

The Knowledge Base for Automating the Architecting of Software Systems

Gleb Guskov, Anton Romanov, and Aleksey Filippov(✉)

Ulyanovsk State Technical University, Ulyanovsk 432027, Russia
al.filippov@ulstu.ru
https://ulstu.ru

Abstract. This article describes an approach to knowledge base (KB) formation. KB may be used for automating the software systems (SS) architecting based on the experience of previous projects.

Software architecting is the presentation of software systems in the form of design artifacts set and their architecture. It is possible to improve the quality of a new SS by attracting the experience of previous projects in a new SS developing process. The experience of previous projects is successful architectural solutions that contain the design organization KB.

That KB should be formed in the analyzing process of design artifacts extracted from previous projects: source code, project diagrams, data models, structured text resources, etc.

This article describes the design organization KB model and the model of the 1C: Enterprise 8 platform solution (1C application) as an example of a design artifact. Also, the article presented the method for KB fragments forming based on analyzing the process of 1C application and a method for generating use-case diagrams based on the KB content.

The set of experiments was executed to evaluate the adequacy of the proposed models and methods. Precision and recall metrics were used to evaluate the quality of the generated use-case diagrams. Precision is the presence of ex-pert diagram elements in the generated diagram. Recall is the presence of elements in the generated diagram that are absent in the expert diagram. According to the results of the experiments, the average value of accuracy is 0.875, and the completeness is 0.6.

Keywords: Architecting · Software systems · Experience of previous projects · Design artifacts · Design experience units

1 Introduction

Currently, the complexity of developing and maintaining software systems has much increased. Therefore, it is necessary to apply the principles and procedures of the architecting [1]. Architecting is the software system (SS) architecture creation based on understanding the essence and basic properties of a SS. The

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12951, pp. 272–287, 2021.
https://doi.org/10.1007/978-3-030-86970-0_20

properties of a SS determine the behavior of the system, its structure, and development approaches. These characteristics influence the stability, usefulness, and viability of the system [2–4]. An architecture description of a SS obtains during the architectural design process. An architecture description is used to coordinate the stakeholders involved in the development, implementation, support, and operation of the system.

Thus, requirements for the system are formed in the architectural designing of a SS for all stages of the life cycle: development, implementation, maintenance. The characteristics of the processes of development, implementation, operation, and maintenance of SS are highlighted.

A SS architecture can be considered as a concept of the primary properties of the system. A SS architecture can be represented as multiple representations: business representation, physical representation, technical representation [1]. An architecture description of a SS is defined in this study as a set of design artifacts obtained in a SS development. Design artifacts are the most primitive constructs that form an architecture description of a SS.

A SS architecture as a set of design artifacts allows systematizing architecture descriptions in different subject areas. The context of a SS defines many architectural requirements and constraints that determine the structure and characteristics of design artifacts [5]:

- components and elements of a SS;
- principles to the organization and interaction of an SS element;
- a SS structure;
- principles of a SS project;
- principles of a SS development within a life cycle.

Thus, an architecture description quality of a SS influences the development process and the SS quality. In the later stage of SS development are design errors discovered, that more resources are required to correct them [6].

It needs to use artificial intelligence methods to minimize errors in a SS at the early stages. Artificial intelligence methods and algorithms [7–9] involve extracting knowledge from design artifacts to build a knowledge base (KB) of a software company.

An architecture description of the current SS project can also be obtained by processing an architecture description of the previous projects [1,4]. Successful design decisions extracted from previous projects should be used in the process of working on new projects.

2 Knowledge Base of a Software Company

A modern large software company has a significant amount of completed projects. Each successfully implemented project is represented by a set of design artifacts obtained in the process of development. Design artifacts contain many experience units (EU) of highly qualified specialists who have been developing and designing SSs for many years. In this study, EU are understood as the implementation of domain entities and their business processes.

The EUs of previous projects is stored in various design artifacts:

- project documentation presented in the form of a weakly structured text in a natural language;
- set of design diagrams, usually presented in UML notation;
- source code;
- set of datamodels describes objects of a problem area, presented in the form of entities and relationships of relational databases;
- features of the project environment, usually presented in the form of wiki resources.

The complexity of the search for the necessary information depends on the amount of heterogeneous unstructured data. The experience of previous projects contained in design artifacts is not used on time. The use of design artifacts decreases the time of a SS development cycle and increases the quality of the developed SS.

The quality of a SS depends on the quality of execution of the development stages [6].

The central stage in the new SS development is the design stage. An architecture description and model of the developed SS are formed on the design stage. The quality of functioning of the whole complex of tasks assigned to a new SS depends on the quality of a new SS project. The following tasks should be solved at the design stage:

1. Search for key project participants.
2. Formation of the final architecture description of a new SS.
3. Object modeling and design of the basic elements of a new SS.

