
[1]School of Computer Science and Engineering
Southeast University
Nanjing, China
[2]Jiangsu Automation Research Institute
Lianyungang, China
wangli1218@seu.edu.cn

Xiang-long KONG
School of Computer Science and Engineering
Southeast University
Nanjing , China
xlkong@seu.edu.cn

Xiao-fei WANG
NARI Group corporation
Nanjing, China
wangxiaofei@sgepri.sgcc.com.cn

Bi-xin LI ‡
School of Computer Science and Engineering
Southeast University
Nanjing , China
bx.li@seu.edu.cn

***Abstract*—Event-driven architecture is one of the common software architecture patterns. In the process of software evolution, the deviation and corrosion often occur to architecture, which leads to larger deviation between actual software architecture and design architecture. Therefore, it is of great significance to study the approach of software architecture identification and refactoring. To solve this problem, we propose an identification and refactoring approach of event-driven based on ontology, i.e., IRABO. We evaluated IRABO on 50 open-source projects and the results show that it performs effectively and efficiently.**

Keywords-Event-driven Architecture; Architecture Identification; Architecture Refactoring

I. INTRODUCTION

Appropriate pattern can solve the design problem of software architecture[1]. Event-driven architecture is a very popular architecture pattern at present, which is usually used in systems that require high agility and quickly response[2]. The event-driven architecture can fulfill that requirement quite well. But, in the process of software evolution, many factors may make the software architecture deviate from the original design, such as the change of requirements, the improvement of functions, et al. So, it is necessary to refactor the architecture. Architecture patterns provide a good direction for refactoring [3].

In this paper, we propose identification and refactoring approach of event-driven architecture based on ontology, i.e., IRABO, which consists of two parts. We firstly extract the dependency information from source code to build the program dependency graph. Then we convert the program dependency graph into RDF (The Resource Description Framework) triples to build the ontology of instance layer. Finally, we use the event-driven architecture usage specification to locate the refactoring point in the identification result, and refactor the software architecture.

To evaluate the effectiveness accuracy and efficiency of IRABO, we conduct experiments on 50 open-sourced projects with manual analysis approach. The results show that IRABO performs much better in terms of accuracy and effectiveness efficiency in our experiments. In summary, our paper makes the following novel contributions:

We put forward the ontology-based event-driven architecture pattern identification approach and the architecture refactoring approach based on event-driven architecture.

We build the experiment for ontology-based event-driven architecture identification and refactoring to verify the ontology-based event-driven pattern identification and refactoring approach.

II. APPROACH

In this section, we present the details of the identification and refactoring approach of event-driven based on ontology, i.e., IRABO. The technique comprises two main steps, identification approach of event-driven architecture based on ontology, refactoring approach of event-driven architecture based on ontology.

A. Identifying Event-driven Architecture Based on Ontology

Event-driven architecture identification based on ontology is essentially a process of abstract matching between source code and event-driven architecture. As presented in Fig.1, First, we use the source code analysis tool to extract the dependency information. Second, we use ontology to descript the dependency information to construct instance layer ontology; meanwhile, we use ontology to describe the structural behavior characteristics of event-driven architecture to construct concept layer ontology. The instance layer ontology and concept layer ontology form the Ontology Knowledgebase. We use ontology inference engine to process the Ontology knowledgebase to obtain the instance of event-driven architecture. Compared with other semi-automatic or manual approaches, IRABO can improve the accuracy and automation of event-driven architecture identification.

1) Construction instance layer ontology:

In this paper, the object of identification is Java projects. We choose JDT to generate an Abstract Syntax Tree, i.e., AST.

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61872078)
DOI:10.18293/SEKE2022-013

