Federal University of Campina Grande

Department of Systems and Computation

Postgraduate Coordination in Computer Science

Master thesis

Recommender Systems for UML Class Diagrams

Saulo Soares de Toledo

Campina Grande, Paraíba, Brazil

September 5, 2016

Federal University of Campina Grande

Department of Systems and Computation

Postgraduate Coordination in Computer Science

Recommender Systems for UML Class Diagrams

Saulo Soares de Toledo

Master's thesis submitted to the Postgraduate Coordination in Computer Science of the Federal University of Campina Grande - Campus I in partial fulfillment of the requirements for the Master's degree in Computer Science.

Main area: Computer Science

Research line: Recommender Systems for Software Engineering

Franklin de Souza Ramalho

Leandro Balby Marinho

(Advisors)

Campina Grande, Paraíba, Brazil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

T649r Toledo, Saulo Soares de.

Recommender systems for UML class diagrams / Saulo Soares de Toledo. – Campina Grande, 2016.

128 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2016.

"Orientação: Prof. Dr. Franklin de Souza Ramalho, Prof. Dr. Leandro Balby Marinho”.

Referências.

1. Engenharia de Software. 2. Diagrama de Classe. 3. Sistemas de Recomendação. 4. OntoRec. I. Ramalho, Franklin de Souza. II. Marinho, Leandro Balby. III. Título.

CDU 004.41(043)

Resumo

Modelos UML são usados de várias formas na engenharia de software. Eles podem modelar desde requisitos até todo o software, e compreendem vários diagramas. O diagrama de classes, o mais popular dentre os diagramas da UML, faz uso de vários elementos UML e adornos, tais como abstração, interfaces, atributos derivados, conjuntos de generalização, composições e agregações. Atualmente, não há maneira fácil de encontrar este tipo de diagrama com base nestas características para a reutilização ou a aprendizagem por tarefas de exemplo. Por outro lado, Sistemas de Recomendação são ferramentas e técnicas que são capazes de descobrir os elementos mais adequados para um usuário, dentre muitos outros. Existem várias técnicas de recomendação, que usam informações dos elementos de várias maneiras, ao uso da opinião de outros usuários. Sistemas de recomendação já foram utilizadas com sucesso em vários problemas da engenharia de software, a exemplo da recomendação de partes de código para reuso (como métodos, por exemplo) e da identificação do desenvolvedor mais adequado para trabalhar em certas áreas do software. Este trabalho tem como objetivo propor e avaliar (i) uma representação baseada em conteúdo para diagramas de classe e as preferências do usuário, (ii) um novo algoritmo de recomendação baseado no conhecimento, (iii) a aplicação deste algoritmo e outros dois outros do estadodaarte para a recomendação de diagramas de classe UML e (iv) uma avaliação destas abordagens contra uma sugestão aleatória. Para atingir este objetivo, foi realizado um estudo de caso com estudantes de ciência da computação e egressos. Depois de comparar os algoritmos, os nossos resultados mostram que, para o nosso conjunto de dados, todos eles são melhores do que uma recomendação aleatória.

Abstract

UML models are used in several ways in the software engineering. They can model from requirements to the entire software, and comprise several diagrams. The Class diagram, the most popular among the UML diagrams, makes use of several UML elements and adornments, such as abstraction, interfaces, derived attributes, generalization sets, compositions and aggregations. Currently, there is no easy way to find this kind of diagram based on these features for reuse or learning by example's tasks, for instance. On the other hand, Recommender Systems are powerful tools and techniques that are able to discover the most appropriate elements to an user among many others. There are several recommender techniques, from using the elements' information in several ways, to using other users' opinions. Recommender systems were already used successfully in several software engineering problems, as discovering pieces of code to recommend (as methods, for example) and finding the best developer to work in certain software problems. This work aims to propose and evaluate (i) a content-based Recommender System's representation for class diagrams' features and user's preferences, (ii) a new knowledge-based recommender algorithm, (iii) the application this algorithm and two other state of the art content-based ones to the recommendation of UML class diagrams and (iv) an evaluation of these approaches against a random suggestion. To achieve this goal, we conducted a case study with computer science students and egresses. After comparing the algorithms, our results show that, for our dataset, all of them are better than a random recommendation.

Acknowledgement

First and foremost, thanks to God Almighty, whose many blessings have made me what I am today and allowed me to complete this work.

I am forever grateful to my advisors Franklin Ramalho and Leandro Marinho for having believed on me and in my job. I will never forget their commitment with my work. Franklin first believed on me while proposing to work with Leandro, that also accepted to be my advisor. I have learned a lot about how to be a researcher from them.

I am also deeply grateful to the Professor Francisco Neto for his time and guidance with this work's experiments. I also learned a lot with him.

I would like to thank Thaciana Cerqueira for our several hours of discussion about recommender systems and UML, and her contributions to this work. I also would like to thank the Professor Matheus Gaudêncio who made great contributions especially in the first months of my Master's degree.

I also thank my girlfriend Emanuele Montenegro. Despite not having directly participated in this work, she helped me throughout the rest I needed to be able to finish this master thesis, whether solving other tasks for me, whether encouraging me and being present all the time. I could not be here without her.

I thank my parents, Zélia Toledo and Severino Toledo, for their patience and support, they did more for me than I can pay. And I thank my sister, Zeneide Toledo, and my brothers, Sérgio Toledo, Sidney Toledo, Sílvio Toledo and Simões Toledo, for their support.

I thank the students Alberto Medeiros and Gustavo Alves. They have participated as research students in this master thesis. They have made some contributions, like starting building a new version of the tool used in our experiments, or downloading several UML class diagrams from an UML repository.

I thank my friends Adson Diego, Alysson Filgueira, Antonio Ricardo, Carol Cabral, Catharine Quintans, Emanuel Batista, Everton Galdino, Fernando Ferreira, Guilherme Neves, Katyusko Farias, Luiz Antonio, Renata Paiva and Vladwoguer Bezerra that, directly or indirectly, contributed to my research. I also thank all the participants of the experiments of this work.

To my parents Zélia Toledo and Severino Toledo.

Contents

List of Symbols

IDF - *Inverse Document Frequency*

IR - *Information Retrieval*

MOF - *MetaObject Facility*

OMG - *Object Management Group*

OWL - *Ontology Web Language*

PCOA - *Presence of Composite Aggregation*

PABC - *Presence of Abstract Class*

PADV - *Presence of Attribute with Default Value*

PAO - *Presence of Abstract Operation*

PASC - *Presence of Association Class*

PDA - *Presence of Derived Attribute*

PEN - *Presence of Enumeration*

PGS - *Presence of Generalization Set*

PNAA - *Presence of Navigation Arrows in Associations*

POD - *Presence of Dependency*

POG - *Presence of Generalization*

POI - *Presence of Interface*

POP - *Presence of Port*

POQ - *Presence of Qualifier*

PRI - *Presence of Realized Interface*

PSA - *Presence of Static Attribute*

PSHA - *Presence of Shared Aggregation*

PSO - *Presence of Static Operation*

PTC - *Presence of Template Class*

RDF -	*Resource Description Framework*
RS -	*Recommender System*
TF -	*Term Frequency*
UML -	*Unified Modeling Language*
XMI -	*XML Metadata Interchange*
XML -	*eXtensible Markup Language*

List of Figures

List of Tables

Source Code List

Chapter 1

Introduction

The UML (Unified Modeling Language) [39] is a family of graphical notations (diagrams), described by a single metamodel (a model that represents another model), for purposes of description and project of software systems (mainly that ones developed by using the Object Oriented software development paradigm [61]). Standardized by OMG (Object Management Group)[1], it currently defines 13 types of diagrams, each one with its specific purpose.

UML is widely used by companies that can store from some to several hundreds of diagrams in its files. Commonly, the production of UML models increases with time, especially in companies who adopt methodologies like model-driven development (MDD) [50], where models are pivotal elements. Unfortunately, finding UML diagrams for reuse, searching or even learning with examples is not an easy task. Lucrédio et al. [29] agree that the current search engines lack features to find UML models. Even if there is a database of UML diagrams, there is the need to identify items that meet the user needs, and a way to do that is through Recommender Systems. More than performing searches in a database, these systems identify the user needs based on the user's profile before recommending items.

In the context of software engineering, it is normal that several artifacts begin to be explored in the context of recommendation. For instance, Robillard et al. [48] did a survey in this area and presented, among others: *Strathcona*, a tool that retrieves relevant source code examples to help developers to use frameworks in a more efficient way; *Dhruv*, that recommends people and artifacts relevant to error reports; and *Expertise Browser*, that recommends experts (people) detecting past changes for a given piece of code or document.

[1]`http://www.omg.org/`

The context of source code has already been extensively studied, and the proposal to do the same with UML class diagrams sounds promising. According to Miles and Hamilton [34], this diagram is the most popular among the language. It is also fairly often pointed out in several studies as the most used among the UML diagrams [14] [15]. Class diagrams describe the types of the objects in the system and its several static relationships that exists between them. There are numerous study possibilities by linking Recommender Systems and UML, (i) in the industry, where there are companies with several hundred UML diagrams archived that could be useful, and (ii) in the university, where good examples could be useful to those students who are learning about the language.

This chapter introduces this work. Section 1.1 presents the problem that is studied. Section 1.2 presents the research objectives. Section 1.3 presents the research scope. Section 1.4 summarizes the research contributions. Section 1.5 briefly discuss about the relevance of this work. Finally, Section 1.6 presents the structure of this document.

1.1 Problem

Despite the increasing number of researches related to Recommender Systems for Software Engineering, it is believed that this is an area that can still be quite explored. Some of their subareas lack related studies, as UML for example. The closest proposal related to UML was the search engine for UML models proposed by Lucrédio et al. [29]. Their work apply information retrieval techniques in the UML metamodel to find relevant diagrams, and requires that the user provides a search string as an input in order to find them. Each UML diagram has its own context of use, some of them are used to model systems, others are best to describe requirements and so on. Thus, we decided that we should start by proposing a study for one of these diagrams, the class diagram, instead of proposing an approach that combine them all.

The main problem tackled in this work is how to recommend UML class diagrams. Thus, we need to discover what approaches could be applied in order to have good recommendations. Also, the conventional recommendation approaches were not originally designed for this problem, and we should investigate some ways of using them in our context.

1.2 Research Objectives

This research work aims to apply some classical state-of-the-art recommender system algorithms and propose new algorithms to UML class diagrams, comparing them to each other and against a random recommendation approach, in order to discover if they are suitable to recommend these diagrams. Our specific goals are:

- to investigate the suitability of the proposed content-based approaches (the bag-of-words, reused from information retrieval techniques [32], and the approach based in a features vector adapted from content-based recommender systems [45]) in the context of the recommendation of UML class diagrams;
- to investigate the suitability of the proposed knowledge-based approach, a completly new recommender system approach based in an ontology created to this purpose, in the context of the recommendation of UML class diagrams;
- to investigate if the proposed approaches are better than a simple random based recommendation;
- to investigate which of the proposed approaches have better results.

1.3 Research scope

In this work we propose and conduct a study that compare four ways of recommending UML class diagrams to users: (i) a random baseline, that randomly suggests items to the users, (ii) a bag-of-words algorithm, reused from information retrieval techniques [32], that identify UML diagrams by considering them as text files and performing word searching, (iii) a content-based algorithm adapted from content-based recommender systems [45] that uses an also proposed profile vector to identify relevant diagrams and (iv) a newly knowledge-based approach that identifies related items by discovering features related to that ones in the user profile.

By reusing, adapting or proposing the four presented approaches and comparing them, we want to identify ways of recommending UML class diagrams based on the user's interests, discovering if they are suitable to recommend these diagrams. An empirical study was performed in order to achieve this goal.

1.4 Contributions

In order to meet the objectives of this work, it proposes four algorithms, as previously presented. The random algorithm is a baseline for comparison. The bag-of-words algorithm is adapted from information retrieval techniques, and it is an adaptation of an old technique commonly used in textual search to the context of UML. It simple consider the UML diagrams files as text files, allowing it to perform searches in the UML diagram's database; this is completely possible because of the XMI file format, explained in Subsection 2.1.2. The other two approaches are based in the recommender systems theory: a content-based approach and a knowledge-based approach. The content-based technique is an adaptation of how several content-based algorithms work to the context of UML. The knowledge-based approach is a completely new generic recommender algorithm and has the Chapter 4 dedicated to it. The similarities and the differences between all the algorithms are presented in the Chapter 3.

The proposed recommender systems based approaches need a representation for the users and items. These representations are known as *user profile* and *item profile*. This work also proposes a representation for them in Section 3.1.

In order to evaluate the proposals, the algorithms were implemented together with a tool that allow subjects to execute the experiment (see Section 5.2 for details). It is also a contribution of this work an empirical study of the proposed algorithms in order to investigate the suitability of them in the context of UML.

1.5 Relevance of the proposal

The amount of information in software development is increasing. Today's software is more complex than some years ago, and the software technology is evolving every day. The amount of information related to it is overwhelming and the software engineering should also evolve to successfully keep pace and guide the software development. This complexity requires planning, abstraction and documentation methods, and UML is designed for those tasks, among others.

On the other hand, the recommendation systems play an important role for situations involving information overload, and the software engineering area has already received great contributions from them [48]. However, the solutions proposed until now are mostly dependent on the application's source code or very specific artifacts. UML does not contain the source code of the final application, and there are no solutions directed to the context of UML class diagrams related to recommendation.

We propose an investigation of recommender algorithms to the UML class diagram context with the purpose of reducing the lack of research in this area. The results of this study can also be used later for learning purpose researches as, for example, when a student searches for features that he/she want to learn, and the system can recommend the best diagrams. The extension of the current study by incrementing researches in educational psychology and cognitive science could be done in future in order to address this issue.

1.6 Dissertation Structure

The remaining of this document is organized as follows. Chapter 2 presents a brief summary of UML Class Diagrams, Recommender Systems and some information retrieval concepts related with this research. Chapter 3 presents the proposed recommender techniques. Chapter 4 formalizes the OntoRec algorithm, our knowledge-based recommender approach. Chapter 5 presents the experiment design of proposed work, its factors and treatments, subjects and objects, result analysis and some of its threats to the validity. Chapter 6 presents some approaches related to this research. Finally, Chapter 7 presents our conclusions and suggestions for further work.

Chapter 2

Background

Before presenting the work proposal, it is important to know some of the fundamental concepts from the related areas: UML class diagrams, ontologies, information retrieval and recommender systems. The following subsections present a brief resume of the background related to these terms in order to base the study. Section 2.1 presents the UML class diagrams, their main elements, a brief introduction to the UML metamodel and the concepts of MetaObject Facility (MOF) and XML Metadata Interchange (XMI). Section 2.2 introduces Information Retrieval (IR) and some related concepts, *bag-of-words* and *tf-idf*. Section 2.3 presents the concept of Recommender Systems. Finally, Section 2.4 presents what are ontologies, the semantic web, the Resource Description Framework (RDF) and the Ontology Web Language (OWL).

2.1 UML Class Diagrams

UML is a modeling language standardized by OMG (Object Management Group), a consortium of companies created to define standards that support software development and systems' interoperability [17, 34]. UML was born in 1997, from the unification of various graphical object-oriented modeling languages that appeared in the late '80s and early '90s [17]. It currently defines 13 types of diagrams, split into 2 categories, the *structure diagrams*, and the *behavior diagrams*.

The Class Diagram is the most popular among the UML diagrams and the most used of the structure diagrams [34]. It states the object types at a system and the static relationships that exist between them, besides showing the operations and properties of each class and other related features, such as *property strings* and cardinalities [17]. Figure 2.1 illustrates a high level class diagram by means of minor adaptations of the class diagram used in Warmer and Kleppe [60].

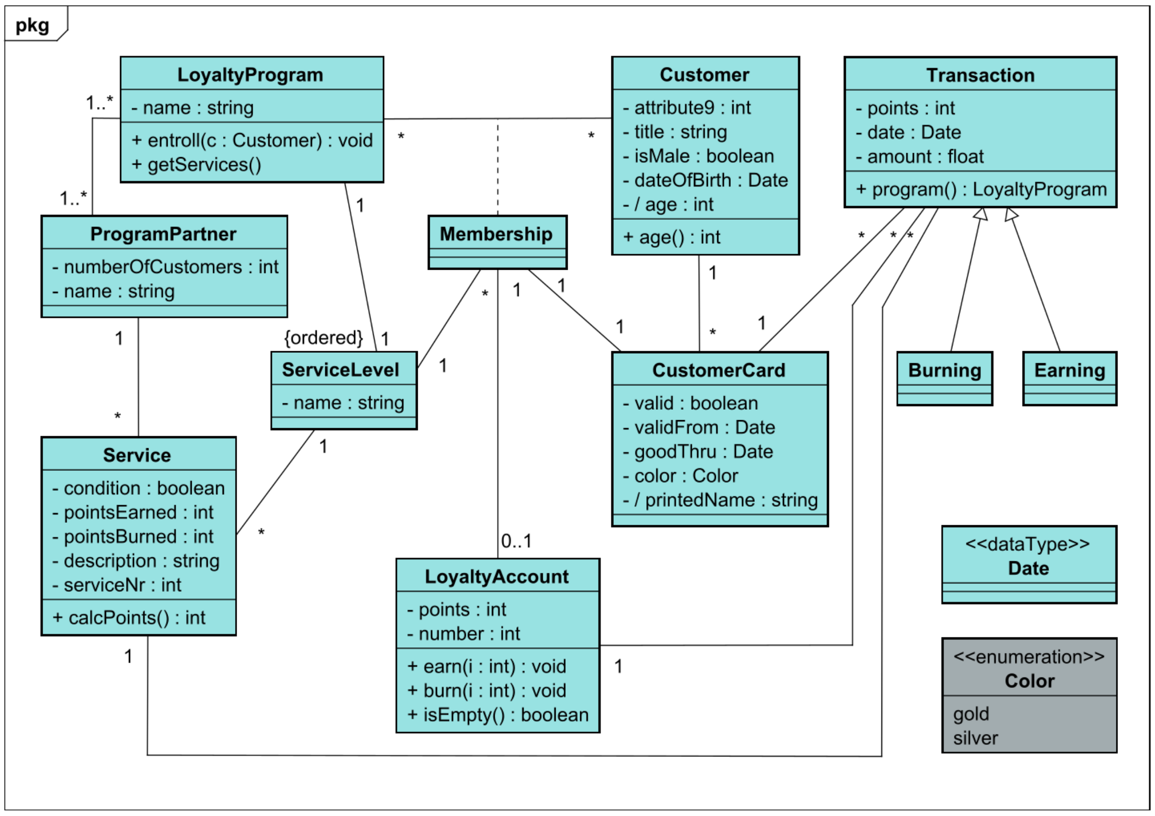


Figure 2.1: Example of UML Class Diagram adapted from Warmer and Kleppe [60]

Next are presented some of the most important elements that can be present in class diagrams:

- **Class**: The most basic element in this diagram, a class is a blueprint to build a specific type of object. There are several classes in Figure 2.1, like *Customer*, for example;
- **Association**: A solid line that represents a relationship between classes, like that one between *Service* and *ServiceLevel* in Figure 2.1. Associations can also be *aggregations* or *compositions*, indicating relationships of the type part-whole and continence;
- **Association Classes**: They are modeling elements that have both association and class

properties. In Figure 2.1 we have *Membership* as an association class, where *Customer* and *LoyaltyProgram* are related by a *Membership*;

- **Generalization**: According to OMG [40], it is “a taxonomic relationship between a more general classifier and a more specific classifier”. The instances of the more specific one are indirect instances of the more general one, and has its features. For example, the relation between the classes *Burning* and *Transaction* in Figure 2.1;

- **Enumerations**: They are elements who enumerates a fix set of literals. *Color*, in Figure 2.1, is an enumeration, whose literals are *gold* and *silver*;

- **Attributes** and **Operations**: Respectively, characteristics like *name* and *title* at *Customer*, and actions, like *enroll* at *LoyaltyProgram*, in Figure 2.1, for a class;

- **Derived attributes**: Attributes that are derived from others. For example, *age* is derived from *dateOfBirth* at *Customer* in Figure 2.1;

- **Property-strings**: Textual approaches for represent properties (for attributes and associations) that are named values denoting characteristics of elements and have semantic impact [27]. A common example is *{ordered}*, present at association between *LoyaltyProgram* and *ServiceLevel* in Figure 2.1;

- **Abstract Classes**: Classes that do not provide a complete declaration and thus cannot be directly instantiated. They are intended to be used by other classes [40];

- **Interfaces**: According to [40], an interface “represents a declaration of a set of coherent public features and obligations”. They specify a contract which every element that performs it should follow;

- **Dependencies**: They are relationships that indicate whether the change in the definition of a particular element may cause changes to the other;

- **Packages**: UML constructs that enable you to organize model elements into groups. Packages can contain classes, other packages, interfaces, enumerations and other UML diagrams’ elements [27]. An example is the package *RoyalAndLoyal* in Figure 2.1.

More details about UML Class diagrams can be found at OMG [40]. Also, in order to introduce how UML class diagrams are described and stored in computer files, the Subsection 2.1.1 will briefly introduce what is the UML metamodel and the Subsection 2.1.2 will introduce the concepts of MetaObject Facility (MOF) and XML Metadata Interchange (XMI). These concepts will be taken up later in this work.

2.1.1 UML Metamodel

The UML metamodel is an abstract grammar containing all the concepts (metaclasses) that can be used with UML and the relationships between them. Thus, every element of a model is an instance of a metaclass [60]. As an example, the UML Metamodel has a metaclass called *Class* from which every class in a UML model is instance of. In other words, the UML metamodel is a model that describe UML models. Figure 2.2, extracted from OMG [40], presents an excerpt of the UML metamodel as an example.