The formats and storage methods for various types of design artifacts are different, which complicates their analysis and the formation of KB. KB of a software company can automate the following tasks:

1. Best practices accumulation.
2. Extraction of frequently used EUs.
3. Evaluation of extracted EUs.
4. Definition of favorites in the choice of the EUs by designers, architects, and developers.
5. Determination of the frequency dynamics of various EUs usage.
6. Prototyping a project based on a set of EUs recorded in KB, etc.

2.1 Literature Review

Works [10–15] noted that using the experience of the previous projects at the initial stages of new project development is crucial. The solution to this problem can be based on intelligent methods and algorithms of design artifacts analysis to build the KB of a software company.

The specifics of the design experience located in the design artifacts of the previous projects require the formation of the KB of a software company with the structure based on an ontology system [16].

Works [17–21] note the importance of using ontological engineering in the design and development of SSs. In works [17,20] describes an approach to using ontologies instead of traditional modeling languages, such as UML, since ontologies allow controlling the logical integrity and consistency of the resulting model.

However, existing methods of forming a KB need experts in the problem area and knowledge engineering. This need leads to significant time costs for the formation of such a KB.

A large number of researchers are working in the field of extracting entities from program code [22–24]. Entities are understood as structural elements of the source code: files, classes, methods, variables, etc. The entities extraction is usually based on the abstract syntax tree analysis with a set of rules. In some cases, machine learning methods and neural networks are used for entity extraction.

In this study, entities are understood as objects of some problem area described in source code classes. Methods (business processes) in which entities are parameters or return value types are also extracted from the source code.

Therefore, the development of models, methods, and algorithms for building the KB of a software company in the automated analysis process of design artifacts is relevant.

The main difficulty of this approach is the necessity to unify design artifacts. Also, necessary to include in the KB information about the various subject areas within which previous projects were developed. The main objective of this research is to study the possibilities and limitations of data analysis and knowledge engineering methods for building KB for automating the process of the architectural design of SS. The methods for extraction of EUs in the KB content analyzing will be considered in subsequent papers.

Let's consider the ontology of the KB of a software company in more detail.

2.2 Domain Ontology

The domain ontology allows adapting the KB of a software company to the points of the subject areas. The domain ontology is the basis for creating a single set of concepts that unifies the entities of various subject areas.

The domain ontology is:

$$O^{dom} = \{O_1^{dom}, O_2^{dom}, \ldots, O_i^{dom}, \ldots, O_n^{dom}\}, \tag{1}$$

where n is the number of subject areas within which previous projects were developed;
O_i^{dom} is i-th ontology containing a description of the i-th subject area:

$$O_i^{dom} = \langle C^{dom}, T^{dom}, R^{dom}, F^{dom} \rangle, \tag{2}$$

where C^{dom} is a set of concepts;
T^{dom} is a set of terms that describe concepts;
F^{dom} is a set of interpretation functions that define a set of relationships R^{dom};
R^{dom} is a set of relationships:

$$R^{dom} = \{R^{dom}_{CC}, R^{dom}_{CT}\},$$

where R^{dom}_{CC} is a set of generalization relations that determine the hierarchy of concepts;
R^{dom}_{CT} is a set of association relationships between concepts and terms describing these concepts.

2.3 Ontology of Design Artifacts

The ontology of design artifacts contains formalized descriptions of design artifacts in the form of KB fragments. The central elements of this ontology are:

- entities are objects of a specific subject area, modeled using a design artifact, for example, a class, table, etc.;
- entity attribute is a property of an entity, such as a class field, table column, etc.;
- constraints of an entity attribute are properties of an entity attribute, for example, the type of a class field, the size of a class field, etc.;
- business processes are processes of a specific subject area, in which some entity is involved, for example, a class method, a use case, etc.

The ontology of design artifacts is:

$$O^{exp} = \langle E^{exp}, A^{exp}, C^{exp}, P^{exp}, R^{exp}, F^{exp} \rangle, \tag{3}$$

where E^{exp} is a set of entities;
A^{exp} is a set of entities attributes;
C^{exp} is a set of constraints on entities attributes;
P^{exp} is a set of business processes which entities are involved;
F^{exp} is a set of interpretation functions that define a set of relationships R^{exp};
R^{exp} is a set of relationships:

$$R^{exp} = \{R^{exp}_{EE}, R^{exp}_{EA}, R^{exp}_{AC}, R^{exp}_{EP}\},$$

where R^{exp}_{EE} is a set of generalization relationships defining a hierarchy of entities;
R^{exp}_{EA} is a set of association relationships defining the relationship between the entity E^{exp}_i and its attribute A^{exp}_j;
R^{exp}_{AC} is a set of association relationships defining the relationship between the entity attribute A^{exp}_i and its constraint C^{exp}_j;
R^{exp}_{EP} is a set of association relationships defining the relationship between the entity E^{exp}_i and business process P^{exp}_j.