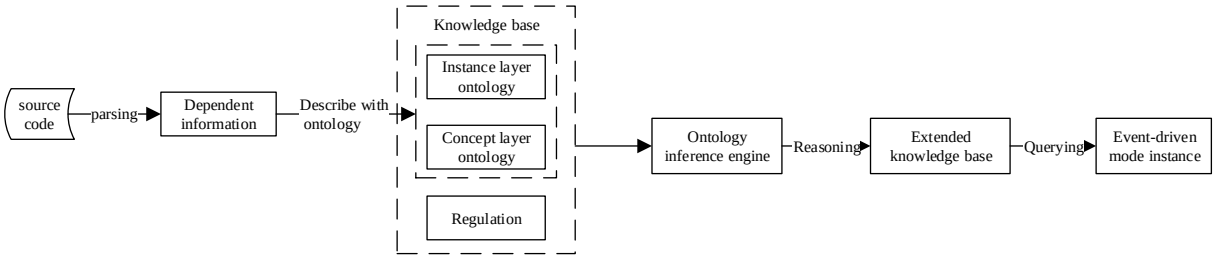

Figure 1. Event-driven architecture identification based on ontology

We extract the dependency information by traversing the ADT to build a dependency graph. As shown in Fig 2, the node represents the program entities, and the directed edge represents the dependency between program entities. We convert the nodes and directed edges into RDF triples set.

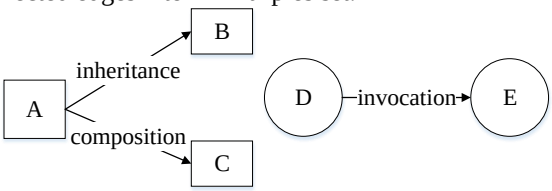

Figure 2. RDF triple set

2) Construction concept layer ontology:

In this paper, we choose the common ontology building Jena to build ontology of the event-driven architecture. First, we use ontology to describe the observer pattern and its specific application in event-driven architecture, thus indirectly describing the behavior characteristics of event-driven architecture. Second, we use ontology to describe the component reuse behavior of the event-driven system to the event-driven framework.

The event-driven architecture has three components: event, listener and event source. The listener acts as the observer, and the event source acts as the observed[3][4] [5]. The behavior characteristics of the event-driven architecture are shown in Fig 3.

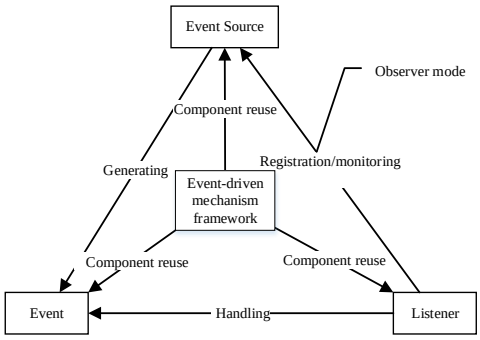

Figure 3. Behavior characteristics of event-driven architecture

An important behavioral feature of event-driven architecture is the component reuse behavior of event-driven framework, as shown in Fig 4. When programmers develop event-driven systems under the framework of event-driven mechanism, they only need to define the listeners through the listener interface, and inherit the sensible operating components under the framework to define their own event sources and the event classes under the framework to define their own events[6]. We describe the event-driven architecture indirectly by describing observer pattern and event-driven framework. We build ontology to describe the component reuse behavior of the event-driven framework on the ontology building platform.

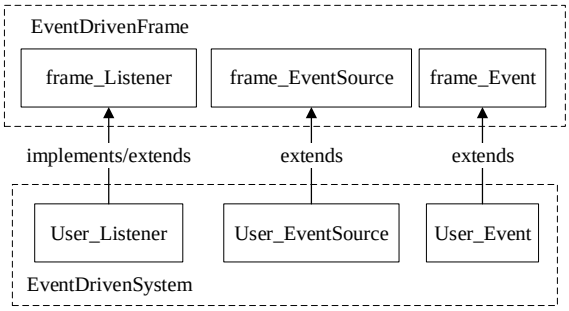

Figure 4. Event-driven pattern framework reuses behavior

3) Reasoning and inquiry

We reason and query based on ontology to match between the target system and the event-driven architecture, so as to obtain the event-driven architecture instance. We reason and query the model defined by Jena[7][8]. We use the ontology query function to obtain the instance of event-driven architecture in the extended ontology knowledgebase.

B. Refactor event-driven architecture

In this section, we refactor architecture based on event-driven architecture identification. As shown in Fig 5, we use the event-driven architecture violate specification to locate the refactoring points. Then we choose the corresponding refactoring scheme to eliminate or reduce the violation of the event-driven architecture, so as to obtain a new architecture. Repeat the steps until there is no violation of event-driven architecture in the target system.