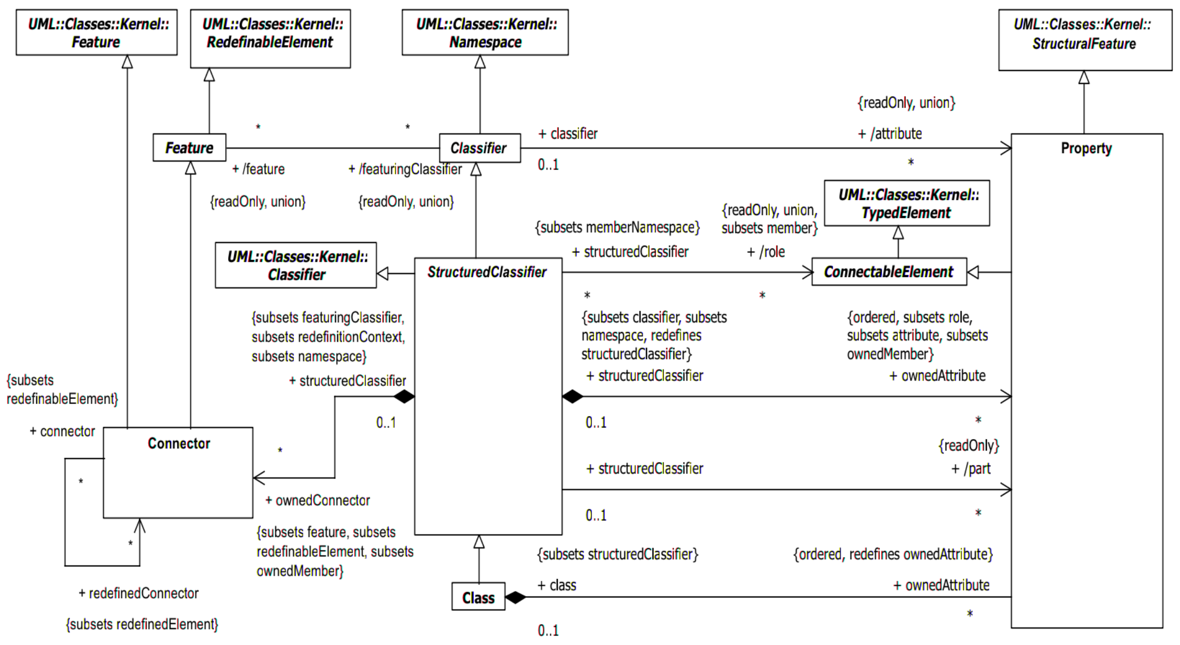

UML::Classes::Kernel::Feature
UML::Classes::Kernel::RedefinableElement
UML::Classes::Kernel::Namespace
UML::Classes::Kernel::StructuralFeature
Feature
Classifier
Property
{readOnly, union}
+ /attribute
+ classifier
0..1
*
+ /feature
+ /featuringClassifier
{readOnly, union}
{readOnly, union}
UML::Classes::Kernel::TypedElement
{subsets memberNamespace}
{readOnly, union, subsets member}
+ structuredClassifier
+ /role
UML::Classes::Kernel::Classifier
StructuredClassifier
ConnectableElement
{subsets featuringClassifier, subsets redefinitionContext, subsets namespace}
+ structuredClassifier
{subsets classifier, subsets namespace, redefines structuredClassifier}
+ structuredClassifier
{ordered, subsets role, subsets attribute, subsets ownedMember}
+ ownedAttribute
{subsets redefinableElement}
+ connector
Connector
0..1
+ ownedConnector
{subsets feature, subsets redefinableElement, subsets ownedMember}
{readOnly}
+ /part
+ structuredClassifier
{subsets structuredClassifier}
{ordered, redefines ownedAttribute}
+ class
+ ownedAttribute
Class
+ redefinedConnector
{subsets redefinedElement}

Figure 2.2: Part of the UML metamodel extracted from OMG [40]

One of the ways to describe the UML metamodel is by using the MetaObject Facility

(MOF)[1] specification [37], an OMG standard that defines the language to define modeling languages. MOF is defined using MOF itself, and it can describe the UML metamodel [24].

2.1.2 MetaObject Facility (MOF) and XML Metadata Interchange (XMI)

XML Metadata Interchange (XMI) [38] is an OMG standard for exchanging metadata information via Extensible Markup Language (XML). XMI is used to define, share, manipulate and integrate XML data and objects [38]. It is recommended by OMG itself to represent MOF [37] models. This file format is also used for integration between tools, applications and repositories, and is typically useful as interchange format for UML tools.

There are several vendor specific formats to represent UML models in a computer environment, but many UML modeling softwares (despite frequent interoperability problems between data generated by them) also support the XMI file format, an OMG standard for exchanging metadata information via Extensible Markup Language (XML).

XMI defines rules for defining schemes for any MetaObject Facility (MOF) model [38], a metadata management framework and a set of metadata services that enable the development and interoperability of a model and systems directed by metadata [37]. UML, whose metamodel can be described in MOF, can be represented in XMI [39].

The XMI Source Code 2.1 contains a snippet of the XMI representation of the diagram at Figure 2.1. Note the classes defined at lines 5 and 8, and the generalization (inheritance) line 9.

Source Code 2.1: XMI representation for the simple class diagram displayed

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI
    /2.1" xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML" xmi:id="
    _qb8akM37EdqwVrslYOdUDA">
3   <packagedElement xmi:type="uml:Package" xmi:id="_w8IxIM37EdqwVrslYOdUDA
      " name="RoyalAndLoyal">
4     ...
```

[1]http://www.omg.org/mof/

```
    <packagedElement xmi:type="uml:Class" xmi:id="_Sp6mINxcEeOTpaO0nxLrdA
       " name="Transaction">
      ...
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_UQPDsNxcEeOTpaO0nxLrdA
       " name="Burning">
      <generalization xmi:id="_l55-M9xeEeOTpaO0nxLrdA" general="
         _Sp6mINxcEeOTpaO0nxLrdA"/>
    </packagedElement>
    ...
  </packagedElement>
</uml:Model>
```

2.2 Information Retrieval

Information Retrieval (IR) is the activity of finding material of an unstructured or semi-structured nature, filtering document collections or further processing a set of retrieved documents, usually from large collections, to satisfy an information need. IR techniques are appropriate when users know what they are looking for, once they need to provide a query to be used by technique in order to retrieve some information. IR tries to locate relevant content [30]. Google [7] is an example of IR application; it searches for terms in a very large database of websites, using IR algorithms to recover relevant results quickly. The next subsections present some of the IR concepts used in this work: the Subsection 2.2.1 presents the bag-of-words model, and the Subsection 2.2.2 presents the *tf-idf* weighting scheme.

2.2.1 The Bag-of-Words model

The bag-of-words model is a representation used by many IR systems. Each document is described as a multiset of its own words [32]. This multiset, B, can be described as a set of pairs, word along with the number of ocurrences $B = \{(w_i, f(w_i)) | 1 \leq i \leq j\}$, where j is the amount of words of B, and f is a function that returns the number of occurrences for the word w_i at the current document.

As an example, if we get a document composed by the sentence “class diagrams are

UML diagrams“, we could represent B as

$$B = \{(class, 1), (diagrams, 2), (are, 1), (UML, 1)\}$$

This model does not store any kind of semantics, since there is only the number of occurrences for each word and the order it occurs does not matter for the model. To demonstrate this behavior, consider the sentence “UML diagrams are class diagrams”: we know that this is not true because UML is much more than just class diagrams, and this sentence has a different meaning from the previous one, but its representation as a bag-of-words is the set B illustrated above, the same as the correct sentence.

2.2.2 Term Frequency - Inverse Document Frequency (*tf-idf*)

Another well known concept of IR that is frequently used is the *tf-idf*f weighting scheme. This is the combination of the definitions of *term frequency* (tf) and *inverse document frequency* (idf). $tf_{t,d}$ represents the frequency of the term t in document d [32].

Also, there are terms that are less important than others, and just the frequency of the term is not enough. Terms too frequent are common and less important, terms less frequent are more specific to that document and, therefore, more important. To attenuate the effect of terms that occur too often in the document collection, we have the inverse document frequency of the term t, idf_t, that is defined as [32]:

$$\text{idf}_t = \log\frac{N}{\text{df}_t} \tag{2.1}$$

Here, N is the number of documents at collection, and df_t is document frequency of the term t, *i.e.* the number of documents that have t [32]. If df_t is low, *i.e.* the term is rarer, idf_t is higher. The *tf-idf* scheme is, then, described as follows [32]:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \tag{2.2}$$

The example next presents two simple text files’ contents, one in Figure 2.3 and other in Figure 2.4, and their frequency tables are shown in Table 2.1 and Table 2.2, respectively.

Since there are 2 documents, $N = 2$. Calculating the *tf-idf* for the term “UML” in the first document, the $\text{tf}_{\text{UML},1} = 1$ and $\text{idf}_{\text{UML}} = \log\frac{2}{2} = 0$. Finally, $\text{tf-idf}_{\text{llll},1} = 0$ because this is a term that occurs in all documents.

```
Class diagrams are UML diagrams.
```

Figure 2.3: A first simple text document

```
UML models are useful.
```

Figure 2.4: A second simple text document

Table 2.1: The first document term count

Term	Class	diagrams	are	UML
Count	1	2	1	1

Table 2.2: The second document term count

Term	UML	models	are	useful
Count	1	1	1	1

On the other hand, the term “diagrams” occurs two times at the same document in Figure 2.3, so we have $\text{tf}_{\text{diagrams},1} = 2$ and $\text{idf}_{\text{diagrams}} = \log\frac{2}{1} \approx 0,301$. Finally, $\text{tf-idf}_{\text{diagrams},1} \approx 0,602$.

2.3 Recommender Systems

Recommender Systems (RS) are software tools and techniques to solve the information overload problem, where users are faced with more information than they can handle [46]. Many RS algorithms were inspired at the idea users often rely on recommendations that are passed by others, directly or indirectly (through recommendation texts, reviewers’ opinions, newspapers etc.) [46]. Despite some RS approaches have taken techniques from IR, the idea of the RS is to differentiate the relevant content. IR techniques are more interested in identifying similar items, and sometimes this is not what the users need. Some RS techniques can try to discover what other items could be also useful for the users, even if they are not so similar. Figure 2.5 shows a typical RS operation, in which the user starts by stating his/her preferences (implicitly or explicitly); then the system uses that preferences to build a profile for the user; with that profile and after getting the items, the system uses the user profile to identify relevant items to recommend to him/her. Usually recommender systems recommend a limited number n of items to the user, which can be user-defined.

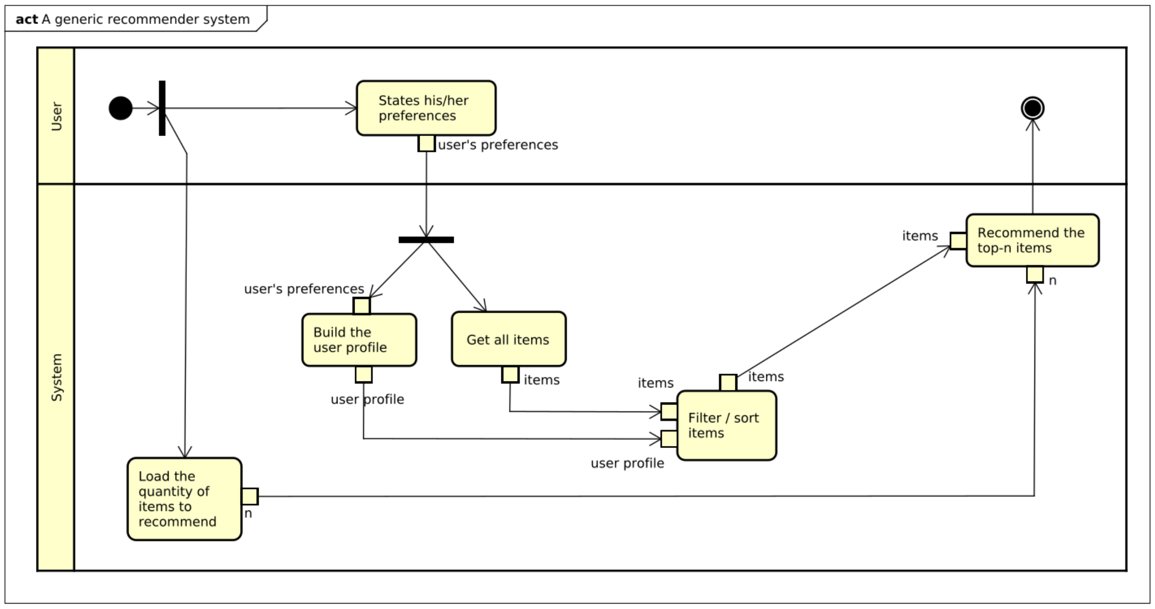


Figure 2.5: A typical Recommender System operation

Over the years, emerged RS applications for Software Engineering, as *eRose*, *Strathcona*, *Suade* and others [48]. In general, any Recommender System refers to three types of objects: the items, the users and the transactions [46]. Namely:

- **Item** is a generic term used to identify each one of the elements that are recommended by the Recommender System. The value of an item may be, for example, positive when a user like it, negative otherwise. According to the Recommender System core technology, the item can be described by a set of features and properties [46]. This feature set is the **item profile** [1];

- **User** is another key concept. The recommendations, in general, are directed to someone and, to do it, the Recommender System needs to collect the preferences of that individual, whether explicitly or implicitly expressed. This features' set is the **user profile** [46];

- **Transaction** is a recorded interaction between a user and the Recommender System. Several of them generate a data set that contains very important information that are generated during the human-machine interaction and which are useful to the recommendation algorithm who generates the system suggestions [46].

Jannach et al. [21] presents a well-known classification of Recommender Systems into four different approaches, namely:

- **Collaborative filtering:** Systems that, considering that users that shared tasted in the past will do the same now and in the future, recommend for the current user A the items selected by a similar user B that are "unknown" by A. Discovering similar users is done by the similarity of their historical data (e.g., purchased books history). Classical examples are the user-based news recommendation based on users with similar tastes presented in Resnick et al. [44] or in Goldberg et al. [19];

- **Content-based:** Systems where the content of the items being recommended is the main focus, generating recommendations based on features associated with the compared items [46] and the ratings that the user gives to items [9]. They have taken ideas from IR techniques, as the way of describing items and comparing them, for instance, but their difference lies in the purpose of each one as previously noted. One example is the news filter proposed by Lang [26];

- **Knowledge-Based:** Systems that recommend products mainly based on domain-specific knowledge about how certain items' features match preferences and needs of users. Burke [8] introduces and gives some examples of this type of system;

- **Hybrid Recommender Systems:** Systems that combine the previous approaches in order to compensate some of the disadvantages from one approach with the strengths from another. Burke [9] introduces several possible combinations of recommendation approaches to generate this type of system.

Formally, a RS can be described as the following function:

$$s : U \times I \to \mathbb{R} \tag{2.3}$$

where $U = \{u_1, u_2, ..., u_m\}$ is the set of users, $I = \{i_1, i_2, ..., i_n\}$ is the set of items, n is the amount of items, m the amount of users) and s is a function that estimates the utility, a real number, of $i \in I$ to $u \in U$.

2.4 Ontologies and knowledge representation

An ontology is a basic structure around which a knowledge base can be built [51]. We can consider two perspectives while talking about ontologies: a traditional one coming from philosophy, that focuses on categorical analysis (what are the existing entities and what are the categories of these entities) to inventory reality, and a computer science perspective known as ontology as technology, that focuses on the same questions but focuses on creating artifacts of reality to be used by software [42]. Despite similar, they are different perspectives and we are interested on the second definition.

Ontologies are being used in Computer Science and related fields because they help to categorize and structure entities and concepts of interest. Areas like artificial intelligence, knowledge representation, information science and database management frequently make use of ontologies [23]. Ontologies can be used to model the knowledge of artificial intelligence, educational data, medical information or any knowledge that one want to represent in a way to be processed by computers. One well known use of ontologies is at *semantic web*, that we present next.

2.4.1 The semantic web

The Semantic Web was proposed by Berners-Lee et al. [6] in 2001, which proposed a way to connect facts instead of just documents, adding semantic meaning to the elements and its connections. This extension of the Web allows machines to use that semantic data to understand what is being transferred, learning about the data and proposing better results to humans. As an example, imagine two web pages, a personal blog and a page who sells digital books; if the owner of the blog create a post about a book who is specifically sold by the other page, machines could use semantic data to connect one page to another, where they can find more information as, for example, some related books. In order to achieve this goal, it was necessary to create new languages and patterns specifically designed for data, and the major were the Resource Description Framework (RDF) [57] and the Web Ontology Language (OWL) [56], which we briefly present later.

2.4.2 Resource Description Framework (RDF)

RDF is a general-purpose language for data interchange on the Web [57]. It can be written in XML [59] and its core structure is represented by a set of triples consisting of a *subject*, a *predicate* and an *object*:

- **Subject:** An entity;
- **Predicate:** Also called a property of a triple. A subject can have one or more of them;
- **Object:** An object that belongs to one or more resources. An object can point to instances or be primitive types like *string*, *boolean*, *integer* or *float*.

A set of triples is called an *RDF graph* [58]. You can see an example of RDF triple at Figure 2.6 [58].

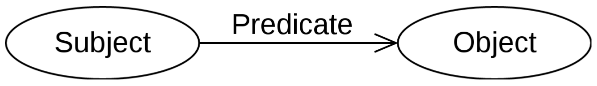


Figure 2.6: An RDF triple example extracted from W3C [58]

The Source Code 2.2 is an example that shows the object "algorithmBook" (line 5) that has a predicate "coverColor" pointing to the object "blue" (both in line 6).

Source Code 2.2: A brief example of RDF

```
1 <rdf:RDF
2     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3     xmlns:feature="http://www.saulotoledo.com.br/bookfeatures#">
4
5     <rdf:Description rdf:about="http://www.saulotoledo.com.br/books#
         algorithmBook">
6         <feature:coverColor rdf:resource="http://www.saulotoledo.com.br/
             colors#blue"/>
```

```
    </rdf:Description>
</rdf:RDF>
```

2.4.3 The Ontology Web Language (OWL)

RDF is not enough to describe rich ontologies, and the Ontology Web Language (OWL) is a W3C[2] proposal defined on top of it with this purpose in mind [56]. In order to classify things in terms of their meaning, OWL defines [55]:

- **Classes:** Describes the domain concepts. It is a way to classify individuals which share characteristics into groups;
- **Individuals:** Members (instances) of a class;
- **Properties:** Allows us assert general facts about the members of classes and specific facts about individuals.

The Source Code 2.3 is an example that represents in the OWL format some metaclasses of the UML metamodel. We model the UML metaclasses "Classifier" (line 12), "BehavioredClassifier" (line 17) and "Interface" (line 23). "BehavioredClassifier" and "Interface" extends "Classifier" by using the "subClassOf" relationship (lines 19 and 25). "Classifier" has a property called "isAbstract" (defined in line 6, relationship with "Classifier" on line 7). We also represent an individual whose ontology class is "Interface" (line 29).

[2]World Wide Web Consortium (`http://www.w3.org/`)

Source Code 2.3: A brief example of OWL

```xml
<!DOCTYPE rdf:RDF [ (...) ]>
<rdf:RDF xmlns="&umlclassmmo;#" (...)>
    <!-- OWL header omitted for brevity -->

    <!-- Definition of the property isAbstract -->
    <owl:DatatypeProperty rdf:about="&umlclassmmo;#isAbstract">
        <rdfs:domain rdf:resource="&umlclassmmo;#Classifier"/>
        <rdfs:range rdf:resource="&xsd;boolean"/>
    </owl:DatatypeProperty>

    <!-- OWL Class Definition - Classifier -->
    <owl:Class rdf:about="&umlclassmmo;#Classifier">
        <rdfs:label>Classifier</rdfs:label>
    </owl:Class>

    <!-- OWL Class Definition - BehavioredClassifier -->
    <owl:Class rdf:about="&umlclassmmo;#BehavioredClassifier">
        <rdfs:label>BehavioredClassifier</rdfs:label>
        <rdfs:subClassOf rdf:resource="&umlclassmmo;#Classifier"/>
    </owl:Class>

    <!-- OWL Class Definition - Interface -->
    <owl:Class rdf:about="&umlclassmmo;#Interface">
        <rdfs:label>Interface</rdfs:label>
        <rdfs:subClassOf rdf:resource="&umlclassmmo;#Classifier"/>
    </owl:Class>