2.4 Knowledge Base Model

The previously considered ontologies form the KB of a software company.

KB model is:

$$O = \langle O^{dom}, O^{exp}, R, F \rangle,$$

where O^{dom} is the ontology for describing various subject areas, which used on previous projects (Eq. 1);
O^{exp} is the ontology for representing design artifacts (Eq. 3). This ontology is formed in the process of analyzing design artifacts;
R is a set of association relations between concepts of domain ontology and design artifacts;
F is a set of interpretation functions that define a set of relationships R.

3 Using Knowledge Base to Architecting of Software Systems

3.1 Formation of KB Fragments in Design Artifacts Analyzing

The 1C: Enterprise 8 platform [25] application (1C application) is used as an example of a design artifact in this paper.

The 1C platform is widely used in CIS countries to automate various types of accounting. 1C applications are developed for each type of accounting.

Thus, the 1C applications are the design artifacts that contain the experience and knowledge of a large number of highly qualified specialists.

A 1C application can be represented as the following expression:

$$App = \{S_1^{Conf}, S_2^{Conf}, \ldots, S_n^{Conf}\},$$

where S_i^{Conf} is an i-th subsystem of a 1C application. The subsystem is used to group application objects within a certain context: “Orders”, “Tools”, “Employees”, etc.

Subsystem S_i^{Conf} can be represented as the following expression:

$$S_i^{Conf} = \langle Name, E^{Conf}, P^{Conf} \rangle, \tag{4}$$

where $Name$ is unique subsystem name;
$E^{Conf} = \langle C^{Conf}, N^{Conf} \rangle$ is a set of application entities. A set of catalogs C^{Conf} and enumerations N^{Conf} are used to represent the entities of the problem area in a 1C application;
$P^{Conf} = \langle D^{Conf}, R^{Conf} \rangle$ is set of application business processes, in which the entities are involved. Set of business processes P^{Conf} contains set of documents D^{Conf} and set of registers R^{Conf}. Documents are used to change the state of the application entities. Registers are used to store the history of entity states. Register entries are formed based on documents data.

Each element of the sets C^{Conf} and D^{Conf} can be represented as the following expression:

$$C_i^{Conf}, D_j^{Conf} = \langle Name, V^{Conf}, T^{Conf} \rangle,$$

where $Name$ is unique element name;
$V^{Conf} = \{V_1^{Conf}, V_2^{Conf}, \ldots, V_m^{Conf}\}$ is a set of element attributes;
$V_k^{Conf} = \langle Name, Type, Constraint \rangle$ is an attribute of an element. $Name$ is attribute name, $Type$ is attribute data type and $Constraint$ is attribute constraints. The attribute data type $Type$ can contain a simple data type and a link to another application object (entity or business-process);
$T^{Conf} = \{T_1^{Conf}, T_2^{Conf}, \ldots, T_m^{Conf}\}$ is a set of application object tables. Each application object can have zero, one or more tables. Each table contains a unique name and set of attributes $T_l^{Conf} = \langle Name, V^{Conf} \rangle$.

Element of the set R^{Conf} represents as:

$$R_i^{Conf} = \langle Name, V^{Conf} \rangle.$$

Element of the set N^{Conf} represents as:

$$N_i^{Conf} = \langle Name \rangle.$$

Table 1 contains correspondence between objects of an i-th subsystem of a 1C application and entities of the ontology of design artifacts O^{exp}.

Table 1. Correspondence between objects of an i-th subsystem of a 1C application and entities of the ontology of design artifacts.

1C application	Ontology O^{exp}
$S_i^{Conf}.Name$	E_i^{exp}
$C_i^{Conf}.Name$	$E_i^{exp}, R_{EE_i}^{exp}$
$N_i^{Conf}.Name$	$E_i^{exp}, R_{EE_i}^{exp}$
$T_i^{Conf}.Name$	$E_i^{exp}, R_{EE_i}^{exp}$
$D_i^{Conf}.Name$	$P_i^{exp}, R_{EE_i}^{exp}$
$R_i^{Conf}.Name$	$P_i^{exp}, R_{EE_i}^{exp}$
$V_i^{Conf}.Name$	$A_i^{exp}, R_{EA_i}^{exp}$
$V_i^{Conf}.Type, V_i^{Conf} \in C_j^{Conf}$	$(C_i^{exp}, R_{EC_i}^{exp})$
$V_i^{Conf}.Type, V_i^{Conf} \in D_j^{Conf}$	$R_{EP_i}^{exp}$
$V_i^{Conf}.Constraint$	$C_i^{exp}, R_{AC_i}^{exp}$

Consider the algorithm for the formation of KB fragments using the example of the analysis of the 1C application of the following structure:

- subsystems: 'Trade'.
- catalogs: 'Products', 'Units', 'Warehouses', 'Suppliers'.
- 'Products' catalog attributes: 'Id' (string), 'Name' (string), 'Unit' (link to the element of 'Units' catalog).
- 'Units' catalog attributes: 'Id' (string), 'Name' (string).