Step 1: We locate the refactoring point in the identification result of event-driven architecture. Refactoring point is the violate specification of event-driven architecture. The event-driven architecture is a distributed processing pattern composed of highly decoupled event listeners with single responsibility. Therefore, the most important usage specifications of the event-driven architecture are the single responsibility of the listener specification and the distributed processing specification. The single responsibility of the listener specification requires a listener to handle only one type of events. If a listener class handles multiple types of events, the change of one event

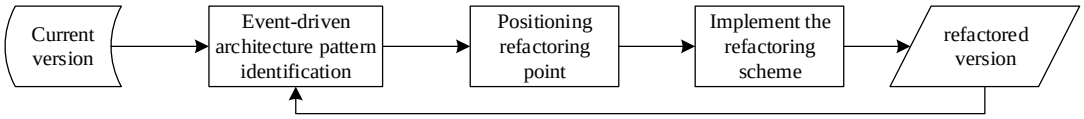

Figure 5. Refactoring process of event-driven architecture

handling method may weaken the handling ability of other events[9]. The distributed processing specification requires that the events are generated and processed in different classes. If a class is both an event source and a listener, it violates the distributed processing specification[5].

Step 2: we implement the scheme for the refactoring points. The appropriate refactoring scheme should specify the refactoring operation according to different refactoring points. In this paper, we propose two refactoring schemes, i.e., RS, for the refactoring points located by the single specification of listener responsibility and the distributed processing.

RS 1: The refactoring scheme for the single responsibility of the listener.

The refactoring scheme is proposed to eliminate the violation of the single responsibility of the listener. In this kind of violation, a listener class handles more than one type of events. The refactoring scheme is to split the listener class into several classes and let each class handle one type of events. As shown in Fig 6, we define a new empty listener class, and transfer the one of the events to the new class. At the same time, the corresponding dependencies are transferred.

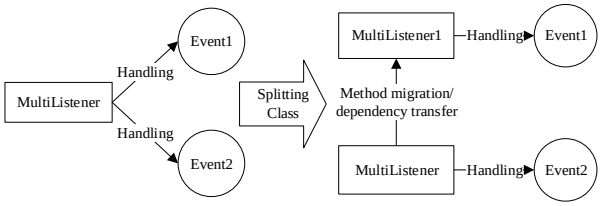

Figure 6. Split listener classes that handle various types of events.

RS 2: The refactoring scheme for distributed processing.

The refactoring scheme is proposed to eliminate the violation of the regulations of the distributed processing. In this kind of violation, the class is both an event source and a listener. The refactoring scheme is to split the class into two classes, one class is listener and the other is event source.

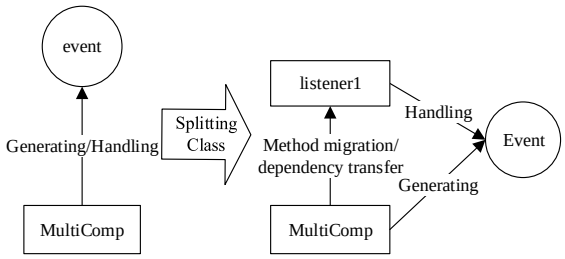

Figure 7. Splitting classes that are both event sources and listeners

As shown in Fig 7, we define a new empty listener class, and transfer the events handling to the new class. At the same time, the corresponding dependencies are transferred. The original class acts as an event source and the new class acts as a listener.

III. Experiment And Results

The identification and refactoring approach of event-driven architecture can identification and refactoring event-driven architecture based on ontology accuracy and efficiency, which can help developers understand and maintain software projects. In this section, we aim to answer the following research questions:

RQ1: How about the accuracy of the software architecture identification technique?

RQ2: How about the accuracy of the software architecture refactoring technique?

RQ3: How about the efficiency of the software architecture refactoring technique?

A. Experimental Setup

1) Subject projects

To answer the above research questions, we select 50 Java projects from GitHub and SourceForge according to the popularity of Java projects. These projects are more popularity with the key words, such as game, game engine, Java awt, Java swing and event-driven. We analysis the documents and source codes of these 50 projects manually to obtain the ground-truth architecture. We select freecol and shiro to analyze their refactoring points.

2) Measurement

We use Precision, Recall and Accuracy to measure the accuracy of software architecture identification technique. It is defined by the following formulas

$$P = \frac{TP}{TP + FP} \tag{1}$$

$$R = \frac{TP}{TP + FN} \tag{2}$$

$$A = \frac{TP + TN}{TP + FP} \tag{3}$$

Where P indicates Precision, R indicates Recall, A indicates Accuracy, TP, FN, FP and TN indicate four numerical values in the confusion matrix of identification results.