    <!-- An instance of the class Interface - a car -->
    <rdf:Description rdf:about="&umlclassmmo;#car">
        <!-- Car is an individual (instance) of the Interface class -->
        <rdf:type rdf:resource="&umlclassmmo;#Interface"/>
        <!-- There is no abstract interfaces -->
        <umlclassmmo:isAbstract>false</umlclassmmo:isAbstract>
    </rdf:Description>
</rdf:RDF>
```

Chapter 3

Techniques for UML Class Diagrams Recommendations

This work proposes to recommend UML class diagrams to users interested in receiving recommendations of diagrams that could be useful for them based in a set of features they have informed. In order to achieve this goal, we try to find the best recommender approach for this scenario by proposing and comparing four proposals: (i) a random recommendation approach, (ii) a bag-of-words based approach, (iii) a classic content-based approach and (iv) a knowledge based approach.

The random approach is a baseline that randomly suggests items and does not need to know the diagrams' content. The bag-of-words approach uses the content of the XMI file as text to identify relevant items. On the other hand, the content-based and the knowledge-based approaches need to represent the items' and users' profiles in some way, and it is a common practice into Recommender Systems area to do it as vectors [45]. With that in mind, we propose a vector representation for users and UML class diagrams items.

Before discussing about the techniques, Section 3.1 presents the profiles that are used by the content-based and the knowledge-based approaches. Then, Section 3.2 presents each one of four recommender algorithm proposals.

3.1 User and item profiles

In order to generate recommendations with a content-based or a knowledge-based algorithm, a suitable representation for the items' and users' profiles is necessary. Initially in this work context, there is a set of UML diagrams for which there is no enough information about the domain where they came from; thus, this work needs a representation that ignores the diagrams' domain. One way to describe these diagrams could be by using a set of features that users might be interested in, such as *interfaces*, *generalizations*, *enumerations* and so on. Thus, this work represents their profiles as a vector where each component is one of these features, extracting them from the class diagrams. About the users, once they are interested in that diagrams, this work represents their interests as a vector in the same format.

Thinking about the features that should compose this vector, this work proposes, based in the UML metamodel, the set of UML class diagram's features presented below (where "P" in the acronym stands for "presence of"):

- **Composite Aggregation (PCOA):** The diagram has at least one composition;
- **Shared Aggregation (PSHA):** The diagram has at least one aggregation;
- **Association Class (PASC):** The diagram has at least one association class;
- **Dependency (POD):** The diagram has at least one dependency;
- **Attribute with Default Value (PADV):** The diagram has at least one attribute initialized with a default value;
- **Realized Interface (PRI):** The diagram has at least one realized interface;
- **Generalization (POG):** The diagram has at least one generalization;
- **Interface (POI):** The diagram has at least one interface (even if it is not realized);
- **Derived Attribute (PDA):** The diagram has at least one derived attribute;
- **Static Operation (PSO):** The diagram has at least one static operation;
- **Port (POP):** The diagram has at least one port;

- **Qualifier (POQ):** The diagram has at least one qualifier;
- **Abstract Class (PABC):** The diagram has at least one abstract class;
- **Enumeration (PEN):** Indicates whether the diagram has at least one enumeration;
- **Navigation Arrows in Associations (PNAA):** In UML, an association can be uni or bi-directional. If an association is uni-directional, an arrow indicating the direction must be present. If it is bi-directional, arrows in both sides of the association are optional. This feature indicates whether the diagram visualization has at least one association with at least one navigation arrow;
- **Generalization Set (PGS):** Diagram has at least one generalization set;
- **Template Class (PTC):** The diagram has at least one template class;
- **Static Attribute (PSA):** The diagram has at least one static attribute;
- **Abstract Operation (PAO):** The diagram has at least one abstract operation.

In the profile vector, the value for each feature can be 0 (absence) or 1 (presence). As an example, the diagram in Figure 2.1 contains the enumeration "Color" and its value for PEN is 1; the value for PASC is also 1, since the diagram contains the association class "Membership"; but it does not contains ports, thus the value for POP is 0. It is a limitation of this proposal consider only binary features. A future work could evaluate the quantity for each vector component, but some of these features can appear more frequently than others (*ports* and *template classes* are less common than *dependencies*, for instance), and this can interfere in the results and should be carefully considered.

3.2 Recommender systems approaches proposed in this work

As introduced before, this work proposes and compares four recommender approaches, that are detailed below. The Subsection 3.2.1 presents the random algorithm; the Subsection 3.2.2 presents the *bag-of-words* based algorithm adapted from Information Retrieval techniques;

then the Subsection 3.2.3 presents a content-based algorithm that uses well known Recommender System techniques; finally the Subsection 3.2.4 presents the main concepts of a new knowledge-based recommender systmem algorithm. All these approaches returns the *top-n* items, considering that the *top-n* items for the random algorithm can be formed by any of them in any order.

Considering the contexts of the users and diagrams could be useful to improve the recommendations, since different groups of users can be interested in different types and details of diagrams. However, the following proposals do not consider it for simplicity, since this task requires a study that consider grouping users and diagrams. This can be accomplished in a future work.

3.2.1 Random items recommendation

This proposal collects all UML diagrams from the database into a set, shuffles them and returns *n* items. The idea behind this approach is to be a baseline for algorithms comparison. All permutations should occur with equal likelihood.

The approach's results can vary between different implementations. We have created our own implementation for this algorithm by using *Collections.shuffle()*[1] from the Java API. The internal applied algorithm in this method is unknown but, for example, the OpenJDK[2] implementation[3] basically does a $\theta(n)$ *Fisher-Yates shuffle* [16]. Figure 3.1 presents this approach's algorithm: the user states his/her preferences, but the system ignores them and just shuffles the items, recommending n random ones to the user.

This approach is limited to randomly recommend items and represents the worst possible recommendation. As it simply randomizes the results, it also ignores the context of the diagrams, their domains or any patterns they could represent, and cannot detect any of the elements or describe relations between them unless it is for a lucky draw.

[1]You can find more information about at Oracle's documentation for Java: `http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html`

[2]An open source JDK implementation. You can find more about at `http://openjdk.java.net/`

[3]This particular source code is available at `http://hg.openjdk.java.net/jdk7/jdk7/jdk/file/9b8c96f96a0f/src/share/classes/java/util/Collections.java`

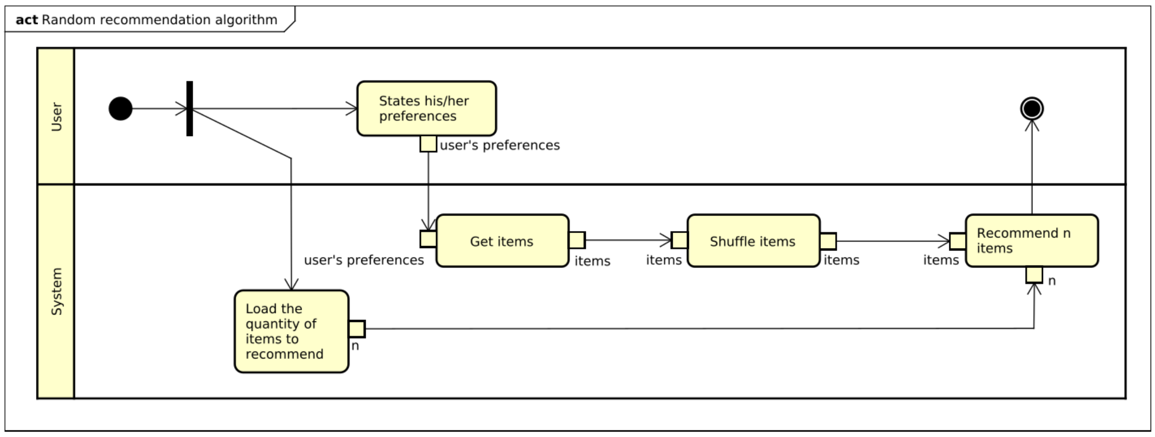


Figure 3.1: The random recommendation algorithm

3.2.2 Bag-of-words based approach

Since UML diagram files can be stored in the XMI file format, we can consider them as simple text files and extract a set of words and their occurrences. After that, we can use the bag-of-words model introduced in Subsection 2.2.1 to represent the diagrams, and the *tf-idf* weighting scheme, introduced in Section 2.2.2, to get the recommended UML class diagrams. Since the bag-of-words set is extracted from the file contents, this approach can be considered as a naive content-based one. This entire approach is a well-known IR technique that is reused in this study; the only innovation is its application to XMI files.

In summary, the algorithm indexes the files and then rank by using *tf-idf*. To perform searches, the user provides a search string to a tool that runs the algorithm. The Table 3.1 contains an example of the search strings used for each user feature that can be found in the user/item profiles: most of the "or" symbols are used to search for UML1.x and UML2.x files (the XMI dialect changes while representing different versions of UML); others are used to identify different elements that could be considered in that search (operations are present in classes or interfaces, and the system can search for both, for instance); the plus signs are used to identify what elements should be found together (as an example, for presence of static attributes – PSA – classes (*i.e.* "uml:Class") with attributes (*i.e.* ownedAttribute) that are static (*i.e.* "isStatic=true"). The complete list of strings can be found in Appendix B.

Figure 3.2 presents this approach's algorithm: the user states his/her preferences; after, the system builds the search string by concatenating the strings for all features selected by

Table 3.1: Examples of search strings for the bag-of-words approach

User selected feature	**Search string**
Presence of Static Operation (PSO)	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (ownedOperation || "UML:Operation") + ("isStatic=true" || "ownerScope=classifier") + (packagedElement || "UML:Model")
Presence of Attribute with Default Value (PADV)	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (ownedAttribute || "UML:Attribute") + (defaultValue || Attribute.initialValue) + (packagedElement || "UML:Model")
Presence of Association Class (PASC)	("uml:AssociationClass" || "UML:AssociationClass") + (packagedElement || "UML:Model")

the user; then, the system searches for the items and orders the results by using the values for *tf-idf* (see Subsection 2.2.2) and finally recommend the n best results. Each search string starts with the most specific term and finishes with the most general one, that is always present in UML files, precluding thus the system returns an empty result. If the system does not find items that are equivalent to the entire string, it will return items that correspond to the more generic one. For instance, the PASC string contains "uml:AssociationClass" and "packagedElement", if this combination does not exists in the UML class diagram database, the system will search for just "packagedElement".

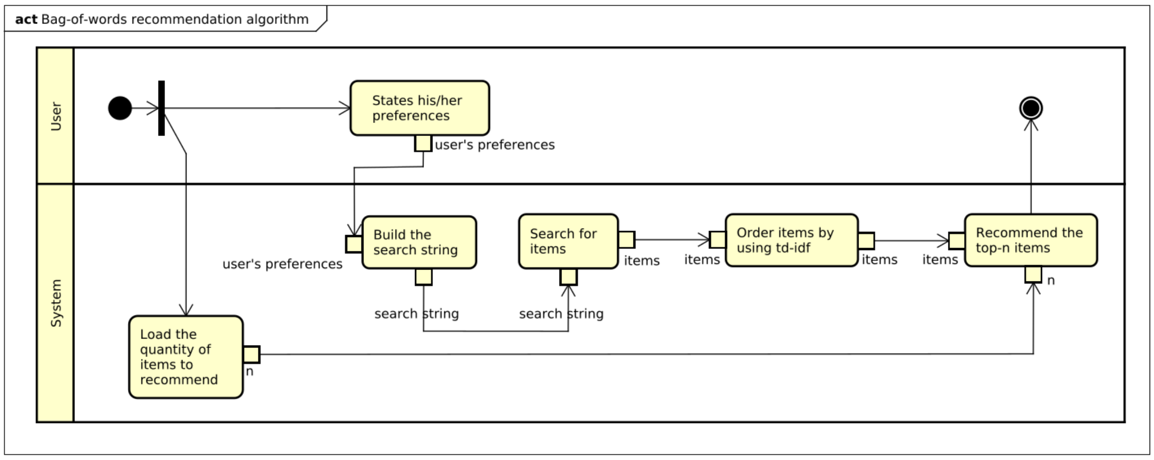

act Bag-of-words recommendation algorithm
User
System
States his/her preferences
user's preferences
Build the search string
user's preferences
Search for items
items
items
Order items by using td-idf
items
items
Recommend the top-n items
n
search string
search string
Load the quantity of items to recommend
n

Figure 3.2: The bag-of-words recommendation algorithm

Considering w as the number of words in the XMI document and d the number of diagrams, extracting the terms from the diagrams costs $\theta(w \cdot d)$, since for each diagram, each word should be verified. Searching is also a linear $\theta(w \cdot d)$ for the same reason. The *tf-idf* in the end is $\theta(w^2 \cdot d)$ and the *top-n* calculations is $\theta(d \cdot \log d)$. Thus, the total cost of calculating the *top-n* results in this algorithm is $\theta(w \cdot d) + \theta(w^2 \cdot d) + \theta(d \cdot \log d$, that is $\theta(w^2 \cdot d)$.

This model is simple to implement, since there is no need to create a new user and item's representations in addition to the bag-of-words structure. Frameworks that provide indexing and search technologies, as Apache Lucene[4], facilitate the implementation of this approach. Unfortunately, there is a major issue with our implementation: it cannot differentiate between the elements tags and their names or values, and this can lead to wrong results; if there are comments in the file containing any substring used by the search string, for instance, there

[4]`https://lucene.apache.org/`

will be a wrong count of that element. This choice was made because the research database does not contain these situations (unless there has been human error, a threat for the validity for the results of this research that should have a minor impact), and implementing a custom parsing that ignores the attributes' values and others should be costly, time consuming and would complicate the algorithm definitions.

This approach is limited to the use of XMI files, since it needs to read the UML files as text, and contains the previously mentioned issue with the elements and tag names. It also cannot diferentiate the "isStatic" from attributes and operations, for instance, since it searches for this string in the entire file, and this can also lead to wrong results. In addition, it ignores the context of the diagrams, their domains or any patterns they could represent. Finally, it can only detect the presence or absence of a determined feature and can not identify the quantity of a determined element in the diagrams or describe relations between them.

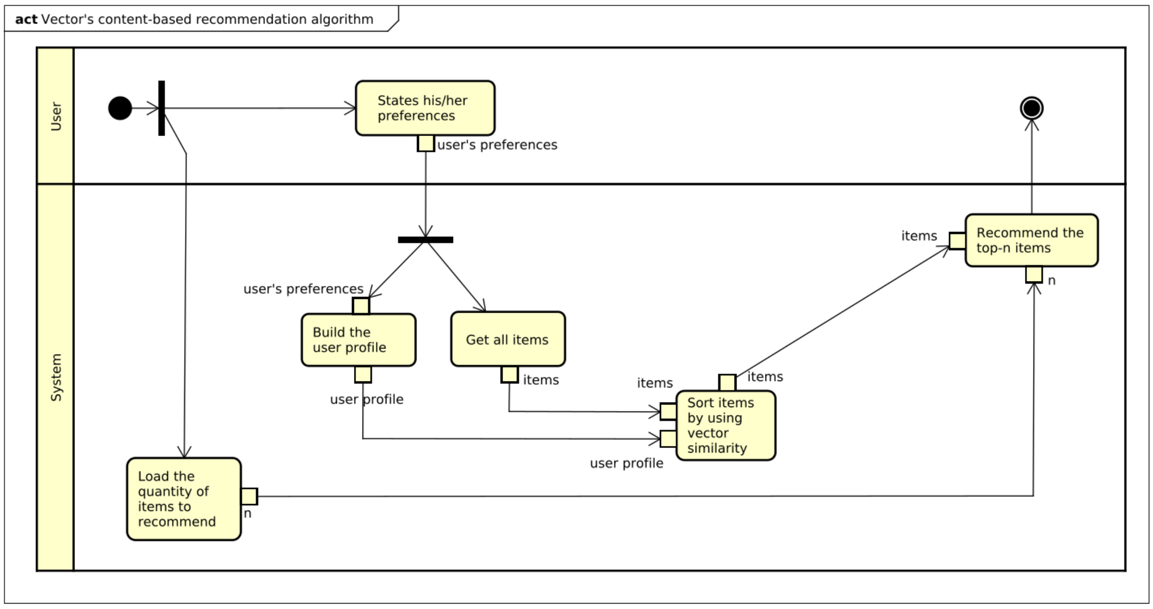
3.2.3 Item's vector (content) based approach

The previous proposal, that is based in Information Retrieval techniques, trust in the files' content as strings, and this can lead to mistakes as previously introduced. Also, in that proposal, the term count must be done in the current file and in all the other files in order to calculate the *tf-idf* for them. With that in mind, this work proposes another way to recommend diagrams by using an items' vector Recommender System content-based approach. Figure 3.3 presents its algorithm: the user states his/her preferences; after, the system, by using the profile defined by a set $F = \{f_1, f_2, ..., f_g\}$ of features (see Section 3.1) to describe items (*i.e.* UML class diagrams) and users' interests, calculates the similarity between the user's profile and each item stored in the database; finally recommend the n most similar results.

Considering m as the vector dimension, the user's profile as a vector $\vec{u}$, and the item's one as a vector $\vec{i}$, we use the cosine similarity function (Equation 3.1) [3] to calculate the proximity between users and items.

$$sim(\vec{u}, \vec{i}) = \frac{\sum_{k=1}^{m} u_k \cdot i_k}{\sqrt{\sum_{k=1}^{m} (u_k)^2} \cdot \sqrt{\sum_{k=1}^{m} (i_k)^2} \tag{3.1}$$

The cosine similarity returns a value between 0 and 1. A value of 1 indicates that both



act Vector's content-based recommendation algorithm
User
System
States his/her preferences
user's preferences
user's preferences
Build the user profile
user profile
Get all items
items
items
Sort items by using vector similarity
items
user profile
items
Recommend the top-n items
n
Load the quantity of items to recommend
n

Figure 3.3: The vector's based recommendation algorithm

vectors are equal. The top-n items for the user $u \in U$ are computed as presented at Equation 3.2, calculating the similarity between $\vec{u}$ and each $\vec{i} \in I$, and returning the n elements with higher similarities (notice the n parameter in argmax).

$$\text{top-}n(\vec{u}) := \underset{\vec{i} \in I}{\overset{n}{\operatorname{argmax}}}\ sim(\vec{u}, \vec{i}) \tag{3.2}$$

Considering the data summarized in Table 3.2, i_1 as a representation for the diagram from Figure 2.1, and i_2 as a representation for a similar diagram, but that does not have the features *presence of enumerations (PEN)* and *presence of derived attributes (PDA)*, both features that were selected by the user, lets compute the *top-1* elements. After applying Equation 3.1 between u_1 and each item, the results are $sim(\vec{u_1}, \vec{i_1}) \approx 0.866$ and $sim(\vec{u_1}, \vec{i_2}) \approx 0.333$. Finally, top-$1(u_1)$ comprises i_1 (the diagram from Figure 2.1).