- 'Warehouses' catalog attributes: 'Id' (string), 'Name' (string).
- 'Suppliers' catalog attributes: 'Id' (string), 'Name' (string).
- documents: 'Products supply'.
- 'Products supply' document attributes: 'Warehouse' (link to the element of 'Warehouses' catalog), 'Supplier' (link to the element of 'Suppliers' catalog), 'Products' (table).
- 'Products' table attributes: 'Products' (link to the element of 'Product' catalog), 'Price' (numeric), 'Quantity' (numeric), 'Amount' (numeric).

The algorithm for the formation of KB fragments based on the analysis of a 1C application can be presented as following steps:

1. Form the entity of the ontology of design artifacts for each subsystem of a 1C application:
$$S_i^{Conf} \rightarrow E_i^{exp},$$
for example:
$$\text{Trade} \in E^{exp}.$$
2. Create an entity of the ontology of design artifacts for each entity of the current subsystem, taking into account the relations between them:
$$C^{Conf} \rightarrow \hat{E}^{exp} \in E^{exp}, \hat{R}_{EE}^{exp} \in R_{EE}^{exp},$$
$$N^{Conf} \rightarrow \hat{E}^{exp} \in E^{exp}, \hat{R}_{EE}^{exp} \in R_{EE}^{exp},$$
for example:
$$\text{Products} \in E^{exp}, \text{Units} \in E^{exp}, \text{Warehouses} \in E^{exp}, \text{Suppliers} \in E^{exp},$$
$$\text{Products } R_{EE}^{exp} \text{ Trade}, \ldots, \text{Suppliers } R_{EE}^{exp} \text{ Trade}.$$
3. If the current entity of a 1C application is a catalog, then create a set of attributes, attribute constraints, and relationships for this entity of ontology:
$$V^{Conf} \in C_i^{Conf} \rightarrow \hat{A}^{exp} \in A^{exp}, \hat{R}_{EA}^{exp} \in R_{EA}^{exp},$$
$$V^{Conf} \in C_i^{Conf} \rightarrow \hat{C}^{exp} \in C^{exp}, \hat{R}_{EC}^{exp} \in R_{EC}^{exp},$$
for example:
$$\text{Id} \in A^{exp}, \text{Name} \in A^{exp}, \text{Unit} \in A^{exp},$$
$$\text{Products } R_{EA}^{exp} \text{ Id}, \text{Products } R_{EA}^{exp} \text{ Name}, \text{Products } R_{EA}^{exp} \text{ Unit}, \ldots,$$
$$\text{Suppliers } R_{EA}^{exp} \text{ Id}, \text{Suppliers } R_{EA}^{exp} \text{Name},$$
$$\text{string} \in C^{exp}, \text{numeric} \in C^{exp}, \text{link} \in C^{exp},$$
$$\text{Id } R_{AC}^{exp} \text{ string}, \text{Name } R_{AC}^{exp} \text{ string}, \text{Unit } R_{AC}^{exp} \text{ link}.$$

4. If the current entity of a 1C application has tables (is catalog), then, form a set of attributes, constraints of attributes, and relations for this entity of ontology for each table:

$$T^{Conf} \in C_i^{Conf} \rightarrow \hat{E}^{exp} \in E^{exp}, \hat{R}_{EE}^{exp} \in R_{EE}^{exp},$$

$$V^{Conf} \in T_j^{Conf}, T_j^{Conf} \in C_i^{Conf} \rightarrow \hat{A}^{exp} \in A^{exp}, \hat{R}_{EA}^{exp} \in R_{EA}^{exp},$$

$$V^{Conf} \in T_j^{Conf}, T_j^{Conf} \in C_i^{Conf} \rightarrow \hat{C}^{exp} \in C^{exp}, \hat{R}_{EC}^{exp} \in R_{EC}^{exp}.$$

5. Create a business process of ontology for each business process of the current subsystem, taking into account the connection between them:

$$D^{Conf} \rightarrow \hat{P}^{exp} \in P^{exp}, \hat{R}_{EP}^{exp} \in R_{EP}^{exp},$$

$$R^{Conf} \rightarrow \hat{P}^{exp} \in E^{exp}, \hat{R}_{EP}^{exp} \in R_{EP}^{exp},$$

for example:

$$\text{Products supply} \in P^{exp}, \text{Products supply } R_{EE}^{exp} \text{ Trade}.$$

6. Create a set of relationships for the current business process and the entities of the ontology based on the analysis of the attributes of the current business process of a 1C application:

$$V^{Conf} \in D_i^{Conf} \rightarrow \hat{R}_{EP}^{exp} \in R_{EP}^{exp},$$

$$V^{Conf} \in R_i^{Conf} \rightarrow \hat{R}_{EP}^{exp} \in R_{EP}^{exp},$$

for example:

$$\text{Suppliers } R_{EP}^{exp} \text{ Products supply}, \text{Warehouses } R_{EP}^{exp} \text{ Products supply}.$$