TABLE I. CONFUSION MATRIX OF IDENTIFICATION RESULTS

Manual analysis results	Identification result	
	Yes	No
Yes	*TP*	*FN*
No	*FP*	*TN*

We use Accuracy, CostRate and Effectiveness to measure the efficiency of the software architecture refactoring technique.

The Accuracy of refactoring point location is the proportion of correctly located refactoring points in all the refactoring points located by this technique.

The CostRate is the proportion of the number of classes to be refactored to the total number of classes in the object.

The Effectiveness is the proportion of eliminated refactoring points in all the refactoring points located by this technique.

3) Experimental *steps*

For each studied subject, we performed the following steps:

Step1: We collect 50 Java projects from GitHub and SourceForge with the key words.

Step2: For each selected project, we identify their architecture pattern manually to confirm wither they are event-driven architecture project.

Step3: For freecol and shiro, we obtain their ground-truth architecture manually to build the comparative experiments.

Step4: For freecol and shiro, we refactor their architecture base on ontology, and we collect all the results to analyze the accuracy and effectiveness.

Step5: For each event-driven architecture project, we obtain the refactoring points and process the refactoring schemes by manual analysis as reference, the effectiveness of the refactoring method based on event-driven architecture identification is evaluated through the accuracy of refactoring point positioning and refactoring cost rate.

In the experiments, we use computer with 64-bit Windows 10 and 8G memory. We use JDK1.8, Eclipse Neon 4.6.0, and MySql 5.6. The ontology inference engine witch we use is Jena 3.10.0.

B. Results analysis

RQ1: The accuracy of the software architecture identification technique

To evaluate accuracy of the event-driven architecture identification-based ontology, we apply the IRABO, and manual analysis work on the 50 projects. The manual analysis work of clone, freecol, jmonkeyengine, Jadventure, libgdx, AndEngine, overlap2d, GameHelper and Shiro is based on the source code and documents. The other 41 projects can only be analyzed according to the source code because of missing documents. Table II presents the results, "√" means the project is event-driven architecture, "×" means the project isn't event-driven architecture.

From Table III, we can find that there are 13 projects with event-driven architecture by manual analysis. There are 12 projects with event-driven architecture by IRABO identification. There are 8 projects whose manual analysis and RABO identification results are both event-driven architectures. The Precision, recall and accuracy of IRABO are 66.6%, 61.54% and 82%. There are 18% identification error rate of IRABO.

The reason of identification error rate of IRABO is false negative and false positive. The reason of false negative is as follows:

IRABO only considers the typical event-driven architecture when identifying the architecture, but it fails to identify the project with atypical event-driven architecture.

IRABO only considers the mainstream event-driven framework when identifying the architecture, but it fails to identify the non-mainstream event-driven framework.

TABLE II. IDENTIFICATION RESULTS

Project	IRABO	Manual analysis	Project	IRABO	Manual analysis	Project	IRABO	Manual analysis
clone	√	√	Terasology	×	×	blog	×	×
openbbs	×	×	pixel-dungeon	×	×	jnativehook	×	×
MyBlog	×	×	FunGameRefresh	×	×	jmonkeyengine	√	√
freecol	√	√	WorldEdit	×	×	Jadventure	×	√
terrier	×	×	JustWeEngine	×	√	jadx	×	×
lionengine	√	√	overlap2d	√	×	JHotDraw	×	×
junit4	×	×	StormPlane	×	×	libgdx	×	√
la4j	√	×	OpenRTS	√	√	AndEngine	×	√
okhttp	×	×	PretendYoureXyzzy	√	×	HikariCP	×	×
mybatis	×	×	Essentials	×	×	arthas	×	×
vert.x	×	×	GameHelper	×	×	Mosby	×	×
beautyeye	√	×	druid	×	×	latexdraw	×	×
symphony	×	×	SSH-master	×	×	MARIO	×	×
mockito	×	×	ssm-master	×	×	log4j	×	×
junit5	×	×	Examination_System	×	×	FXGL	×	×
litiengine	√	√	shiro	√	√	JabRef	×	×
inxedu	×	×	realm	×	×			

IRABO describes the event-driven architecture by describing the structural behavior characteristics of the observer pattern and its application. Therefore, the false identification of the observer pattern will lead to the false identification of the event-driven architecture.