Considering w as the number of words in the XMI document and d the number of diagrams, extracting the items' vectors for the diagrams costs $\theta(w \cdot d)$ because each word in the XMI file must be checked to identify if it represents the presence of a feature. Once it is done it can be stored in a database. On the other hand, considering that the number of features is a constant c, the cosine similarity has a constant cost, since if depends of the length of the vectors (that is the number of features), and the similarity for all diagrams is $\theta(c \cdot d)$. Finally, the *top-n* is calculated and costs $\theta(d \cdot \log d)$. Thus, the total cost of the approach is

Table 3.2: Examples of user and items profiles

	PCOA	**PSHA**	**PASC**	**POD**	**PADV**	**PRI**	**POG**	**POI**	**PDA**	**PSO**
u_1	0	0	1	0	0	0	0	0	1	0
i_1	0	0	1	0	0	0	1	0	1	0
i_2	0	0	1	0	0	0	1	0	0	0
	POP	**POQ**	**PABC**	**PEN**	**PNAA**	**PGS**	**PTC**	**PSA**	**PAO**	
u_1	0	0	0	1	0	0	0	0	0	
i_1	0	0	0	1	0	0	0	0	0	
i_2	0	0	1	0	0	0	0	0	0	

$\theta(c \cdot d) + \theta(d \cdot \log d)$, that is $\theta(d)$.

The idea of using the a vector similarity method between profile and item's vectors and returning the *top-n* items is a well-known RS technique. The innovation of this approach lies in describing UML diagrams as vectors in order to be possible to apply this technique.

This approach is limited to a static description of the diagrams and users, disregarding the context of them, the domains of the diagrams or any patterns they could represent. It also can only detect the presence or absence of a determined feature and can not identify the quantity of a determined element in the diagrams or describe relations between them.

3.2.4 Knowledge-based approach

The previously presented proposals can identify features in the diagrams that were selected by the users, but they are unable to discover other features that the user might be interested in. The knowledge-based approach uses the same vector used by the item's vector content-based one, but uses knowledge-based information to increment it with values of partial interests in features that were not selected by the user. In other words, it discovers in what other features the user might be interested in and the weight of the interest. Therefore, we use almost the same algorithm, but before calculating the similarity, we apply a transformation on the user's vector, expanding it to contain other features than those explicitly indicated by the user.

Figure 3.4 presents the approach proposal: the user states his/her preferences; next, the algorithm builds an initial user profile and expands it by using a newly proposed knowledge-based generic algorithm that uses (i) the user's preferences, (ii) an OWL ontology file, (iii) a feature-to-ontology class/attribute mapping table, and (iv) some adjustment parameters; then it calculates the similarity between the user's profile and each item stored in the database; finally recommend the n most similar results. We named the proposed knowledge-based algorithm OntoRec and it is a completely new algorithm.

Since the idea is to recommend UML class diagrams, it was built an ontology based on the UML metamodel of the current version of the UML language, 2.4.1[5]. The following topics contains a brief introduction about how the entire approach was built, starting in how its ontology was built, after explaining the concept of *mapping table* that is used by the algorithm, then introducing the algorithm parameters, and finally a brief summary about how the recommendations are performed. Further details about this approach can be found in Chapter 4 and in Appendix A.

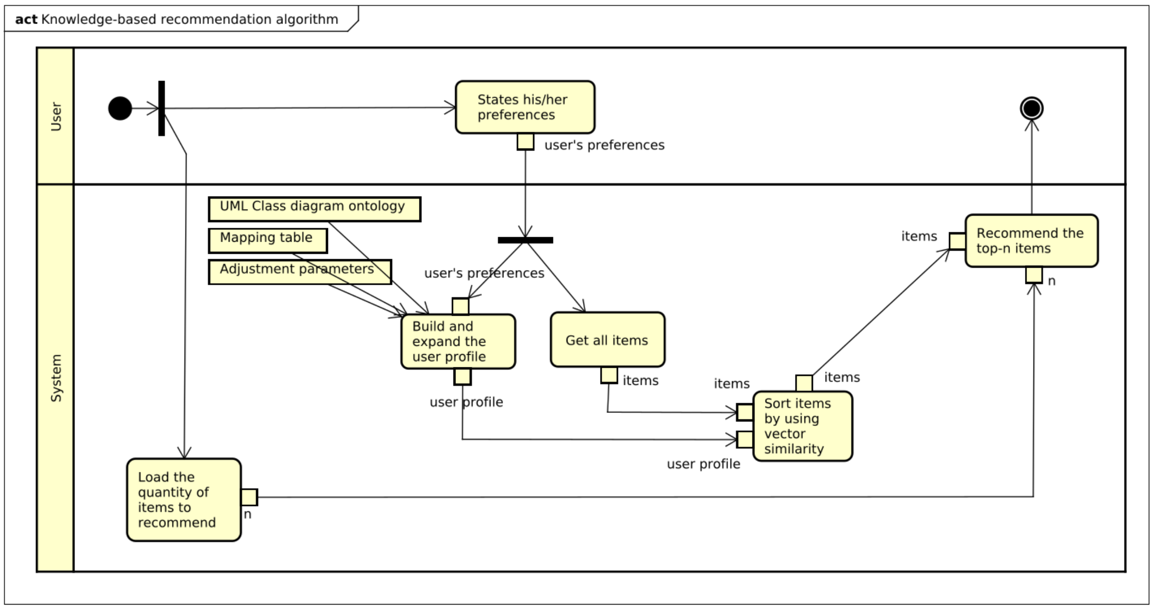

act Knowledge-based recommendation algorithm
User
System
States his/her preferences
user's preferences
UML Class diagram ontology
Mapping table
Adjustment parameters
user's preferences
Build and expand the user profile
Get all items
items
user profile
items
items
Sort items by using vector similarity
user profile
items
Recommend the top-n items
n
Load the quantity of items to recommend
n

Figure 3.4: The knowledge-based recommendation algorithm

Considering w as the number of words in the XMI document and d the number of diagrams as in the item's vector based approach (Section 3.2.3), extracting the items' vectors for the diagrams also costs $\theta(w \cdot d)$ for the same reasons, and once it is done it can also be

[5]Available at `http://www.omg.org/spec/UML/2.4.1/`

stored in a database. In the same way, the total cost of the approach disregarding the vector expansion is $\theta(d)$, but the total cost of this approach is higher and is detailed in Appendix A.

As the content-based approach, despite the extension of the user profile, this approach is also limited to a static description of the users and diagrams, disregarding the context of them, the domains of the diagrams or any patterns they could represent. It also can not identify the quantity of a determined element in the diagrams or describe relations between them.

Building the ontology

According to Section 3.1, the user profile vector contains a well defined feature set. In the UML metamodel, each one of these features can be found as (i) a metaclass or as (ii) a meta-attribute. Thus, the next step is to connect the feature list to the ontology's elements (classes or properties, in this case) and, to do that, we have mapped each feature from the profile's vector to classes or attributes from the UML metamodel. As an example, consider the feature *presence of static operations (PSO)*: this feature was connected to the attribute "isStatic" of the class "Operation" in the UML metamodel. After applying this process to all of the features, the *mapping table* required by OntoRec (see Section 4.3 for details) presented at Table 3.3was built.

The approach's ontology is a simplified version of the UML metamodel in which are considered just the elements related to the feature set. Thus, the classes and attributes in the ontology are the same as in the UML metamodel. Each unique related UML metaclass presented in Table 3.3 is built as a class in the resulting ontology. After that, all its ancestors until *Element* (that is the top ancestor element according to the UML metamodel) are also represented as classes in the ontology. The inheritance relationships between all classes are also built into the ontology as a "is subclass of" relationship. Lastly, all the related properties in that table are modeled as attributes of their respective classes. It is important to note that some features are mapped directy to classes, as *presence of association classes (PASC)*, once the metaclass *AssociationClass* is enough to describe it; others are mapped to attributes as *presence of abstract classes (PABC)* and *presence of template classes (PTC)*, once they are more related to attributes in the metamodel than to the metaclass *Class*.

Figure 3.5 presents a graphical representation of the reached ontology; as an ex-

Feature	**Related metaclass in the UML metamodel**	**Related meta-attribute in the UML metamodel**
PSA	Property	isStatic
PADV	Property	defaultValue
PSO	Operation	isStatic
PRI	Interface	-
PASC	AssociationClass	-
PABC	Class	isAbstract
PTC	Class	isTemplate
PEN	Enumeration	-
PCOA	Association	ownedEnd.aggregation = composite
PDA	Property	isDerived
PGS	GeneralizationSet	-
PNAA	Association	navigableOwnedEnd
POD	Dependency	-
POP	Port	-
POQ	Property	qualifier em Property
PSHA	Association	ownedEnd.aggregation = shared
POG	Generalization	-
POI	Interface	-
PAO	Operation	isAbstract

Table 3.3: Feature’s mapping to the UML metamodel

ample, “isDerived” is an attribute from the class “Property”, which is a subclass of “StructuralFeature”, “ConnectableElement” and “DeploymentTarget”. It is important to note that are modeled only the generalizations from the UML relationships. Other approaches could consider others relationships between the classes, as dependencies, compositions and others, but this is not the case and can be addressed in future works.

The mapping table

There is a semantic difference between the ontology class or attribute and a feature in the list: a feature is something that the user is interested in, the classes and attributes are things that have semantic meaning in the ontology space. In order to connect these concepts, OntoRec needs a structure which links them. This structure is named *mapping table*. This table will somehow link features and the ontology classes to use the ontology knowledge to infer a proximity result between features. For instance, one can connect the diagram’s feature *presence of abstract classes (PABC)* to the attribute “isAbstract” in the class “Class” in the ontology.

Note that a feature can be mapped to a class or an attribute, and we have done something similar before when we mapped features to the UML metamodel’s classes. Thus, we already have the mapping table: the Table 3.3, considering that the metaclasses are ontology classes.

The adjustment parameters

OntoRec receives the same vector used in the content-based approach and enriches it with background knowledge. The updated vector is then used to calculate the recommendations.

To discover knowledge to enrich the input vector, it is necessary to choose some fine tuning parameters. The main parameter is τ, and it ranges from 1 to the height of the ontology graph that, according to Figure 3.5 (that represents the proposed ontology), is 7. For now, one needs to know that the higher is τ, the higher is the chance to affect other features. If τ is smaller, the enhanced vector is very similar to the first one.

With this in mind, it is possible to deduce that if a lower value for τ is chosen, the recommendations will be similar to that from the content-based approach, and this is something that is not wanted. The higher is τ, the more features it affects, and if τ is the height of the ontology, it will affect all features. Since recommending all features is like suggesting

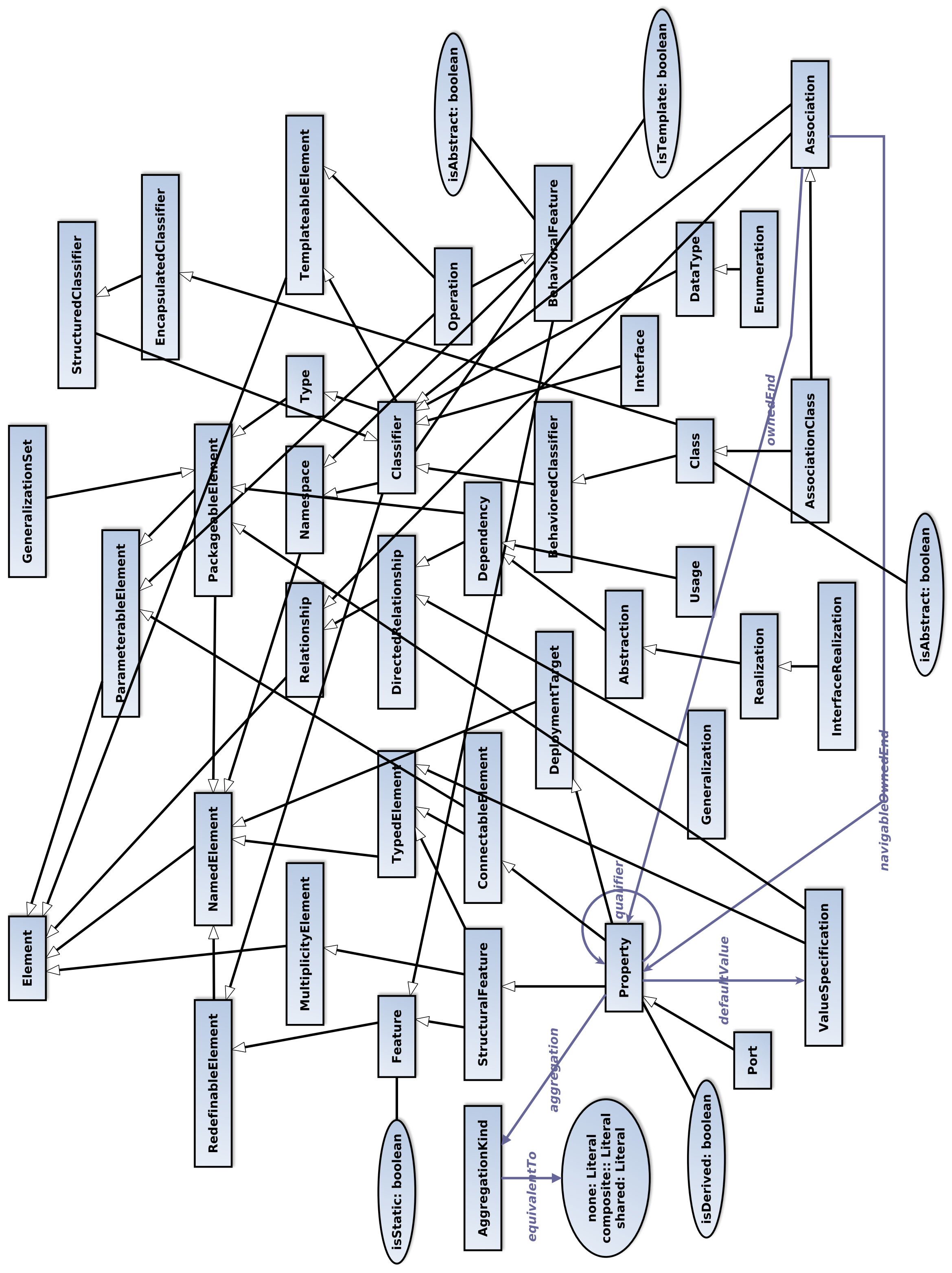


Figure 3.5: The UML class diagrams ontology

all books in a library for somebody that is looking for a few of them, a lower value for τ is desirable. Therefore, this work uses $\tau = 3$ as the value to be used in the experiments. τ will allow that the approach affect the nearest features, discovered by OntoRec by using inheritance; in Figure 3.5, "Abstraction" and "Usage" are closer to each other than "Usage" and "Property", for instance.

OntoRec has also two binary parameters, λ and υ. λ will make OntoRec ignore elements with only one child , increasing the chance to discover similar nodes, and υ will decrease the final values of the discovered features. This work does not comprises an experiment to discover the impact of choosing different values for these parameters. . Thus, we chosen $\lambda = 1$ and $\upsilon = 0$ as an initial setup, which is the way the algorithm was thought before the idea of these parameters, but they were proposed because we believe that they can be important and should be evaluated in future researches. This is also true about the other parameters: different ontologies and problems could be affected in a distinct way for different setups, but this should be evaluated by an specific experiment.

Finally, OntoRec computes the enhanced vector values by navigating through the ontology nodes, and it does this task by using one of two approaches: the Breadth-First Search (BFS) and the τ-nth Ancestor approaches. The same way as before, once both will do the same job and there is no statistical data to support this decision, it was chosen the simplest one: the BFS approach.

After these configuration options, it is possible to run the knowledge-based approach.

Running the recommendations

This is the last step for this approach. Since all the OntoRec's prerequisites are set, it is just needed to get the user's interest vector and send it (with all the other requirements) to OntoRec. The algorithm will return a new expanded user's interest vector. OntoRec does not calculate similarity, it only enriches the input vector with background knowledge. Thus, this work has have chosen to use the same similarity approach that was used on one of the proposed content-based approaches: the cosine similarity. The recommendation is then done as described in that approach.

Chapter 4

OntoRec: A Recommender Profile Generation Approach Based on Ontologies

One of the main contributions of this work is the proposal of the OntoRec (Ontology based Recommendations), an algorithm that generates new profiles (user's or items' ones) based on an ontology. OntoRec was initially developed as an algorithm to recommend UML diagrams based on an ontology representation of the UML metamodel, but evolved to a generic algorithm that can be used for any situation where similar ontologies can be defined and used.

The main idea of OntoRec is to use a domain ontology to discover related concepts that may be of interest to users. It maps the initial user profile to elements in the ontology in order to discover related attributes, giving them different weights based on how much related they are. Thus, the algorithm can be used to expand user profiles to discover possible related concepts for each profile.

In order to achieve its results, OntoRec needs to know how to use the ontology to infer what diagrams can be useful to the users. The users inform in what features they are initially interested, so OntoRec uses the classes of the ontology that represents each one of the selected features and its attributes to discover other features that can be useful to them. Thus, the first step is to understand what are the differences and similarities between ontology classes and domain features (Section 4.1).

Going further, once is known how OntoRec discover how ontology classes are useful for its processing, it should be known how OntoRec rely in the subclasses relationships in the ontology to discover related classes that were not selected by the user, and that can be useful (Section 4.2).

After that, it is time to understand more deeply as the algorithm relates features in the real world to classes and attributes in the ontology, explaining what is the *mapping table*, a structure that lists the features and their respective classes and attributes in the ontology (Section 4.3).

OntoRec contains some configuration parameters. They can change the results of the algorithm and must be defined before its execution. The first and most important parameter is τ, that defines the number of ancestors of a class in an ontology that can be achieved. τ is presented in details in Section 4.4. Section 4.5 introduces how the user profile is built, and this is done with the help of two other parameters, λ (Subsection 4.5.1) and υ (Subsection 4.5.2) and a third that is a choice of the routing method that is used by the algorithm to navigate the ontology (Subsections 4.5.3 and 4.5.4). The last three parameters are detailed in the discussion about the user vector prepared by OntoRec in Section 4.5. Support information can be found in the Appendix A.

4.1 Features versus Classes

The first main issue we should consider before starting to describe OntoRec is the difference between what we call "features" and the ontology concept "classes". Classes are the concepts presented in the ontology and features are qualities in the real world problem. Therefore, lets imagine that we have an ontology based on the UML metamodel to represent a UML class diagrams database. In this ontology, each UML metaclass is modeled as an ontology class, and a relationship "subclass of" for each subclass relationship. In order to simplify our example, suppose that the diagram in Figure 4.1 (that is a simplification of the Figure 3.5) represents the final ontology. The elements depicted by boxes are ontology classes, and the elements depicted by circles are attributes. For instance, "isStatic" is an attribute of "Feature", that in turn is a subclass of "RedefinableElement".