7. If the current business process of a 1C application has tables (is a document), then form a set of relationships to connect the current business process and entities of ontology for each table:

$$V^{Conf} \in T_j^{Conf}, T_j^{Conf} \in D_i^{Conf} \rightarrow \hat{R}_{EP}^{exp} \in R_{EP}^{exp},$$

for example:

$$\text{Products } R_{EP}^{exp} \text{ Products supply}.$$

Also, the following domain ontology is formed for the above 1C application:

$$\text{Catalog} \in C^{dom}, \text{Edit information about} \in T^{dom},$$

$$\text{Catalog } R_{CT}^{dom} \text{ Edit information about},$$

$$\text{Document} \in C^{dom}, \text{Create document for} \in T^{dom},$$

$$\text{Document } R_{CT}^{dom} \text{ Create document for},$$

3.2 Generating Use Case Diagrams Based on the Knowledge Base Content

A use case diagram in UML notation can be represented as the following expression:

$$UML = \langle A^{UML}, P^{UML}, R^{UML} \rangle, \tag{5}$$

where A^{UML} is a set of actors;
P^{UML} is a set of use cases;
R^{UML} is a set of relationships:

$$R^{UML} = \{R_C^{UML}, R_I^{UML}, R_E^{UML}\},$$

where R_C^{UML} is a set of generalization relationships;
R_I^{UML} is a set of inclusion relationships;
R_E^{UML} is a set of expansion relationships.

Thus, to generate of use case diagram in UML notation based on the content of the KB is need to transform the axioms of ontology O^{exp} (Eq. 3) into use case diagram UML elements (Eq. 5).

Table 2 contains correspondence between entities of ontology O^{exp} and the use case diagram structural components.

Table 2. Correspondence between entities of the ontology of design artifacts and the use case diagram structural components.

Ontology	Use-case diagram
Business process P_i^{exp}	Use case P_i^{UML}
Business processes $\hat{H}^P$	Inclusion relationship R_I^{UML}

As you can see from Table 2, the expansion and generalization relations are not formed. Additional work is required to improve the methods for generating and analyzing the content of the ontology O^{exp} in use UML case diagram creation.

The algorithm for generating the use cases diagram in UML notation based on the content of the KB consists of the following steps:

1. Selecting business processes of ontology as the use case diagram basis. For example, Products supply $\in P^{exp}$.
2. Analysis of the relationships R_{EP}^{exp} between selected business processes $\hat{P}^{exp}$ and other concepts $\hat{E}^{exp}$ of ontology to create a list of entities L^E for use case diagram generation. For example, Products $\in E^{exp}$, Units $\in E^{exp}$, Warehouses $\in E^{exp}$, Suppliers $\in E^{exp}$.
3. Formation of a hierarchy of entities H^E from the list of entities L^E based on analysis of relationships R_{EE}^{exp} of ontology. For example, Products R_{EE}^{exp} Trade, ..., Suppliers R_{EE}^{exp} Trade.

4. Replace the entities of the resulting hierarchy H^E with related business processes H^P if such a business process is not present in the diagram.
5. Form a use case P_i^{UML} for each node. Create inclusion relations R_I^{UML} between use cases.
6. Create an actor and link it to the root use case.

The set of commands is generated for the PlantUML system [20] after executing this algorithm. For example:

@startuml
:user: — (Trade)
(Trade)..>(Products supply):include
(Products supply)..>(Edit information about Suppliers):include
(Products supply)..>(Edit information about Warehouses):include
(Products supply)..>(Edit information about Products):include
(Products supply)..>(Create document for Products supply):include
@enduml

Also, the domain ontology considered above is used in the process of generating a use case diagram.

The resulting use-case diagram is shown in Fig. 1.

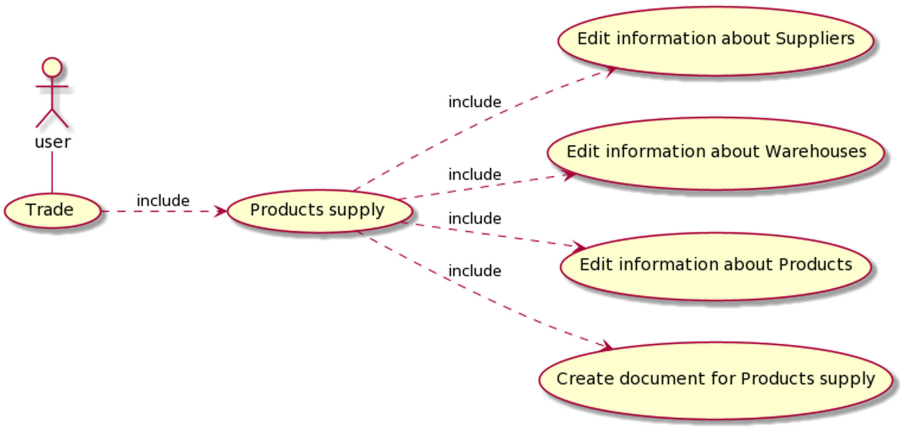

Fig. 1. The example of a generated use case diagram

4 Experiments Results

4.1 Plan of Experiments

A series of experiments were carried out to evaluate the quality of the proposed in this paper methods and models.