TABLE III. EXPERIMENTAL RESULTS

Manual analysis	IRABO		
	Yes	No	Total
Yes	8	5	13
No	4	33	37
Total	12	38	50
Precision	66.67%		
Recall	61.54%		
Accuracy	82%		

The reason of false positives is as follows:

Architecture is the overall design of software. When the project partially implements the event-driven mechanism, IRABO would identify it as an event-driven architecture project.

We obtain the ground-truth architecture by manual analysis, and the false of manual analysis results will lead to false positives.

RQ2: The accuracy of the software architecture refactoring technique

We choose two typical event-driven architecture projects, freecol and shiro. We use IRABO to obtain the event-driven architecture instances of two projects compare with manual analysis results. The accuracy of IRABO is measured by Precision and Recall, as shown in Table IV.

TABLE IV. THE ACCURACY OF IRABO

Project	Component	*Precision*	*Recall*
freecol	audio monitor	75.45%	65.76%
	event	54.23%	44.54%
	Event source	58.51%	41.71%
shiro	audio monitor	83.35%	66.23%
	event	57.68%	46.54%
	Event source	65.43%	58.92%
Average value		65.78%	53.95%

From Table IV, we can find that the average Precision and Recall of IRABO are 65.78% and 53.95%. The false positives and false negatives in the event-driven architecture identification results are caused by event-driven architecture variants and false manual analysis result.

RQ3:The efficiency of the software architecture refactoring technique

a) Eliminate the single responsibility of listener specification violation

We positioned refactoring points that violate the single specification of listener responsibilities in all the event-driven architecture. We fined refactoring points in clone, freecol, lionengine and litiengine by IRABO and manual analysis. The Refactoring points positioned by IRABO and manual analysis work are shown in Table V. In clone, freecol, lionengine and litiengine, the Accuracy of IRABO is 50%, 39.1%, 46.2% and 60.2%, and the CostRate of IRABO is 0.035, 0.153, 0.172 and 0.272.

TABLE V. REFACTORING POINTS

Project	Classes	Refactoring points (IRABO)	Refactoring points (Manual)	Classes needing refactoring
clone	115	2	1	4
freecol	1224	192	75	188
lionengine	843	132	61	145
litiengine	445	88	53	121

We choose a refactoring point CanvasMouseListener in freecol, which violates the single specification of the listener specification. Then we refactor CanvasMouseListener by RS 1 to eliminate the single responsibility of listener specification violation.

b) Eliminate the distributed processing specification verification

We positioned refactoring points that violate the distributed processing specification in all the event-driven architecture. We fined refactoring points in reecol, lionengine and litiengine by IRABO and manual analysis. The Refactoring points positioned by IRABO and manual analysis work are shown in Table VI. In reecol, lionengine and litiengine, the Accuracy of IRABO is 75%, 65.2% and 77.3%, and the CostRate of IRABO is 0.009, 0.018 and 0.038.

TABLE VI. REFACTORING POINTS

Project	Classes	Refactoring points (IRABO)	Refactoring points (Manual)	Classes needing refactoring
freecol	1224	12	9	11
lionengine	843	23	15	15
litiengine	445	22	17	17

We choose a refactoring point BuildingPanel in freecol, which violates the distributed processing specification. Then we refactor BuildingPanel by RS 2 to eliminate the distributed processing specification verification.

IV. THREATS TO VALIDITY

Threats to external validity. The ground-truth architecture obtained by manual analysis is used to verify the accuracy of the architecture obtained by IRABO. Influenced by the ability of analysts or the complexity and scale of the project, the architecture obtained by manual analysis is subjective to some extent. That may threaten the accuracy of the software architecture identification and refactoring technique. To reduce this threat, we will select more excellent open-source projects of event-driven architectures; conduct a more comprehensive analysis to obtain ground-truth architecture more accurately.

Threats to internal validity. IRABO only considers the typical event-driven architecture when identifying the architecture. For the projects with atypical event-driven architecture, false negative and false positive may occur. To reduce this threat, we will consider more variants of event-driven architecture to build a more complete ontology knowledge base of event-driven architecture.