Now, suppose that it is desirable to create a Recommender System to recommend UML

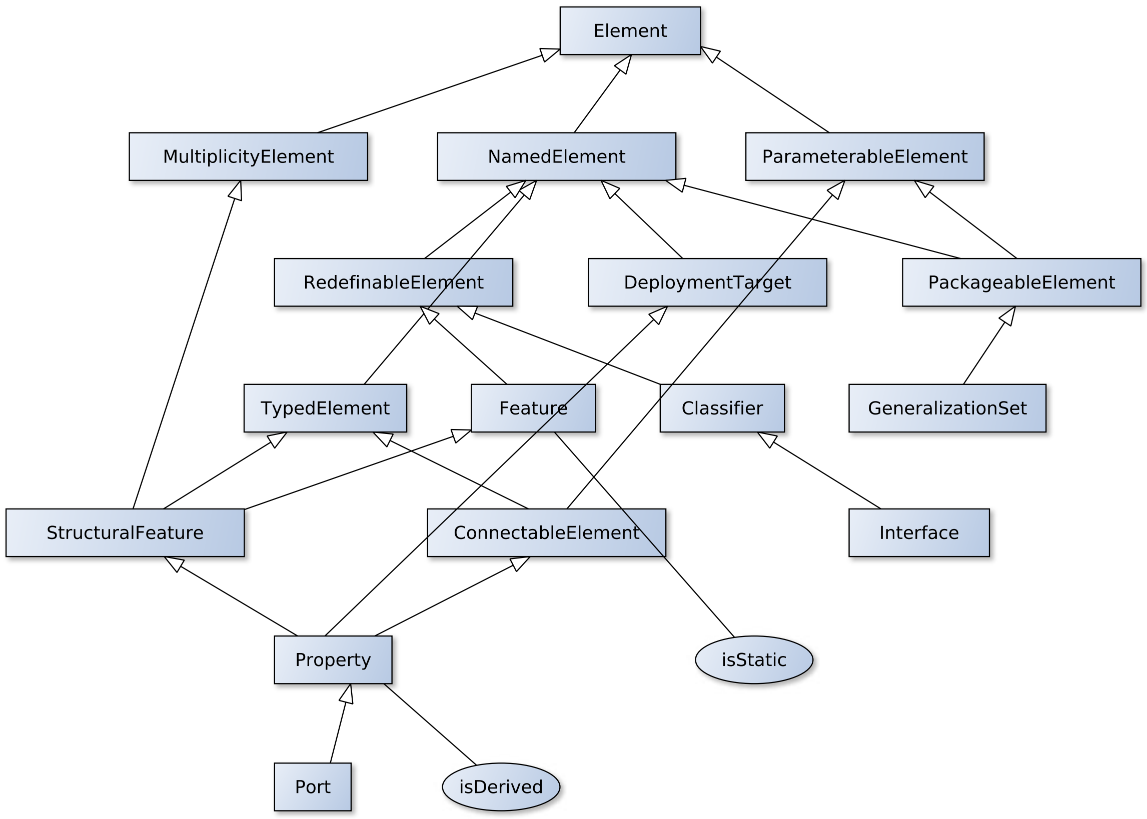


Figure 4.1: Simplified class diagram ontology

class diagrams to users by using the information in the ontology in Figure 4.1, using the relationship between the classes to discover diagrams to recommend to the user. With this information, it is possible to answer the following question: "if the user is interested in *static attributes*, what other features he could be interested in?"

Considering that users are interested in features like presence of *static attributes*, *interfaces*, *generalization sets* and others, these features are used to build a binary profile. Together, they compose an user or item profile that describes the interest or presence of the related feature. Table 4.1 presents some hypothetical UML class diagrams to illustrate our discussion. The features in the table are some of the described in Section 3.1. The diagrams may contain more than one of each feature.

Put all these concepts, it is finally possible to explain the difference between a feature like "Presence of Interface (PIN)" and the ontology class "Interface". Although they are related, the first one is some kind of interpretation of the problem instance, the second one is a static concept in the ontology domain. OntoRec uses the concept of **mapping table**

Table 4.1: UML class diagrams examples

Diagram Name	**PSA**	**PDA**	**POP**	**PIN**	**PGS**
Class Diagram 01	Yes	Yes	Yes	No	No
Class Diagram 02	No	No	Yes	No	No
Class Diagram 03	Yes	No	Yes	No	No
Class Diagram 04	Yes	No	No	Yes	No
Class Diagram 05	No	Yes	No	No	Yes

(see Section 4.3) to link both concepts and use the relationship among the concepts of the ontology to infer a proximity value between concepts in our problem setting.

4.2 The need of inheritance

OntoRec relies on the ontology relationship “is subclass of”. This is a main definition of the algorithm that reduces its complexity. Only this kind of relationship and the classes attributes are considered by OntoRec, and every class in the ontology (except the root) should be subclass of another one. One should also avoid circular dependencies (where some class ancestor is subclass of one of its descendants), or OntoRec will enter in an infinite loop while trying to find all paths between an ancestor and its mapped children.

One can note that in this kind of relationship there are **inherited attributes**. In our example, “StructuralFeature”, “Property” and “Port” are descendants of “Feature”, so the first three have, by inheritance, the attribute “isStatic”, as shown in Figure 4.2 .

4.3 The Mapping Table

In the previous section, features that are related to classes (“Presence of Interface (POI)”, that is related to the class “Interface”, for example)and features related to attributes (“Presence of Static Attribute”, that is related do the attribute “isStatic”) have been defined. The mapping table is an OntoRec structure that is capable to map **features** from the user profile to **classes or attributes** from the ontology, whether the last ones are inherited or not. Ta-

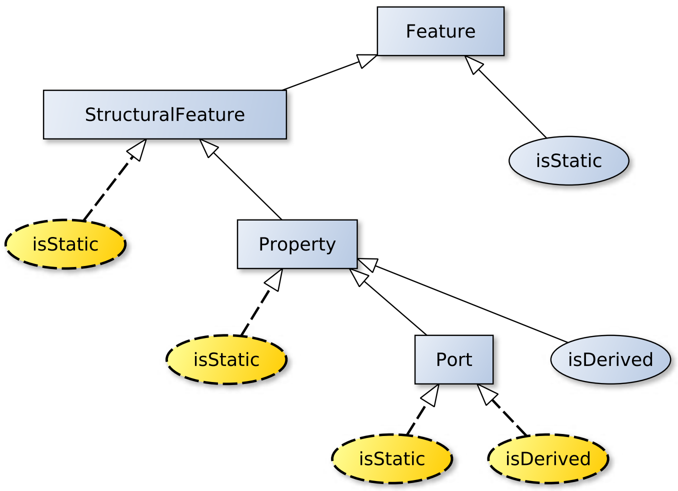


Figure 4.2: Ontology inheritance example for OntoRec

ble 4.2 is an example of mapping table: in this table there is a mapping, for instance, from the feature “POI” directly to the class “Interface”, whereas there is a mapping from the feature “PSA” to the attribute “isStatic” of the class “Property”.

Profile feature	Ontology class	Ontology attribute
POI	Interface	-
PGS	GeneralizationSet	-
POP	Port	-
PSA	Property	isStatic
PDA	Property	isDerived

Table 4.2: A Mapping Table Example

In Figure 4.1, the attribute “isStatic” is connected to the class “Feature”. The profile in Table 4.2 contains a feature called “PSA”, that stands for *presence of static attribute*, whose meaning is closer to “Property” than “Feature”. Since “Property” is a descendant of “Feature”, the attribute “isStatic” was mapped into “Property”, which it is possible due to the inheritance presented in Section 4.2.

By using the mapping table, every time that is considered a feature in the recommender algorithm, the mapped ontology elements are used to infer the most (or less) related features. The next sections will present how this is used by the algorithm.

4.4 The τ parameter

The τ parameter is known as **influence range factor**; it defines how much ancestors can be achieved given a mapped selected feature. Figure 4.3 helps to illustrate an example of how this parameter works: considering that the feature *presence of static attribute (PSA)* is mapped to the attribute *isStatic* into *Property* as explained in Section 4.3, if τ is 3, it is possible to achieve, as exemplified by $e_0 \rightarrow e_1 \rightarrow e_2 \rightarrow e_3$ in that figure, the following τ*-nth* ancestors:

- **MultiplicityElement**: by the path *isStatic* $\rightarrow$ *Property* $\rightarrow$ *StructuralFeature* $\rightarrow$ *MultiplicityElement*;
- **TypedElement**: by the paths *isStatic* $\rightarrow$ *Property* $\rightarrow$ *StructuralFeature* $\rightarrow$ *TypedElement* or *isStatic* $\rightarrow$ *Property* $\rightarrow$ *ConnectableElement* $\rightarrow$ *TypedElement*;
- **MultiplicityElement**: by the path *isStatic* $\rightarrow$ *Property* $\rightarrow$ *StructuralFeature* $\rightarrow$ *Feature*;
- **NamedElement**: by the path *isStatic* $\rightarrow$ *Property* $\rightarrow$ *DeploymentTarget* $\rightarrow$ *NamedElement*;
- **ParameterableElement**: by the path *isStatic* $\rightarrow$ *Property* $\rightarrow$ *ConnectableElement* $\rightarrow$ *ParameterableElement* (the example in Figure 4.3).

τ makes the algorithm behave as if the other ancestors dit not exist. Another example will help to explain the parameter: considering the feature *presence of static attribute (PSA)* mapped as before, if τ is 1, it is only possible to achieve the classes in Figure 4.4, and the other classes do not exists in that iteraction, while if τ is 2 the achieved classes are presented in Figure 4.5. Finally, if τ is 3 the achieved classes are presented in Figure 4.6.

4.5 Preparing the user vector

For OntoRec each user profile is a vector in which each component represents a different feature. Considering the mapping table described previously in Table 4.2, each feature in the user profile is identified by 1 if the user is interested on it, or 0 if the user is not interested

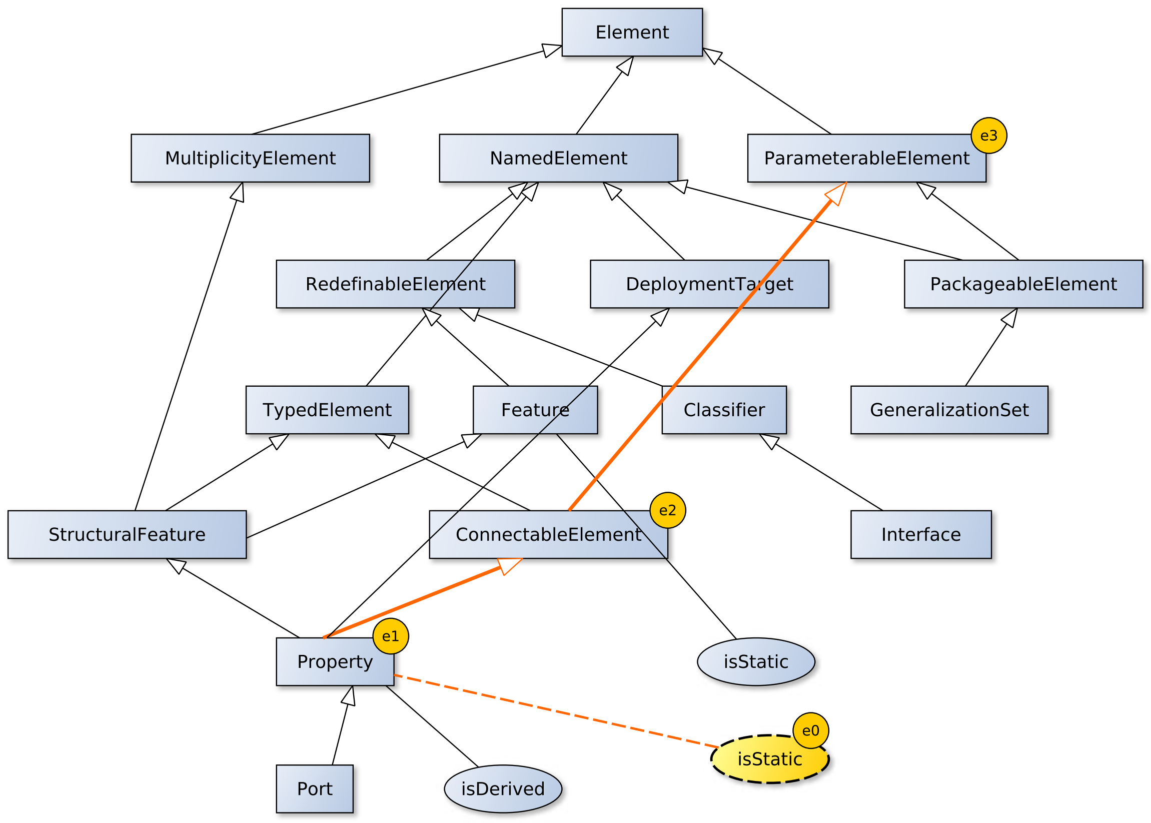


Figure 4.3: Example of path for mapped feature

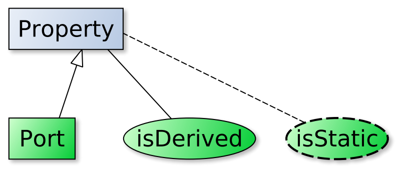


Figure 4.4: Example for $\tau = 1$

in that feature. Table 4.3 contains an example of user profile where the user is interested in the features *presence of interfaces (POI)* and *presence of derived attributes (PDA)*, but not in the others in that table.

Table 4.3: A sample user profile

	POI	**PGS**	**POP**	**PSA**	**PDA**
u	1	0	0	0	1

OntoRec recalculates the user's profile, discovering what we define as "partial interests".

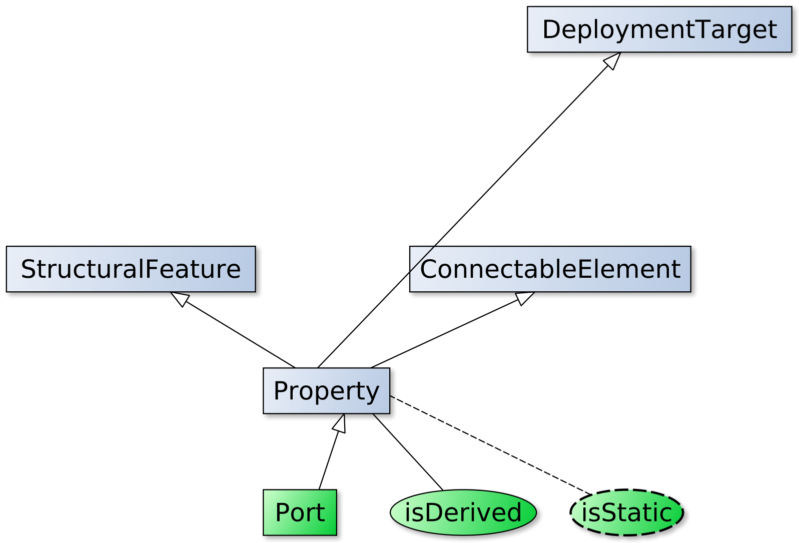


Figure 4.5: Example for $\tau = 2$

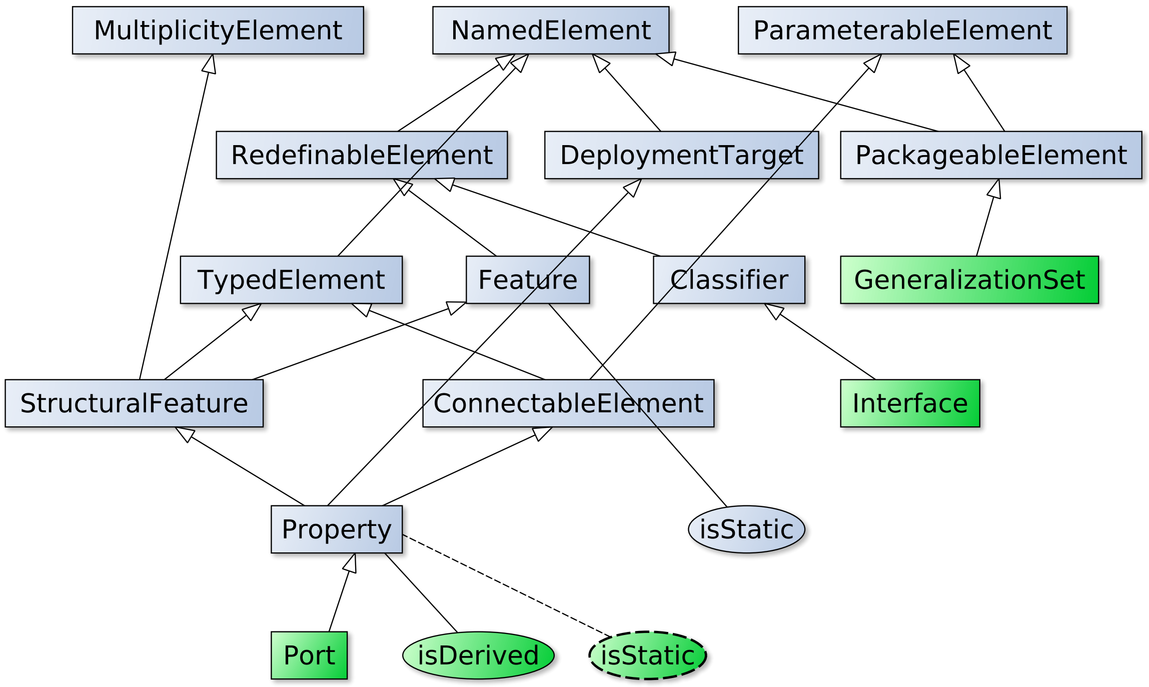


Figure 4.6: Example for $\tau = 3$

To do that, OntoRec employs two approaches: the **Breadth-First Search (BFS)** and the **τ-nth Ancestor**. The Subsections 4.5.3 and 4.5.4 will present each one of them in detail. Each approach executes one time for each feature that was selected by the user, starting from it and calculating weights for all the other features with value 0 based on how far are the other features from the starting one in the ontology. In order to discover what features can be reached from the starting one, they use the τ parameter. By using the example in Table 4.3 both approaches will calculate the weights for the features with value 0 if they are reachable according to the parameter τ, starting in “POI” and by using its mapping to the ontology; after finishing the first iteration, it will calculate again by starting in “PDA”; if a feature contains a nonzero value (and was not selected by the user), the final weight for that feature will be the mean between the old and the new value. Selecting the approach that will be used is a configuration task before running the algorithm.

Considering *presence of interface (POI)* as the starting element, OntoRec will use the ontology to discover some elements that are related. We define as **related** to a given element m the mapped elements that are descendants of the τ-nth ancestors of m, whatever the chosen approach is. Thus, the parameter τ is important.

Independently of the selected approach, it considers the parameters λ (subsection 4.5.1) and υ (subsection 4.5.2) that, as we will see later, changes the amount of covered paths that will be used to calculate the partial interest values.

After discovering the distances between the starting and destiny features, it is time to compute the score of the related feature. We define this score in the Equation 4.1 below:

$$score(m) : 1 - \frac{distance}{pathsSum} \tag{4.1}$$

If m is a feature of interest of the user, the value of this score is always 1. If m already has another previous score value, the new score value is the mean between the current value and the old one.

4.5.1 The λ parameter

To explain this parameter, consider the Figure 4.7. There, it is possible to check that some elements as “ConnectableElement” and “Classifier” just have one direct descendant (all of them are displayed in red boxes in the figure). Suppose that we want a path with length 2 starting from “isStatic” to some ancestor. If $\lambda = 1$, considering the path highlighted in Figure 4.3, “ConnectableElement” will be ignored and the result path will be (“isStatic” $\rightarrow$ “Property” $\rightarrow$ “ParameterableElement”). “Property” is marked as red because its only child is “Port” and the others are properties, but the owner of the property is not ignored: since "isStatic" is a property of "Property", it is not ignored even if $\lambda = 1$. If $\lambda = 0$, "ConnectableElement" will not be ignored and the result path will be ("isStatic" $\rightarrow$ "Property" $\rightarrow$ "ConnectableElement").

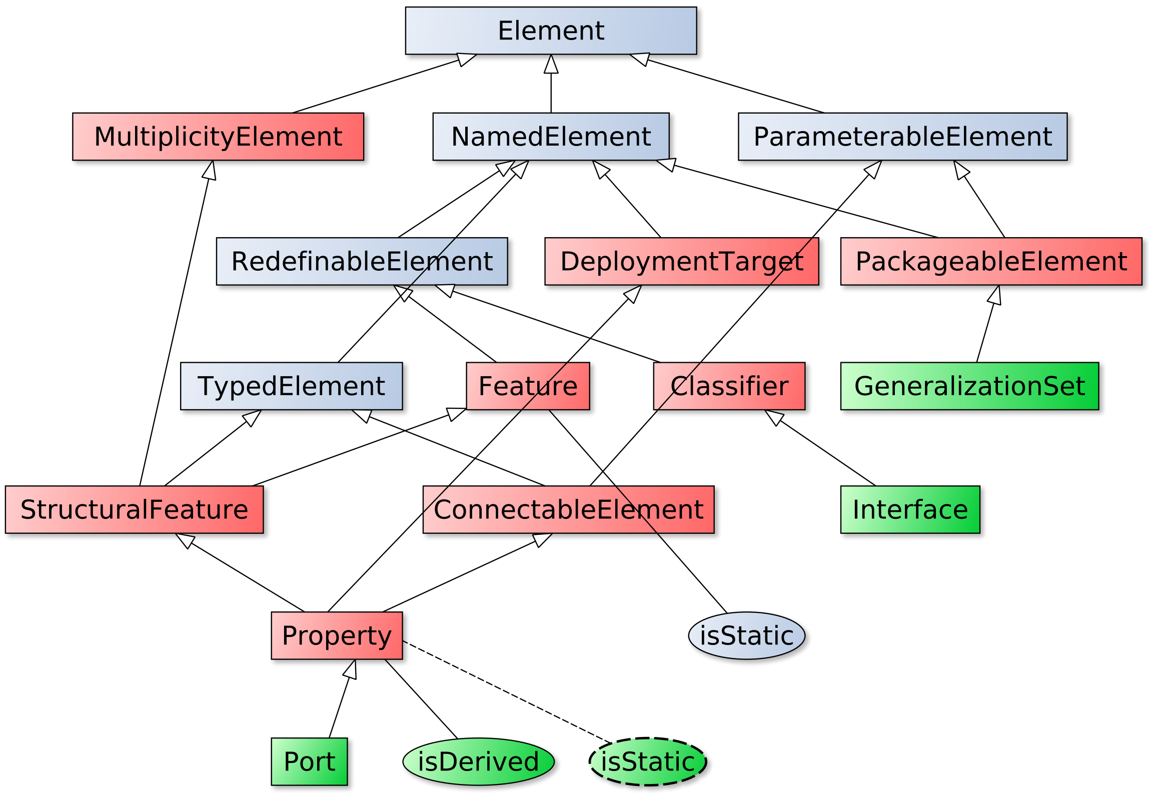


Figure 4.7: Elements to be ignored if $\lambda = 1$

The idea behind the λ parameter is that elements with just one direct specialization does not classify the elements below, and may be ignored. However, the validity of this information depends more of the problem instance than of the algorithm, and the parameter value should be defined for each application of OntoRec.

4.5.2 The υ parameter

This parameter will define if the paths to other mapped nodes will be added to the sum of the paths used to calculate the weights for other nodes. In order to introduce how this parameter works, it will be presented an example where it will be calculated the weight of other features concerning one of the features selected by the user. As instance, it will be considered an user vector with 3 features, "POP", "PDA" and "PSA", and $\tau = 1$. The starting point is in

Figure 4.8, in which the notes represent the mapped features, and the user image represents an user's interest in that feature.

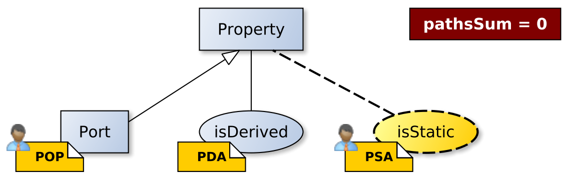


Figure 4.8: A simple example to present the υ parameter

Proceeding with the weight calculations for our example and considering $\upsilon = 0$, no paths will be ignored and we have the Figure 4.9. The variable "pathsSum" is used to count the the sum of visited paths and will be, later on Figure 4.9e, used to calculate the user's interest for features he/she do not selected. If we stop here, the result weight for "PDA" is 0.5.

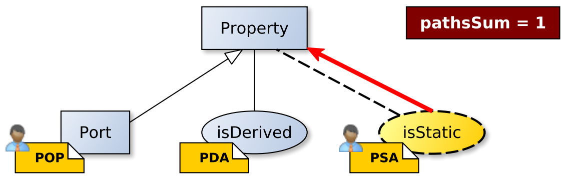
Considering $\upsilon = 1$, the algorithm will identify that "POP" is already chosen by the user and ignores the path described in Figures 4.9a and 4.9b. The result is that fewer paths will be covered and we will have the Figure 4.10, and we will have a different result for "PDA", that is 1.

4.5.3 Breadth-First Search (BFS) approach

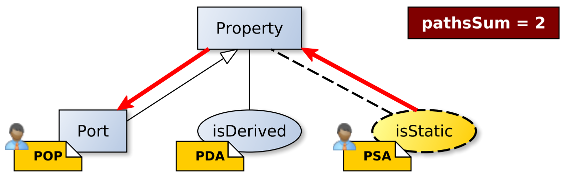
This approach considers the ontology as a graph, using the well known Breadth-First Search (BFS) algorithm [12] to find the lesser distance to other mapped features. This distance will be used later to infer the "amount" of "partial interest" of the user for the other mapped features. Figure 4.11 shows an example of the path used to reach the node "GeneralizationSet" starting from "isStatic" in "Property", considering that the τ-nth class is "Element".

4.5.4 τ-nth Ancestor approach

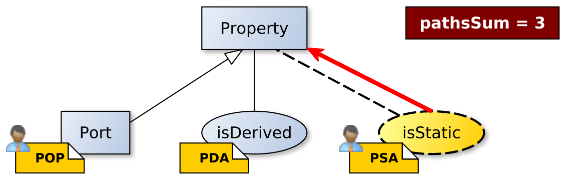
This approach considers the ontology as a graph, walking to the τ-nth ancestor, and complete the path from there to each mapped feature except the starting one by using the BFS algorithm. Figure 4.12 shows an example of the path used to reach the node "GeneralizationSet" starting from "isStatic" in "Property", considering that the τ-nth class is "Element".



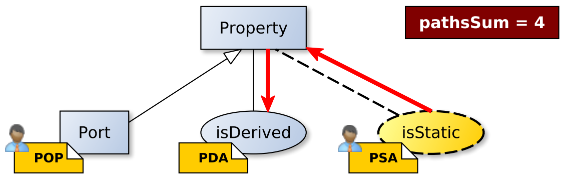
(a) $v = 0$, step 1



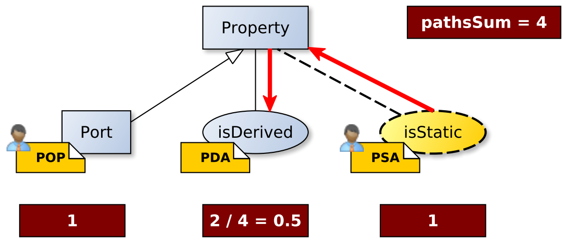
(b) $v = 0$, step 2



(c) $v = 0$, step 3



(d) $v = 0$, step 4



(e) $v = 0$, step 5

Figure 4.9: The steps for $v = 0$

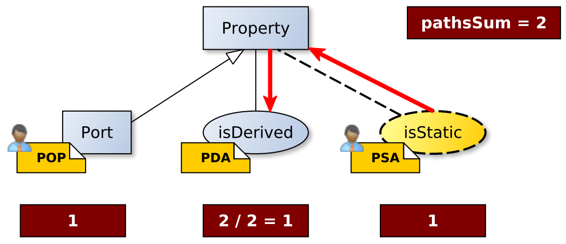


Figure 4.10: The result for $v = 1$

4.6 Calculating the recommended items

After adjusting the profile vector by using the parameters and instructions presented in the previous sections, all we need to do is to recommend items based on the that profile. OntoRec is focused on expanding a profile, thus any vector similarity approach can be used next.

In order to use a vector similarity approach, the items' profiles should have the same format as the user's profiles.

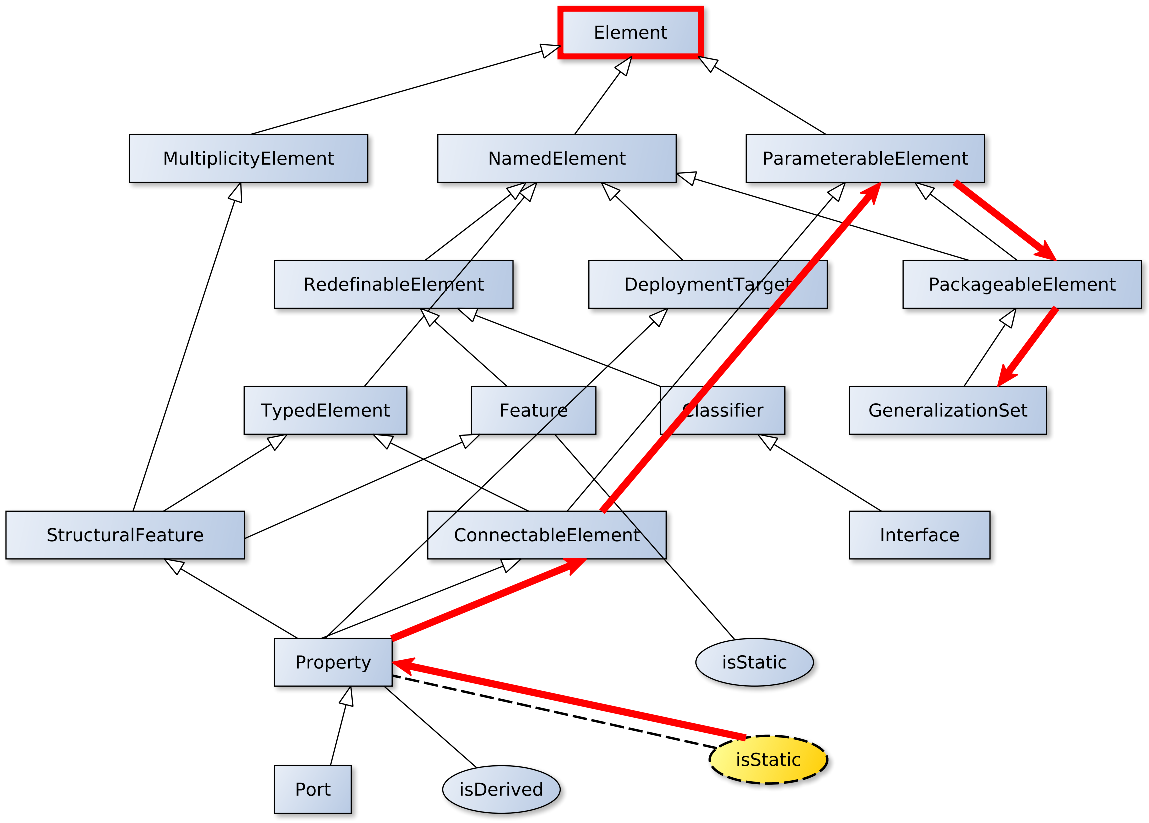


Figure 4.11: The knowledge-based algorithm BFS approach example

Some suggestions of methods commonly used to compute the similarity between vectors are the *cosine similarity* and the *euclidean distance* [30]. One can be free to use one of them or any other known approach.

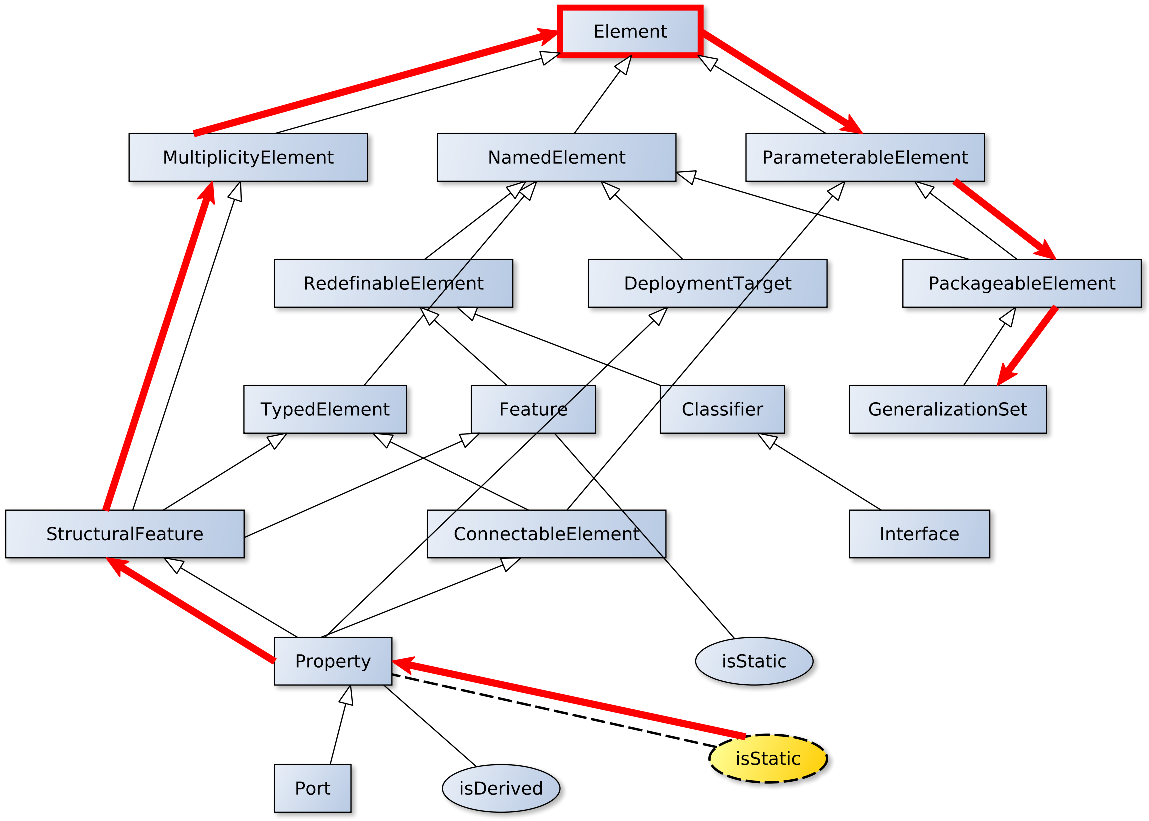


Figure 4.12: The knowledge-based algorithm τ-nth approach example

Chapter 5

Evaluation

In order to evaluate the proposed profiles and recommendation algorithms, we have performed an empirical study that compares them. This study included 51 participants. It was reformulated by Cerqueira [10][1] and was conducted with Computer Science equivalent courses volunteer students and egress. First, Section 5.1 presents the problem statement; the setup is introduced in Section 5.2, covering the context and scope (Subsection 5.2.1), the experiment design (Subsection 5.2.2), the execution (Subsection 5.2.3), the results and analysis (Subsection 5.2.4) and the interpretation of the results (Subsection 5.2.5). A complementary experiment was necessary to explain some results of the first one, and is presented in Section 5.3, covering topics similar to the first one. Finally, the threats to the validity of the results are presented in Section 5.4.

5.1 Problem statement

Finding UML class diagrams can be a challenge. Check them one by one looking for some set of features is a tiring job, and we want to facilitate this task to users by recommending suitable diagrams. In order to explore this problem, we investigated 4 recommender approaches, trying to find their strengths and weaknesses, and if at least one of them is suitable to recommend UML class diagrams. Thus, we propose the following research questions and

[1]A PHD student at Federal University of Campina Grande that is applying the approaches defined in this work to UML sequence diagrams. She suggested changes that allowed a less tiring test process to volunteers, as well as a qualitative set of questions to the end of the experiment.

how we plan to answer them:

- ***RQ1***. *Are the proposed recommender approaches different regarding to their recommendation accuracy?*
- ***RQ2***. *How different are the approaches taken 2 by 2?*

As we will see later in the analysis, the responses for the previous research questions led us to a third research question, that is evaluated in a complementary experiment:

- ***RQ3***. *As we increase the number of selected features to be found, do the precision for all proposed algorithms behave similarly?*