The experiments plan includes:

1. Formation of the KB in the form of OWL ontology in the process of analyzing a 1C application.
2. Validation of the obtained OWL ontology for correctness using the reasoner in Protege [27].
3. Generation of a set of PlantUML system command to generate use case diagrams in UML notation.
4. Evaluating the quality of generated design artifacts by comparing them with similar design artifacts created by an expert.

The following metrics were used to evaluate the quality of generated design artifacts:

1. Precision:
$$Precision = \frac{|Gen \cap Exp|}{|Exp|},$$
where $|Gen \cap Exp|$ is the number of matching elements in the generated design artifact and the design artifact created by an expert;
$|Exp|$ is the number of elements in the design artifact created by an expert.
2. Recall:
$$Recall = \frac{|Gen - Exp|}{|Gen|},$$
where $|Gen - Exp|$ is the number of elements in the generated design artifact that are missing in expert design artifact;
$|Gen|$ is the number of elements in the generated design artifact.

Precision and recall metrics used in evaluating the quality of design artifacts generated using the proposed approach are not similar to quality metrics of information retrieval. Precision and recall are not mutually exclusive metrics.

4.2 Evaluation of the Quality of the Generated Use Case Diagram in UML Notation

The "Trade Management 11.2" 1C application was used in this experiment as the design artifact to form the KB content.

The following objects from subsystem "Discrepancies" of application for the "Trade Management 11.2" 1C application were selected:

- subsystems "Purchase" (parent subsystem) and "Discrepancies";
- accumulation register "ShipmentSchedules";
- catalogs "Organizations", "Currencies", "Partners", "Counteragents", "Users", "Nomenclature", "Appointments", "Warehouses";
- Documents "DivergenceActAfterProductsMovement", "PostAcceptanceDiscrepancyAct", "PostDispatchAct".

Verification of the generated OWL ontology in the Protege editor was successful.

Also, the domain ontology O^{dom} (Eq. 1 and 2) was created. The ontology O^{dom} contains alternative names for objects of the "Trade Management 11.2" 1C application used as names of use cases. For example, for the concept "Purchase" set alternative name "Purchase Products", for the concept "Warehouses" set alternative name "Get info about warehouses", for the concept "DivergenceActAfterProductsMovement" – "Get divergence act after products movement".

The generated use case diagram (Fig. 2) contains 14 use cases. Nine use cases are not contained in the diagram created by the expert. The number of use cases in the diagram created by the expert (Fig. 3) is 5. Thus, values of quality metrics are: $Precision = 5/5 = 1$, and $Recall = 9/14 = 0.643$.