Limited by manpower and time, the projects selected in this paper are small-scale, which cannot verify the accuracy and effectiveness of this technique in large-scale projects. In the future work, we will repeat the experiments with more large-scale projects to reduce this threat.

V. Related Work

In the aspect of pattern identification and description of architecture, Mavridou Anastasia points out that architecture can be represented by logic and architecture style can be described by configuration[9]. Cortella Essav and others proposed to use logical predicates to model anti-patterns, and build an engine based on these logical predicates to detect anti-patterns in the target system[10]. Rabiaz et al. proposed a method of knowledge retrieval to identify instances of architecture patterns in software systems.[11]. The powerful ability of ontology description is exactly what is needed to describe the very high level of abstraction such as architectural patterns.[12]. Velasco-Elizondo P and others put forward an automatic analysis architecture model based on knowledge representation and information extraction, and then reconstructed the system according to the analysis results.[13]. The main problem of the existing architecture pattern identification and refactoring methods is the lack of a special method for event-driven architecture identification and refactoring. Therefore, this paper proposes an ontology-based pattern identification and refactoring method for event-driven architectures.

VI. Conclusion

In this paper, we present an approach of identification and refactoring approach of event-driven architecture based on ontology, i.e., IRABO. We identifying event-driven architecture based on ontology. We refactor event-driven architecture according to the usage specification of event-driven architecture. Experiments verify the accuracy of pattern identification based on ontology-based event-driven architecture and the effectiveness of the refactoring scheme. We evaluate IRABO by conducting experiments on 50 projects and compare with manual analysis work. The results show that IRABO perform efficiency and effectively. And there is still space for improvement of architecture recovery effectiveness. The follow-up work can start with the method of identification more variants of event-driven architecture to further improve the accuracy and effectiveness of software architecture recovery.

References

[1] Ta'id Holmes, and U. Zdun . Refactoring Architecture Models for Compliance with Custom Requirements[C]. ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems ACM, 2018.

[2] Elish K O , Alshayeb M . Using Software Quality Attributes to Classify Refactoring to Patterns[J]. Journal of Software, 2012, 7(2):p.408-419.

[3] Overbeek S , Janssen M , Bommel P V . Designing, formalizing, and evaluating a flexible architecture for integrated service delivery: combining event-driven and service-oriented architectures[J]. Service Oriented Computing&Applications, 2012, 6(3):167-188.

[4] Tragatschnig S , Stevanetic S , Zdun U .Supporting the evolution of event-driven service-oriented architectures using change patterns[J]. Information and Software Technology, (2018):133-146.

[5] Woodside M . Performance Models of Event-Driven Architectures[C]. CPE '21: ACM/SPEC International Conference on Performance Engineering ACM, 2021.

[6] Abel Gómez, Iglesias-Urkia M , Urbieta A , et al. A model-based approach for developing event-driven architectures with AsyncAPI[C].MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems. ACM, 2020.

[7] Yu Lei, Ma Hui, Wang Cheng. Research on equipment PHM knowledge ontology construction and semantic reasoning method[J]. ournal of Ordnance Equipment Engineering, 2019,40(S1):126-130.

[8] Lerlertvanich R, Vatanawood W. Facade Layer for Apache JENA[J]. Arpn Journal of Systems & Software, 2012, 2(11).

[9] A. Mavridou, E. Baranov, S. Bliudze, et al. Configuration logics: Modeling architecture styles[J]. Journal of Logical and Algebraic Methods in Programming, 2017, 86(1): 2-29.

[10] V. Cortellessa, A. D. Marco, C. Trubiani. An approach for modeling and detecting software performance antipatterns based on first-order logics[J]. Software & Systems Modeling, 2014, 13(1): 391-432.

[11] Rabinia Z , Moaven S , Habibi J . Towards a knowledge-based approach for creating software architecture patterns ontology[C].International Conference on Engineering & Mis. IEEE, 2016.

[12] Guessi M, Moreira D A, Abdalla G, et al. OntolAD: a Formal Ontology for Architectural Descriptions[C].ACM SAC 2015. ACM, 2015.

[13] Velasco-Elizondo P, Marín-Piña R, Vazquez-Reyes S, et al. Knowledge representation and information extraction for analysing architectural patterns[J]. Science of Computer Programming, 2016, 121: 176-189.