5.2 Setup

We have executed an experiment with human subjects whose design, execution and analysis are presented below.

5.2.1 Context and scope

To answer the research questions we have decided to execute an experiment where the *subjects* represent developers that request UML class diagrams of their interest in a repository. To formalize their interests, the subjects select different UML class diagrams' properties, defining a user profile. Also, the research subjects should have some knowledge about UML class diagrams, and we ensure that by requiring that they have attended at least one UML discipline in a higher education course.

In this experiment we have one *factor*, that comprises the **different recommender approaches** we need to apply to answer the research questions. For the recommender approaches, we have the following 4 *treatments* (each one already described in Section 3.2): (i) random (a baseline for the experiment), (ii) bag-of-words, (iii) items' vector content-based approach and (iv) knowledge-based approach.

To execute the experiment we need to have *objects*, *i.e.* a set of UML class diagrams that contains all the properties that allow us to explore user's interest. We had a total of 325 UML class diagrams, obtained from the following sources[2]:

- by downloading from the *GenMyModel* tool website[3], an on-line UML editor with code generation features. They have made available some public diagrams in the tool repository that could be exported and downloaded in the XMI format;
- by applying reverse engineering of open source systems written in the Java language, randomly chosen from the most downloaded projects at *SourceForge*[4]. These diagrams were manually inspected to remove generated $<< use >>$ relationships (this kind of relationship was generated in large quantities by the adopted tool and could make the diagrams difficult to read by the subjects). The tool used to generate these diagrams is now deprecated and their last information is available at its vendor's website[5]. Also, UML is more expressive and platform independent than Java code, and the quality of the conversion from Java code to UML depends on the tool that is used to do the job. Listing the risks of the conversion demands the job of evaluating each source code and the generated diagrams for each particular Java and UML feature; this is a time consuming task which was disregarded in this work;
- by downloading from the UML repository by the *Chalmers University of Technology* and the *Universiteit Leiden*[6]. The UML class diagrams available in this repository are for UML 1.x, were automatically processed and their XMI files contain errors. We manually inspected one by one, (i) removing diagrams that describe other contexts rather than object oriented software (as database modeling and metamodels representations, for instance), (ii) fixing all XMI features that are significant to the experiment according to the UML images provided along with the diagrams (some diagram images presented generalizations, but the generalization relationship was missing at correspondent XMI, for instance; this problem also happened with other features)[7] and

[2]This experiment's XMI files are available at `http://www.saulotoledo.com.br/masters_thesis/uml_recsys_database.tar.gz`

[3]`http://www.genmymodel.com/`

[4]`http://sourceforge.net/`

[5]`https://www.polarsys.org/topcased`

[6]Available at `http://cse-poros.cse.chalmers.se/`

[7]It is important to know that we do not cleaned all features in these files, but just that ones that we are using at profiles.

(iii) removing too small diagrams (diagrams containing only 1 to 3 classes and no relationships).

When comparing the proposed recommender approaches, we want to know which of them brings better results to the users. In general, metrics such as *precision*, *recall* and *f-measure* are widely used as quality measures by information retrieval and recommender systems communities [43]. *Precision* is defined as the fraction of recovered items that are relevant, whereas *recall* is the fraction of relevant instances that were recovered, and *f-measure* is the harmonic mean between the two previous metrics [43].

We cannot compute *recall* because this metric requires the relevant diagrams for each user to be known beforehand, but we just know the intersection between the relevant and recovered ones (*i.e.* the numerator of Equation 5.1). If we do not know *recall*, we can not compute the *f-measure*. Thus, we just use the metric *precision* (Equation 5.1) to identify the quality of the recommender approach proposals:

$$precision = \frac{|\{relevant\ items\} \cap \{recov.\ items\}|}{|\{recov.\ items\}|} \tag{5.1}$$

After receiving the recommendations, the participants can accept (or not) some of the recommended diagrams. This information is therefore used to calculate the metric *precision* for all approaches.

5.2.2 Experiment design

We have obtained a total of 51 answers through a convenience sample, grouped into different educational backgrounds, distributed in the following way:

- 11 undergraduate subjects;
- 17 graduated subjects;
- 14 postgraduate students subjects;
- 9 postgraduated subjects.

There is no relationship between the groups above and the experience with UML. Thus, we considered them all as equal in the analyses.

In order to answer the research questions, we have one unit of analysis (UA) that aims to answering the research questions by using all the collected data and have **one factor and four treatments**.

For the experiment, the participants performed the following five steps: (i) they filled out the questionnaire presented below reporting their interests regarding the diagrams to be recommended; (ii) after that, the recommendations were generated by the approach by applying the four algorithms described in Chapter 3; (iii) since evaluating all diagrams from the database would be unpractical (there are too many diagrams to be checked there), it was presented the *top*-3 computed diagrams of each approach for the participants; (iv) next, they judged the class diagrams recommended by the system as accepted or not accepted, *i.e.* that satisfies or not their search needs; and (v) finally, they filled out the qualitative set of questions presented in Table 5.1 about the tool and the recommendations.

The aforementioned questionnaire asked to the subject which of the following features he/she was interested to see in UML class diagrams:

- Composite Aggregation;
- Shared Aggregation;
- Association Class;
- Dependency;
- Attribute with Default Value;
- Realized Interface;
- Generalization;
- Interface;
- Derived Attribute;
- Static Operation;
- Port;
- Qualifier;
- Abstract Class;
- Enumeration;
- Navigation Arrow in Association;
- Generalization Set;
- Template Class;
- Static Attribute;
- Abstract Operation.

Cerqueira [10] contributions adopted in this work were the following: (i) the questions style for the user's interests selection step; (ii) the way we run all approaches for all users

Table 5.1: The experiment's final form

Question number	**Question**
1	According to the information you have selected, the recommendations of the diagrams were useful?
2	Which positive features led you to conclude that the recommended diagrams were useful?
3	Which negative features led you to conclude that the recommended diagrams were not useful?
4	Overall, how satisfied were you with the recommendations made by the tool?
5	Would you use this type of search tool again to find UML diagrams according to your information needs?
6	Would you have any suggestions to improve the tool? If you could change something, what would you do differently?

— we also randomize the algorithms order, so the user do not know which algorithm he is evaluating (in fact, the user does not know in any way that he is evaluating more than one recommender approach); (iii) the idea of showing only the *top-3* instead of *top-5*, to decrease the number of diagrams to be evaluated by the volunteers — since we have 4 algorithms, we show 4 pages, each one with the *top-3* for each algorithm; (iv) the qualitative set of questions in Table 5.1, presented at the end of the experiment.

Finally, we formally define this experiment as follows:

- In order to answer **RQ1**, by performing the following hypothesis test:
  H1-0: The precisions for all algorithms are equal
  H1-1: The precisions for all algorithms are different

- In order to answer **RQ2**, considering the combinations 2 by 2 of all approaches, and the approaches a_i and a_j, with $i \neq j$, we do it in 2 ways:

1. By using the *Vargha Delaney* effect size metric [54] to understand the probability of having differences between the approaches a_i and a_j;
2. By performing the following hypothesis test to compare the approaches a_i and a_j:
   H2-0: The precisions for the approaches a_i and a_j are equal
   H2-1: The precisions for the approaches a_i and a_j are different

5.2.3 Execution

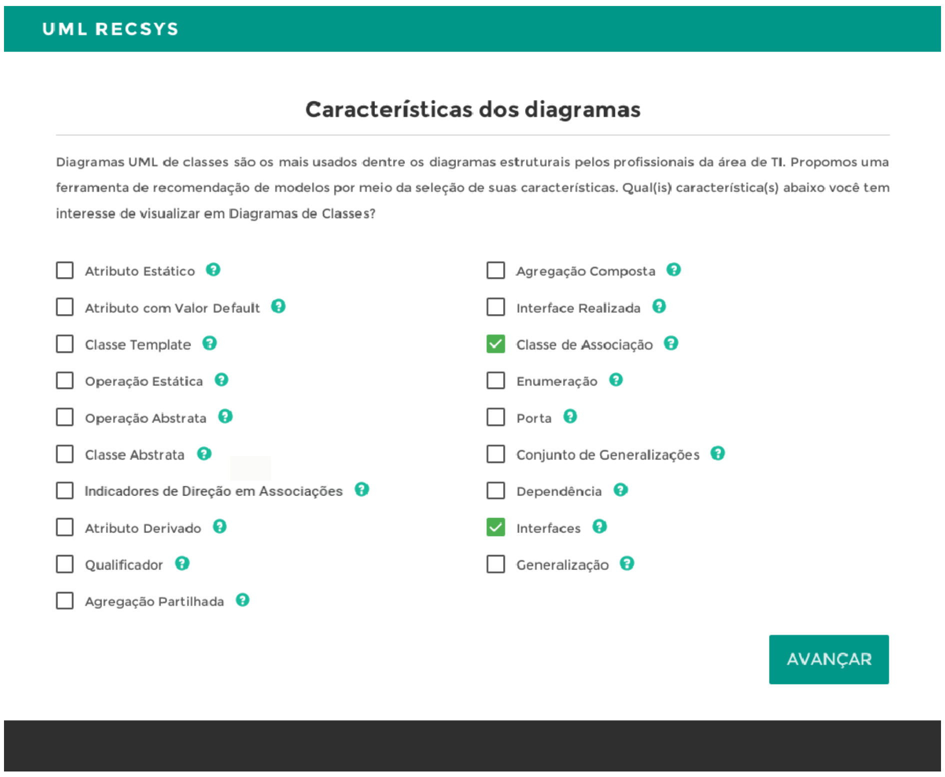
Since there is no survey tool able to execute the proposed algorithms, we have chosen to build a web tool by using the Java language to allow volunteers to execute the proposed experiment. The source code of the tool as well as the software database and instructions on how to replicate the experiment are available at GitHub[8]. The user opens the tool and, after agreeing to participate in the experiment, he/she is directed to a page where he/she can provide his/hers interests (Figure 5.1). After that, he/she can evaluate some recommendations according to the selected features (Figure 5.2). Finally, he/she fills out an open form (Figure 5.3).

This tool (i) uses Apache Lucene[9] in the *bag-of-words* algorithm implementation to create the indexes and conduct the searches, (ii) extract the diagram's profile vector from their corresponding XMI representations and (iii) uses a Java implementation of the OntoRec algorithm that we have developed for our experiments that is also available at GitHub[10].

The users have participated in this experiment in places of their own choice, by running their own personal computers, and by means of an internet access. They also have chosen their own time to start the experiment.

To analyze the experiment results, we have used the GNU R[11] software tool. All the evaluation scripts are available for download and replication at GitHub[12].

[8]`https://github.com/saulotoledo/UMLRecExperiment`
[9]`http://lucene.apache.org/core/`
[10]`https://github.com/saulotoledo/OntoRec`
[11]`http://www.r-project.org/`
[12]`https://github.com/saulotoledo/UMLRecExpAnalysis`


UML RECSYS

Características dos diagramas

Diagramas UML de classes são os mais usados dentre os diagramas estruturais pelos profissionais da área de TI. Propomos uma ferramenta de recomendação de modelos por meio da seleção de suas características. Qual(is) característica(s) abaixo você tem interesse de visualizar em Diagramas de Classes?

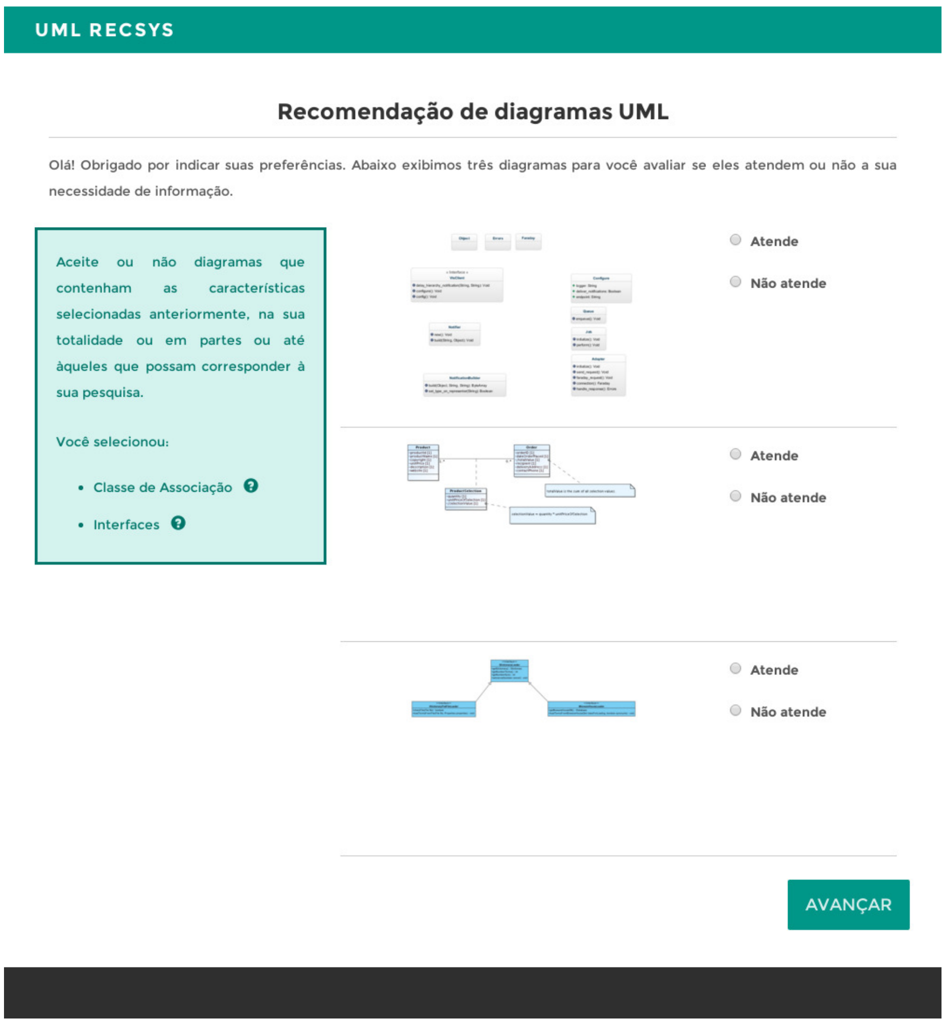
☐ Atributo Estático	☐ Agregação Composta
☐ Atributo com Valor Default	☐ Interface Realizada
☐ Classe Template	✓ Classe de Associação
☐ Operação Estática	☐ Enumeração
☐ Operação Abstrata	☐ Porta
☐ Classe Abstrata	☐ Conjunto de Generalizações
☐ Indicadores de Direção em Associações	☐ Dependência
☐ Atributo Derivado	✓ Interfaces
☐ Qualificador	☐ Generalização
☐ Agregação Partilhada	

Figure 5.1: The screen where the user informs his interests

5.2.4 Results and analysis

As 43 from 51 subjects mixed among all educational background groups marked that already had professional experience, we have decided not to analyze the professional experience because we have very unbalanced groups (only 8 from 51 subjects without professional experience).

Figure 5.4 shows the number of times the subjects selected each feature. Some features are more frequently selected than others; *presence of interfaces (POI)* and *presence of abstract classes* were the most selected, and *presence of ports* and *presence of qualifiers* were the less selected. Each subject also selected about $^{8}/_{19}$ features on average. We evaluated, as the result data of the study, the precision computed by using the diagrams selected as accepted by the test subjects. We applied some normality tests to help to decide what statistical tests we should apply in the data. Among the possibilities, we have chosen the *Shapiro-Wilk* and the *Anderson-Darling* tests for normality, two widely used methods to detect if the data comes from a normal distribution [52]. The null hypothesis for both tests is that the data is normally distributed. If the p-value is less than the chosen α level, we reject the null hypoth-


UML RECSYS

Recomendação de diagramas UML

Olá! Obrigado por indicar suas preferências. Abaixo exibimos três diagramas para você avaliar se eles atendem ou não a sua necessidade de informação.

Aceite ou não diagramas que contenham as características selecionadas anteriormente, na sua totalidade ou em partes ou até àqueles que possam corresponder à sua pesquisa.

Você selecionou:

- Classe de Associação
- Interfaces

Atende

Não atende

Atende

Não atende

Atende

Não atende

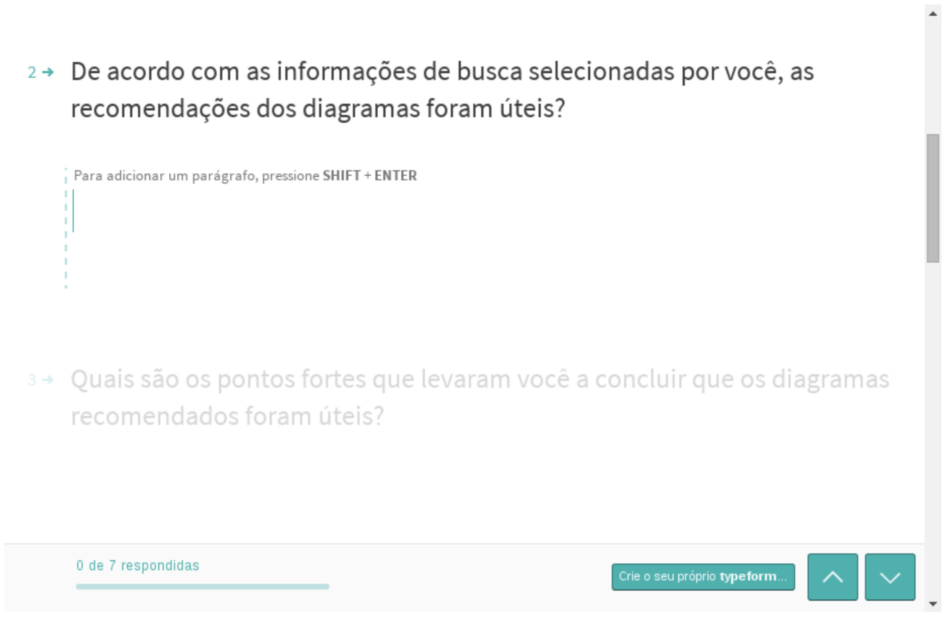
Figure 5.2: Example of screen where the user evaluates the tool's recommendations

esis and there is evidence that the data tested are not from a normally distributed population. Table 5.2 summarizes the testing results, showing us that we cannot detect normality for any significant value of α; W and A are the statistics in which the tests are respectively based on.

Table 5.2: Normality tests for all data

Test name	**Results**
Shapiro-Wilk	$W = 0.84$ p-value = 1.02×10^{-13}
Anderson-Darling	$A = 12.10$ p-value = 3.7×10^{-24}

The results on Table 5.2 are expected because, since we just evaluate the *top-3* diagrams


2→ De acordo com as informações de busca selecionadas por você, as recomendações dos diagramas foram úteis?

Para adicionar um parágrafo, pressione SHIFT + ENTER

3→ Quais são os pontos fortes que levaram você a concluir que os diagramas recomendados foram úteis?

0 de 7 respondidas

Figure 5.3: The final form about the tool and the recommendations

for each algorithm, there are only four possible values for the precision: 0 (no diagram was indicated as relevant), 0.33 ($^1/_3$ diagram was indicated as relevant), 0.66 ($^2/_3$ diagrams were indicated as relevant), 1 (all diagrams were indicated as relevant). Figure 5.5 shows the mean of the precisions for each approach; the means are almost equal, even to the random approach. Figure 5.6 shows their respectives boxplots; their medians are all the same.

Since the data is not normal, *Kruskal-Wallis*, a non-parametric test, is applied [62]. We had a p-value of 0.5344, a very high value for any reasonable value of α, indicating that the test support the idea that the null hypothesis is true, *i.e.* the approaches precisions are equivalent. Again, we cannot detect any significant difference between the approaches, an expected result after visualizing the Figure 5.6.

After that, we have decided to realize a posthoc analysis by testing the pairs of approaches, ignoring the others in an attempt to find some difference. Thus, we have applied the *Vargha Delaney* effect size metric to understand the probability of having differences between the groups [54]. We have also applied the *Wilcoxon* pairwise hypothesis test to compare each group with each other [62] (its null hypothesis is that there is no difference between the two compared approaches). The results are summarized at Appendix C, in Tables C.1 and C.2. Again, we do not have found statistical evidence that there is a better approach: all p-values are too high for any significant value of α in the *Wilcoxon* tests (*i.e.*

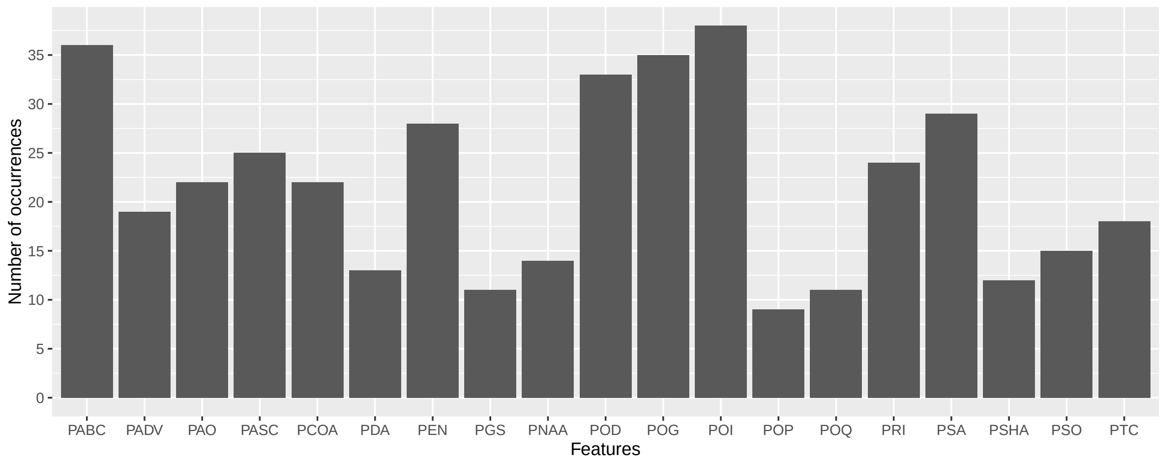


Figure 5.4: The features usage

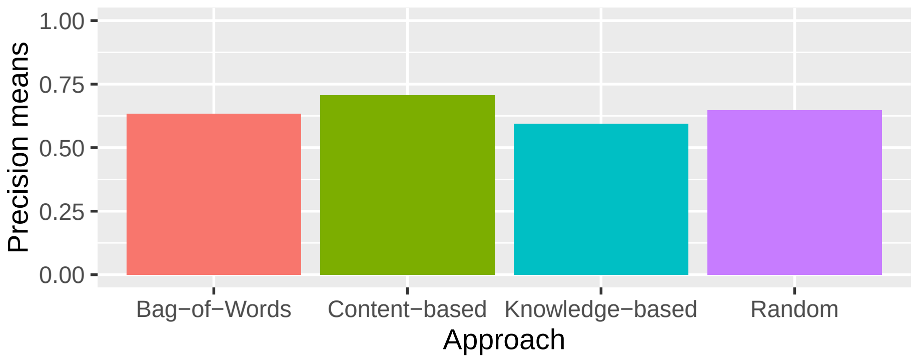


Figure 5.5: Precision by approach for each recommender approach

the null hypothesis is true and both approaches are equivalent), and the effect sizes are all small in the *Vargha Delaney* tests. These results lead us to a next unity of analysis, where we will investigate their causes.