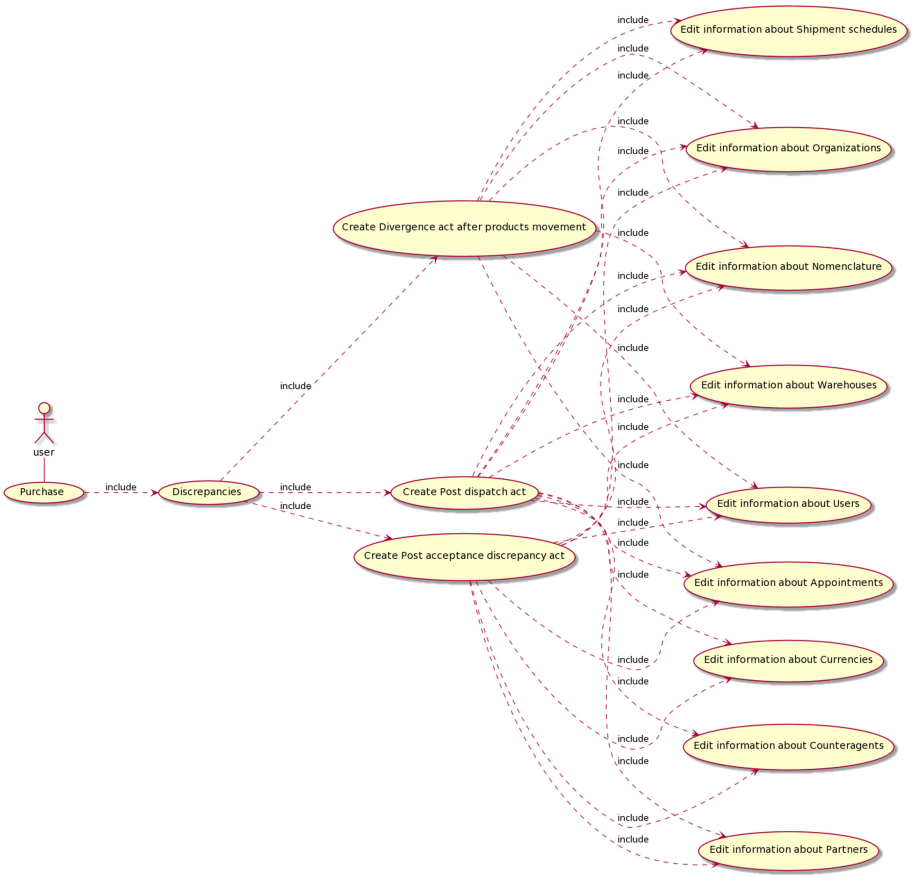

Fig. 2. The generated use case diagram

The time spent on the automatic generation of the use cases diagram was 3 min (without domain ontology O^{dom} formation time). An expert spent 17 min on a similar task.

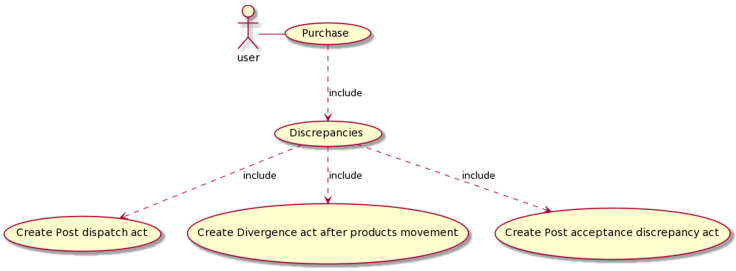

Fig. 3. The use case diagram built by the expert

Table 3 contains the results of experiments on the formation of use case diagrams in UML notation based on the KB content. KB was formed based on the analysis of various 1C applications.

Table 3. Experimental results.

1C application	Subsystem	The number of use cases			Precision	Recall
		Generated	Created by expert	Matching elements		
Trade management 11.2	Discrepancies	14	5	9	1	0.643
Salary and HR management 3.1	Personnel	8	4	4	1	0.5
Complex automation 2	Warehouse	12	6	4	0.667	0.667
Trade management 11.2	Bank and cash desk	13	6	5	0.833	0.615

The experimental results show that the generated design artifacts have a high precision rate and contain more structural elements, which increases the rate of recall. The high value of recall is related to a set of connections between the objects of the analyzed 1C application obtained during their life cycle.

5 Conclusion

This paper presents an approach to automating the architectural design process of a SS based on the KB of a software company. The KB contains ontologies that store knowledge about points of the subject area and successful design and architectural solutions extracted from previous projects. The new model of the KB of a software company is presented.

An application for the 1C platform is used in this paper as an example of a design artifact.

The paper also presents the experiments to evaluate the adequacy of the proposed models and methods by comparing the precision and recall values indicators for generated design artifacts and design artifacts created by the expert.

Acknowledgments. The authors acknowledge that the work was supported by the framework of the state task of the Ministry of Science and Higher Education of the Russian Federation No. 075-00233-20-05 "Research of intelligent predictive multimodal analysis of big data, and the extraction of knowledge from different sources".

The reported study was funded by RFBR and Ulyanovsk region, project numbers 19-47-730005 and 19-47-730006.