5.2.5 Practical significance (interpretation)

We have achieved no statistical significance in the results, and we have some guesses about the results. First, we believe that we had a low number of diagrams recommendations for each approach (*top-3*), a limitation of the experiment design. A single value chosen (or not chosen) by the user penalizes the precision result by a factor of 25% (remember that we just have four options: from 0 to 3 diagram selections by the user), increasing the precision variance in the results. This is easily verified at previous boxplots (see Figure 5.6). Thus, the Vargha Delaney and Wilcoxon tests, that are also non-parametric (which further decreases

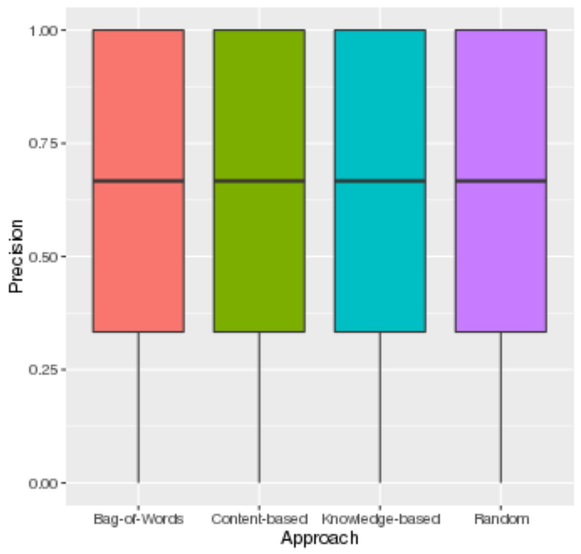


Figure 5.6: Precision by approach boxplots for each recommender approach

the precision of the results) and sensible to the data variance, could not identify differences.

Most subjects (41 of them) filled out the final form. Table 5.3 summarizes the main evaluation of their answers and told us some information: (i) subjects would use a similar tool to search for UML diagrams, and further investigation may be valuable; (ii) subjects have a positive feeling about the recommendations related to have the selected features; (iii) subjects think that the recommended diagrams were too big and containing too many information, making them difficult to understand.

Table 5.3: Some form results

Analysis factor	**Yes**	**No**	**Uncertain / Not applicable**
Subject would use similar tools	33	1	7
Subject is satisfied with results	34	1	6
Diagrams were easy to understand	2	22	17

By analyzing the other form responses, we have noticed that the subjects tried to find the exact features they have selected in the beginning. As the subjects selected a high number of features on average, there is a high chance that the recommended diagrams contain at least one of the selected features. In addition, this increases the chance the most complex and biggest diagrams be recommended. We think that if we had limited to one or two features, it should be easier to identify differences between the approaches in our statistical tests. We also think that this could also reduce the rejection of users for difficult diagrams pointed in Table 5.3, since the system should recommend smaller diagrams with fewer features. Nevertheless these problems are increased by the small size of the experiment database.

The knowledge-based algorithm also increases the chance to recommend items containing similar features to the selected ones if the exact ones were not found. 11 subjects answered that they were trying to find the exact items at recommendations. We understand that the subjects did not have a background context that allowed them to do this kind of evaluation, and they remained searching only by the exact selected items. Thus, this experiment model is not exploring the full potential of the knowledge-based recommender approach, and we do not had the time to execute a proper experiment for this algorithm in the scope of this work.

Although we have no conclusions about the best approach, we think that it is possible to repeat the experiment and get better results by (i) reducing the size of choices the subjects do in the beginning (limiting the number of features to select there to 2, for instance); (ii) increasing the number of recommended diagrams (from top-3 to top-5, for instance); (iii) increasing the number of subjects; and (iv) increasing the database size. In order to check our assumptions, we have proposed an complementary experiment, presented in Section 5.3.

5.3 Additional experiment

The main experiment did not find statistical differences between the proposed recommender approaches, including the random one. In order to understand the causes, we made some assumptions, in which we should:

1. reduce the size of choices the subjects do in the beginning;
2. increase the number of recommended diagrams;
3. increase the number of subjects;
4. increase the database size.

Since the subjects answered that they have selected as accepted the diagrams that had at least one of the selected features, we can simulate this behavior by automatic running the algorithms and checking if the recommended ones contains at least one of the selected features. Therefore, we proposed a simple experiment to check some of the above assumptions by simulating users' input and evaluation.

5.3.1 Context and scope

Our idea was to generate sets of features of different sizes, execute all the approaches for each set and automatically evaluate the relevant ones.

First, in order to generate the user profiles, we counted the number of times each feature was selected by the subjects in the previous experiment (if we just have 2 subjects, one that selected *POI* and *PGS*, and other that selected *POI* and *PDA*, the feature *POI* was selected 2 times, and *PGS* and *PDA* just one time each, for instance). Thus, we discovered that some of them are selected more frequently than others, *i.e.* subjects seem to be more interested in a particular subset, and we selected the 8 of them that were more selected by all subjects, , creating the set $F = \{POI, PABC, POG, POD, PSA, PEN, PASC, PRI\}$. After that, we got all the possible subsets of F, 2^F, *i.e.* all the possible combinations of its elements. Then, we ignored the empty set $\{\} \in 2^F$ and the set F: there is only one possible subset of F with all elements, that is F itself, but we want a greater number of possibilities to compare them with each other. After that, we had a total of $2^8 - 2 = 254$ possible profile candidates. It is important to note that we could to combine all of our 19 features, generating a set of $2^{19} - 2 = 524286$ possibilities, but besides being very costly, 256 is much larger than the subjects responses we had at experiment. Thus, there is no need for now to compute more combinations than that.

We executed all of our approaches for each of the previously computed profile candidates, and checked the adequacy of the recommendations by confirming if the recommended diagrams contained at least one of the searched features. In this scenario, we could even compute *recall*, that is defined as in Equation 5.2 [43], because we can discover the relevant items to the user, but we preferred not to do it for two reasons: (i) we have too much diagrams with the same features at database, leading us to high number of relevant items when comparing to the numerator of the Equation 5.2, which maximum is 3, generating very small values of *recall*; and (ii) we are running this experiment to check some assumptions related to the previous one, that does not have *recall*.

$$recall = \frac{|\{relevant\ items\} \cap \{recov.\ items\}|}{|\{relevant\ items\}|} \tag{5.2}$$

With this experiment, we could not increase our database size, but we could calculate the impact of a different number of choices in the beginning of the experiment, the the number of recommended diagrams and the number of subjects.

5.3.2 Execution

We generated the 254 profile candidates and automated the execution of the recommender portion of the tool built for the previous experiment (see Subsection 5.2.3). Further instructions about how to replicate this execution can be found at GitHub repository for that tool[13].

The acceptance of the recommended items was simulated by using GNU R scripts, also available in a GitHub repository[14] with the experiment results analysis scripts.

5.3.3 Experiment design

We have a total of 254 simulated answers, distributed in 7 groups containing from 8 to 70 elements. This time, besides of the *factor* **different recommender approaches** from the previous experiment, we have a second one that comprises our **different groups**, leading us to another unity of analysis composed by a full factorial experiment design with **two factors, each one with four treatments**. For this analysis, since we could run a full factorial analysis, we have preferred it instead of a dimensionality reduction one.

The four *treatments* are the same as the previous experiment: (i) random; (ii) bag-of-words; (iii) items' vector content-based approach; and (iv) knowledge-based approach. We used the same *objects* of the previous experiment and did not evaluate the final form.

This experiment answer **RQ3** and, in order to do it, we separated our data in groups where its components contain registers whose profile vectors have the same length. For each group we checked the formal tests defined to answer **RQ1** and **RQ2**.

[13]`https://github.com/saulotoledo/UMLRecExperiment`

[14]`https://github.com/saulotoledo/UMLRecExpAnalysis`

5.3.4 Results and analysis

Figure 5.7 shows the precisions for the top-3 recommendations of each recommender approach, considering the quantity of features in the searches. As we increase the number of searched features, the algorithms tend to recommend relevant items more frequently, including the random. In order to explain why even a random selection of items we can recommend items that contain at least one of the searched features, we believe that our database is very small and poorly diversified: it lacks of examples with less features, and each diagram contains too many of them.

One could think that each of the recommender approaches should recommend only diagrams containing all the searched features, but this is unpractical because, as we have 19 features, there are about $2^{19} = 524288$ possible searches, and we should have a larger database to have diagrams for all of them. In this situation, too many searches would return empty.

Figure 5.8 goes a little deeper in our assumptions, it shows the precisions for the top-5 recommendations of each recommender approach, also considering the quantity of features in the searches. Comparing it with the Figure 5.7, we can see that most of the time the medians of the *random* approach for the precision in the top-5 results are lower than for the other approaches. Also, we can see a slight variation in the *bag-of-words* one for profiles lengths 1, 2 and 3 that we do not see in the top-3 results.

Even increasing the number of searches to 70, although there is no statistical difference between the *bag-of-words* and the *content-based* approaches, the *bag-of-words* one recommend more outliers than the *content-based* one.

The *knowledge-based* algorithm increases the chance to recommend similar features, and some of them overlapped the importance of some of the main features. Further studies are necessary to identify in what cases this can lead us to worse results, but we had best results compared with the random approach.

In the same way as in the Section 5.2.4, we checked the normality of the data, and the results are presented at Tables 5.4 and 5.5. As expected, there is no normality for the same reason as in Subsection 5.2.4.

As the analysis in Subsection 5.2.4, we applied *Kruskal-Wallis* because there is no evidence that our data comes from a normal distribution. The results are presented at Tables 5.6 and 5.7 and, this time, we found statistical significance for almost all groups, except for the

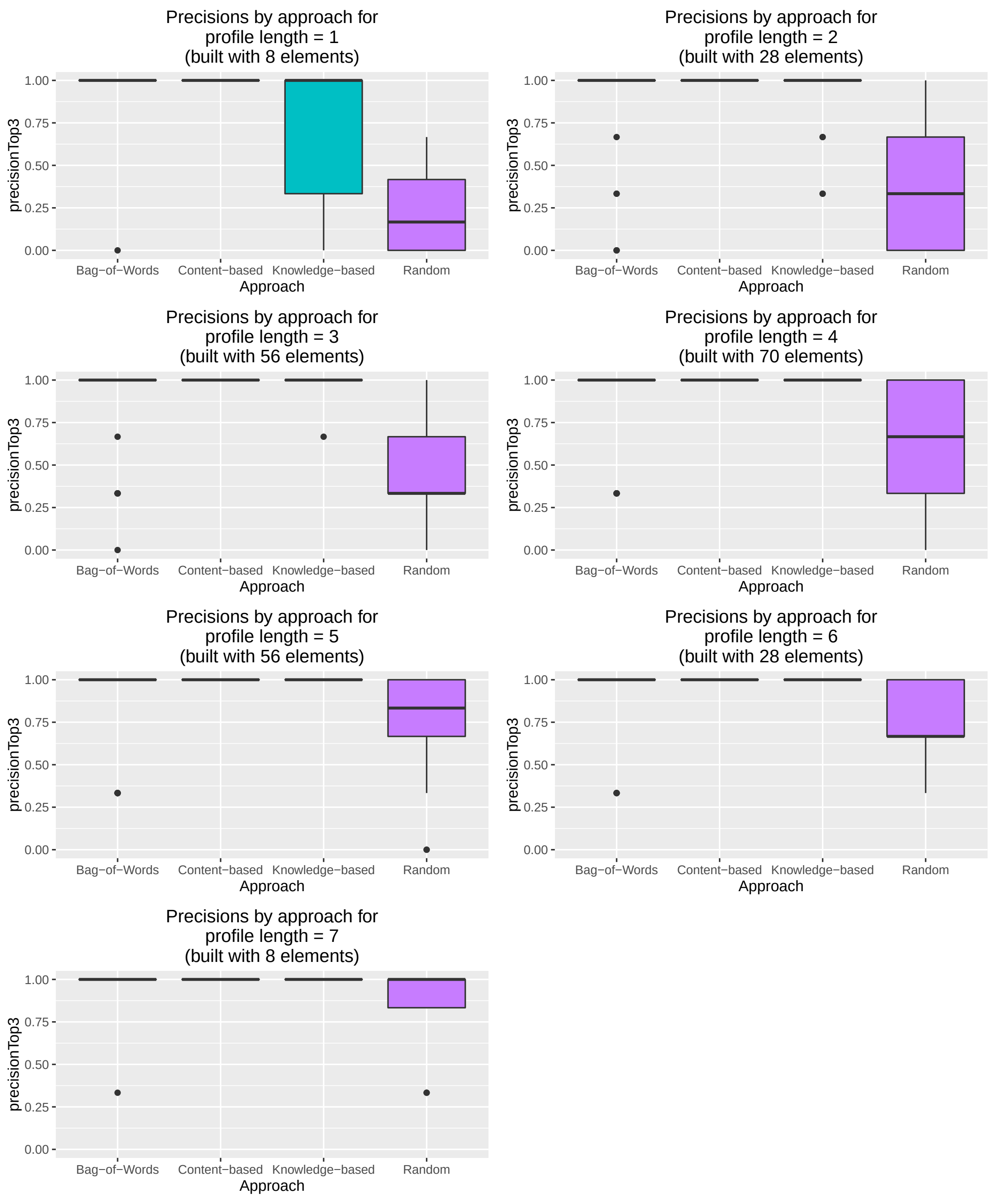


Figure 5.7: Precision of the recommendation approaches for the top-3 items with different amounts of searched features

last one, where all approaches are statistically equal.

We also realized a posthoc analysis by testing every pair of approaches, by using the *Vargha Delaney* effect size metric and the *Wilcoxon* pairwise hypothesis test. Details of the

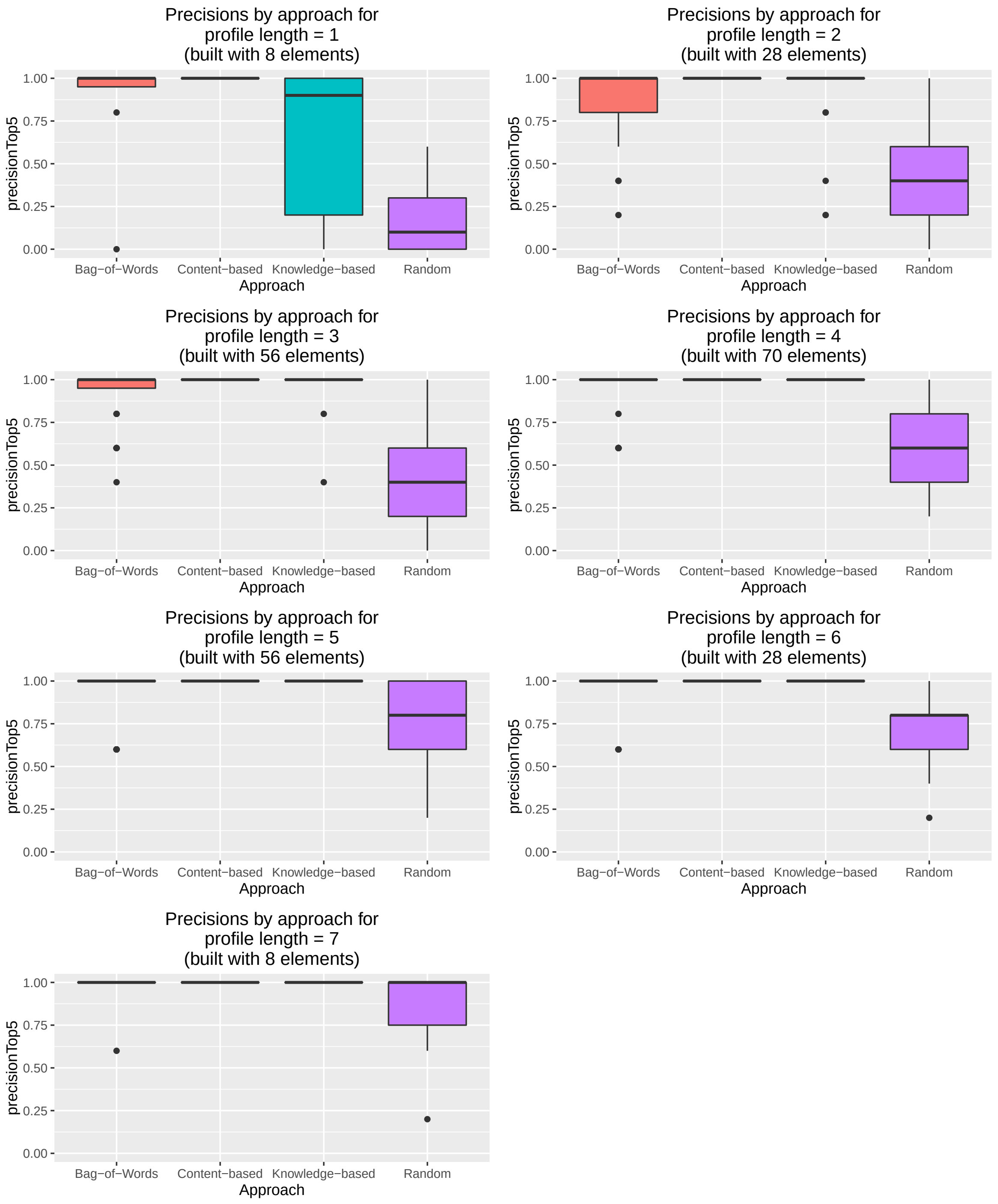


Figure 5.8: Precision of the recommendation approaches for the top-5 items with different amounts of searched features

tests are presented at Appendixes D and E for the top-3 and top-5 analysis, respectively. It is difficult to say that one approach is better than another one, except for the random, that is the worst for all situations.

Table 5.4: Normality tests by user profile length for top-3 results

User profile length	Shapiro-Wilk	Anderson-Darling
1	$W = 0.68$ p-value = 4.26×10^{-7}	$A = 4.7$ p-value = 6.10×10^{-12}
2	$W = 0.61$ p-value = 6.81×10^{-16}	$A = 21.09$ p-value = 3.7×10^{-24}
3	$W = 0.6$ p-value = 1.67×10^{-22}	$A = 42.87$ p-value = 3.7×10^{-24}
4	$W = 0.55$ p-value = 3.08×10^{-26}	$A = 60.68$ p-value = 3.7×10^{-24}
5	$W = 0.48$ p-value = 2.95×10^{-25}	$A = 57.11$ p-value = 3.7×10^{-24}
6	$W = 0.54$ p-value = 3.95×10^{-17}	$A = 24.53$ p-value = 3.7×10^{-24}
7	$W = 0.33$ p-value = 6.08×10^{-11}	$A = 10.59$ p-value = 3.7×10^{-24}

5.3.5 Practical significance (interpretation)

We identified that the more features are selected for searching, the harder it is to identify differences between the approaches in terms of precisions. This partially explains why we had the same results for all approaches in the experiment described in Section 5.2 (when the users selected about 8 features on average), but do not explain why the user medians for the experiment with humans remains 0.66 (see Figure 5.5 for details), and not 1 as we automatically calculated in the automated experiment. We believe that this last behavior is related to human misunderstanding, since the users wrote the diagrams were too big and difficult to understand.

When we increased from top-3 to top-5, the p-value for the test with the user profile length 7 decreased from 0.27 to 0.09, *i.e.* a closer result to find statistical significance that there are difference between the approaches, which is consistent with our assumption that we should try to increase the number of recommended items in future experiments to better

Table 5.5: Normality tests by user profile length for top-5 results

User profile length	**Shapiro-Wilk**	**Anderson-Darling**
1	$W = 0.70$ p-value = 1.02×10^{-6}	$A = 4.21$ p-value = 1.03×10^{-10}
2	$W = 0.68$ p-value = 2.28×10^{-14}	$A = 16.34$ p-value = 3.7×10^{-24}
3	$W = 0.63$ p-value = 9.78×10^{-22}	$A = 38.23$ p-value = 3.7×10^{-24}
4	$W = 0.6$ p-value = 5×10^{-25}	$A = 52.87$ p-value = 3.7×10^{-24}
5	$W = 0.54$ p-value = 6.17×10^{-24}	$A = 49.22$ p-value = 3.7×10^{-24}
6	$W = 0.58$ p-value = 2.64×10^{-16}	$A = 21.85$ p-value = 3.7×10^{-24}
7	$W = 0.38$ p-value = 1.75×10^{-10}	$A = 9.07$ p-value = 1.6×10^{-22}

Table 5.6: Kruskal-Wallis by user profile length for top-3 results

User profile length	**Kruskal-Wallis (p-value)**
1	5.49×10^{-4}
2	1.25×10^{-10}
3	2.2×10^{-16}
4	2.2×10^{-16}
5	3.061×10^{-13}
6	9.009×10^{-11}
7	0.27

evaluate the results. In both Tables 5.6 and 5.7, we just could not find statistical significance for the user profile length 7. The pairwise tests and the effect size evaluations presented in the Appendixes D and E also confirm this assumption: more difference is identified for

Table 5.7: Kruskal-Wallis by user profile length for top-5 results

User profile length	Kruskal-Wallis (p-value)
1	6.13×10^{-4}
2	3.13×10^{-14}
3	2.2×10^{-16}
4	2.2×10^{-16}
5	2.2×10^{-16}
6	1.526×10^{-14}
7	0.09

smaller profile sizes.

The knowledge-based algorithm sometimes has lower precision values than the other approaches, and this is related to the diversification it applies to the user profile in order to discover similar items. The impacts of this diversification should be further evaluated in an experiment that is more appropriate to this algorithm, *i.e.* that consider situations where similar features may be appropriate to the users.

5.4 Validity Threats

We identified some validity threads to this experiment and group them as presented by Wohlin et al. [62].

5.4.1 Conclusion validity threats

The main conclusion validity threat is that we could not identify the UML knowledge level of the users. Thus, we could not create groups and balance them because of the lack of this information, and the participation of the users was voluntary and independent. Nonetheless, since all of our subjects are not specialists in UML, we believe that this validity threat is not meaningful for our study.