References

1. ISO/IEC/IEEE 42010:2011 Systems and Software Engineering - Architecture Description. http://docs.cntd.ru/document/1200139542. Accessed 22 Apr 2021
2. Van Heesch, U., Avgeriou, P., Hilliard, R.: A documentation framework for architecture decisions. J. Syst. Softw. **85**(4), 795–820 (2012). https://doi.org/10.1016/j.jss.2011.10.017
3. Hilliard, R.: Using aspects in architectural description. In: Moreira, A., Grundy, J. (eds.) EAW 2007. LNCS, vol. 4765, pp. 139–154. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76811-1_8
4. Sosnin, P.: Substantially evolutionary theorizing in designing software-intensive systems. Information **9**(4), 91 (2018). https://doi.org/10.3390/info9040091
5. Sosnin, P., Pushkareva, A.: Ontological controlling the lexical items in conceptual solution of project tasks. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10409, pp. 31–46. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62407-5_3
6. McConnell, S.: Code Complete. Pearson Education (2004)
7. Novák, V., Perfilieva, I., Jarushkina, N.G.: A general methodology for managerial decision making using intelligent techniques. In: Rakus-Andersson, E., Yager, R.R., Ichalkaranje, N., Jain, L.C. (eds.) Recent Advances in Decision Making. Studies in Computational Intelligence, vol. 222. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02187-9_7
8. Yarushkina, N., Moshkin, V., Andreev, I., Klein, V., Beksaeva, E.: Hybridization of fuzzy inference and self-learning fuzzy ontology-based semantic data analysis. In: Abraham, A., Kovalev, S., Tarassov, V., Snášel, V. (eds.) Proceedings of the 1st International Scientific Conference on Intelligent Information Technologies for Industry (IITI'16). AISC, vol. 450, pp. 277–285. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33609-1_25
9. Yarushkina, N., Filippov, A., Moshkin, V.: Development of the unified technological platform for constructing the domain knowledge base through the context analysis. In: Kravets, A., Shcherbakov, M., Kultsova, M., Groumpos, P. (eds.) CIT&DS 2017. CCIS, vol. 754, pp. 62–72. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65551-2_5
10. Sosnin, P.: A way for creating and using a theory of a project in designing of a software intensive system. In: 17th International Conference on Computational Science and Its Applications (ICCSA). IEEE (2017). https://doi.org/10.1109/iccsa.2017.7999646
11. Sosnin, P.: Conceptual experiments in automated designing. In: Projective Processes and Neuroscience in Art and Design, pp. 155–181. IGI Global (2017). https://doi.org/10.4018/978-1-5225-0510-5.ch010
12. Henninger, S.: Tool support for experience-based software development methodologies. In: Advances in Computers, pp. 29–82. Elsevier (2003). https://doi.org/10.1016/s0065-2458(03)59002-7

13. Abioye, T.E., Arogundade, O.T., Misra, S., Akinwale, A.T., Adeniran, O.J.: Toward ontology-based risk management framework for software projects: an empirical study. J. Softw. Evol. Process **32**(12), e2269 (2020)
14. Júnior, A.A.C., Misra, S., Soares, M.S.: A systematic mapping study on software architectures description based on ISO/IEC/IEEE 42010:2011. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11623, pp. 17–30. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24308-1_2
15. Oluwamayowa, A., Adedeji, A., Sanjay, M., Faith, A.: Empirical framework for tackling recurring project management challenges using knowledge management mechanisms. In: Gervasi, O., et al. (eds.) ICCSA 2020. LNCS, vol. 12254, pp. 954–967. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58817-5_67
16. Shaaban, A.M., Gruber, T., Schmittner, C.: Ontology-based security tool for critical cyber-physical systems. In: Proceedings of the 23rd International Systems and Software Product Line Conference 2019, pp. 207–210. ACM Press (2019). https://doi.org/10.1145/3307630.3342397
17. Bhatia, M.P.S., Kumar, A., Beniwal, R.: Ontologies for software engineering: past, present and future. Indian J. Sci. Technol. **9**(9), 1–16 (2016). https://doi.org/10.17485/ijst/2016/v9i9/71384
18. Falbo, R.A., et al.: An ontology pattern language for service modeling. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing 2016, pp. 321–326. ACM Press (2016). https://doi.org/10.1145/2851613.2851840
19. Ilyas, Q.M.: Ontology augmented software engineering. In: Software Development Techniques for Constructive Information Systems Design, pp. 406–413. IGI Global (2013). https://doi.org/10.4018/978-1-4666-3679-8.ch023
20. Isotani, S., Ibert Bittencourt, I., Francine Barbosa, E., Dermeval, D., Oscar Araujo Paiva, R.: Ontology driven software engineering: a review of challenges and opportunities. IEEE Lat. Am. Trans. **13**(3), 863–869 (2015). https://doi.org/10.1109/tla.2015.7069116
21. Pan, J.Z., Staab, S., Aßmann, U., Ebert, J., Zhao, Y.: Ontology-Driven Software Development. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-31226-7
22. Godfrey, M.W., Zou, L.: Using origin analysis to detect merging and splitting of source code entities. IEEE Trans. Softw. Eng. **31**(2), 166–181 (2005)
23. Ali, N., Sharafi, Z., Guéhéneuc, Y.-G., Antoniol, G.: An empirical study on the importance of source code entities for requirements traceability. Empir. Softw. Eng. **20**(2), 442–478 (2014). https://doi.org/10.1007/s10664-014-9315-y
24. Savić, M., Rakić, G., Budimac, Z., Ivanović, M.: A language-independent approach to the extraction of dependencies between source code entities. Inf. Softw. Technol. **56**(10), 1268–1288 (2014)
25. What is 1C:Enterprise? Source, https://1c-dn.com/1c_enterprise/what_is_1c_enterprise/. Accessed 22 Apr 2021
26. PlantUML. UML Diagram Generator. https://plantuml.com. Accessed 22 Apr 2021
27. Protege. Free, open-source ontology editor and framework for building intelligent systems. https://protege.stanford.edu. Accessed 22 Apr 2021