Conclusion validity threats are also commonly related to the selected statistical methods. Since we have not normality in our results, we could not apply other but non-parametric tests. This choice decreases our statistical power, but is also was the best choice we had. We also could have applied a pairwise test with Bonferroni correction, but we thought that was more appropriate to choose the pairwise separately to check if there is some difference ignoring the other techniques, even so we could find no statistical significance.

5.4.2 Internal validity threats

A common internal validity threat is a bad design of the testing instrumentation. We have thoroughly tested the tool used in the experiment and the algorithm implementations prior to the experiment execution in order to prevent this kind of validity threat. Also related to this threat was the low number of suggestions to be evaluated by the users, the *top-3*; we prefer recommending only 3 items because we think that the user could have more time to evaluate each diagram without making the experiment too much boring.

We had a low number of subjects to evaluate all the possible combinations of feature selections and compare results. Nonetheless, we got a more than expected number of subjects concerning experiments with humans. Also, the subjects executed the experiment by using their own internet access; thus, we need to trust that they have evaluated the recommendations diligently.

The experiment database was also limited, presenting a low number of UML class diagrams; some features had only a few examples, and several combination of possibilities that subjects may choose in the beginning of the experiment did not exist in the database. In order to mitigate this threat we tried to increase the number of UML models in the database by searching for publicly available databases. The quality of the models is another threat, and we have ensured it by manually checking our database models one by one, removing from our experiment database models that were not suitable to the experiment (models presenting modeling errors, mix of class diagrams with other type of UML diagrams, models with unreadable images and so on).

Another important threat in this category is related to the problem where users tried to find the exact selections while evaluating our proposed knowledge-based algorithm. We have introduced before that the idea of this algorithm is to increase the chance to recommend similar items, but the users searched only for the exact selected items. This is another threat infeasible to mitigate in this experiment and probably requires further studies to answer if this kind of proposal is suitable to this context.

5.4.3 Construct validity threats

We think that the selected approaches were appropriate to our experiment. There are surely several implementations for these approaches, and several others possible choices, but the chosen approaches are feasible to be implemented and representative for the recommender systems' context. The chosen approaches are also well explored in other research contexts besides ours, and we think that could be good choices to be applied.

The set of features to be selected by the subjects is also something to be considered as a possible construct validity threat. We believe that we have mitigated this threat by selecting a set of features that are easy to be identified by users and which generate questions about UML class diagrams.

5.4.4 External validity threats

Unfortunately we had a low number of participants, and we had no enough representativeness to generalize our results. We also had a low number of UML models, and this is another problem while trying to generalize our results. In future experiments we plan to increase the number of subjects and increase our UML class diagrams database. We tried to mitigate these threats by searching for publicly available UML class diagrams databases and for experiment subjects.

Chapter 6

Related Works

In this chapter we review the approaches to recommender systems, including traditional recommender systems, recommender systems for software engineering and recommender systems for UML models.

6.1 Traditional Recommender Systems

The major traditional approaches are usually classified into four categories: collaborative filtering, content-based approaches, knowledge-based approaches and hybrid approaches.

Collaborative filtering models try to predict the utility of items for a particular user based on the items previously rated by multiple users. The main challenge in designing collaborative filtering methods is that the underlying rating matrices are sparse [2]. There have been many collaborative systems developed in the academia and the industry. The Grundy system [47] was the first recommender system, which used stereotypes to build user models based on the amount of information specified for each user. Later on, The Usenet News [25], Ringo [49] and Video Recommender [20] were the first systems to use collaborative filtering to prediction users' preferences. We did not find any publicly available repository of interaction data between users and UML models. Thus, we did not have to make recommendations using collaborative filtering.

The content-based approach to recommendation is based on the content of the items being recommended and has its roots in information retrieval [5]. This approach recommend items similar to those that a user liked in the past [35]. The user' profile can be matched with item descriptions to make recommendations and have some advantages in making recommendations for new items when sufficient rating data are not available for that item [2]. Our approach used this method because it was possible to define the users' and items' profiles and it is reasonable to compare with other retrieval information methods, like *bag-of-words*.

To improve recommendation accuracy, knowledge-based techniques exploit background knowledge about the recommendable items. For example, the system Entrée [53] uses some domain knowledge about cuisines and foods to recommend restaurants to its users. Knowledge-based recommendation systems have been developed for application domains where domain knowledge is readily available in some structured machine-readable form, e.g., as an ontology [36]. For example, the Quickstep and Foxtrot systems [33] use research paper topic ontology to recommend online research articles to the users. To increase the accuracy of recommendations of our approach, we define an ontology for UML models, increasing our reach to recommendations based on the content of the features of class diagrams.

6.2 Recommender Systems for Software Engineering

The idea of building Recommender Systems for Software Engineering artifacts is not new, but most of them are related to source code and some other artifacts [48]. We selected some works that led us to conduct this study.

In the context of Software Engineering, recommendation systems are used to minimize the effort of the developer and help her have faster access to artifacts of interest. Most works aim to increase the reusability, providing ease of maintenance, improving productivity and making suggestions according to the preferences of the developer.

Cubranic et al. [13] exploits recommender systems for bug fixes. Ye and Fischer [63], Lozano et al. [28], and McCarey [31] propose to recommend classes and methods based on the current class being used by the developer. Ankolekar et al. [4] go beyond recommendation methods and indicate artifacts based on the bug fix process, differing from Cubranic et al. [13] by the approach adopted in the bug fix process; they both use information from different developers to correct a defect. Finally, Palma et al. [41] recommend project artifacts, specifically design patterns, taking a different approach than the others because it is more focused on identifying problem contexts.

Other studies deal with comparisons between algorithms of recommendation systems, suggesting improvements, improving the user profile and even new recommendation algorithms, but no one has focused on recommending systems for software engineering, specifically for design.

6.3 Recommender Systems for UML Models

Despite the lack of proposals relating Recommender Systems and UML, we have identified some works that are related to this one. The idea of building Recommender Systems for Software Engineering artifacts is not new, but most of them are related to source code and some other artifacts [48]. We selected some works that led us to conduct this study.

The work of Cerqueira et al. [11] is a study that uses a variation of this works' proposals, the bag-of-words and the vector's content-based approaches, but is focused on sequence diagrams with different users' and items' profiles. Also, Cerqueira [10] documented the experiment definitions that were adopted in our test tool.

Lucrédio et al. [29] investigated a way to use metamodel information to build a search mechanism to models (including UML models). They use Information Retrieval techniques to extract information from models in order to perform their searches. The main differences between our works are: (i) while they built a search engine based on search strings that uses information retrieval techniques, we proposed an user profile and compared some recommender systems approaches; and (ii) while they search for any kind of model based on metamodel information, we plan to recommend only UML class diagrams, based on a set of features also extracted from the metamodel, but specific to this type of diagram.

Finally, Gašević et al. [18] and Kim and Lee [22] proposed ways to transforming UML models into OWL ontologies. We have not used their approaches directly, but they have inspired us about how to do it, by using the metamodel classes as ontology classes.

The aforementioned works are important and represent an emerging area where recommender system techniques are used for searching and recommending software engineering artifacts. This research complements these works by being one of the first research efforts in using recommender systems for recommending UML models.

Chapter 7

Conclusion

In this work we proposed and compared four ways of recommending UML class diagrams: (i) a random baseline, that randomly suggest items to users; (ii) a bag-of-words approach, that identify features in UML diagrams by using search terms; (iii) a content-based recommender approach that uses information extracted in the form of vector profiles from the UML diagrams; and (iv) a newly knowledge-based approach that uses an extended version of the user profile vector indicating other features that might interest the user. We also proposed a profile for users and items to be used by recommender systems.

Two of the proposed approaches, the bag-of-words and the vector's content-based ones, are the application of state-of-the-art algorithms to UML class diagrams. The random approach is a baseline for testing purposes. The knowledge-based approach, the last of them, is a generic new way to identify wishes not explicitly indicated by the users.

In order to compare the algorithms, we conducted two experiments, a first one by using humans, and a second automated experiment to test some assumptions we have proposed to justify the first experiment results. We could not find statistical significance in the first experiment, even when comparing the proposed approaches with the random one. Thus, in the second experiment we have increased the number of executions, evaluated the top-3 and the top-5 results, and controlled the number of selected features that indicates the user's interests in order to evaluate what happens with the results when we increase their number. The second experiment confirms the following assumptions: (i) reducing the size of choices the user have in the beginning of the experiment increases the quality of the recommendations, *i.e.* the proposed approaches are better if the number of searched features at the same time is reduced; (ii) increasing the number of recommended diagrams lead to better results, what is proved by the better p-values and effect size results when recommending the *top-5* instead of the *top-3* diagrams. We also believe that the best results we had in the second experiment are related to the increased number of subjects. It was not possible to increase the database size, but this could also lead to better results.

Another factor that should be better evaluated in future works is the diversification of the recommendation. This is only made by the knowledge-based approach while estabilishing weights to features that were not selected by the users. In the same way, the context of the diagrams and users is also ignored, but could enrich the results and lead to other real world uses of them. These aspects could be evaluated as an upgrade of this research in future works. Thinking about how to represent the context is a future work. An initial suggestion is to create another structure that could store metadata about it and adapt the algorithms. In addition, patterns and domains could be also considered.

Lastly, we did not have historical or collaboration data of users in the context of class diagrams, and we needed to start without them. But we believe that the proposed approaches can be used in these contexts, and that is why we still prefer to classify the approaches as recommender systems approaches.

7.1 Limitations

After running the experiment we found that some features were not detected by the automatic processing of the diagrams. This is related to mistakes in the manual cleanup and fix of the XMI files, but we believe that the errors are not large enough to change our conclusions. The size and the quality of the UML database are also points to be considered. The limited number of subjects and the lack of UML specialists to evaluate the system are also limitations, we just had non-experienced volunteers and we could not discover what users really interested in UML class diagrams think about the study. We also could not evaluate the UML diagrams in some context of use, and we just recommended them in a static way, disregarding their context of use. These observations preclude us to generalize the results to real world contexts.

We had no normality in the data and because of that we needed, most of the time, to use non-parametric statistical tests, which decreases the results statistical power. Also, we could not properly evaluate the knowledge-based approach because the users were not searching for similar features, but only the selected ones. We performed all the evaluation around the precision, and we do not have results for other measures.

7.2 Future work

We present below a list of future work proposals.

- **Apply other metrics in addition to precision:** We just have applied the precision metric, but there are others to be investigated. One example is the *Mean Reciprocal Rank (MRR)*, that evaluates rankings;
- **Describe UML diagrams by using other features:** We have proposed a set of features, but there is still room for others. They could be features that we have not explored (number of parameters, for example), quantities for features (we just have explored the presence or absence of some features) or code metric features (there are a lot of works about features extracted from source code that can be explored together with UML diagrams);
- **Evaluate OntoRec setups:** We have chosen a single setup for OntoRec, but there are other possible configurations for this algorithm that can be evaluated by changing its parameters;
- **Evaluate OntoRec adequacy to the context:** We have identified that users were searching for the exact terms they have selected before. However, OntoRec has the ability to increase the chances to recommend items that have features that are somehow similar to the selected ones. We think that it is possible to so some study that better appreciate this peculiarity;
- **Explore other UML relations at OntoRec:** The experiment ontology just describes the generalization relationship between the elements, but there is a lot of other relationships that can be described in the ontology, as aggregations, compositions and others.

One could explore what happens if we just change the ontology to use these other relationships rather than the generalization, or a combination of them, for instance.

- **Equivalences table to OntoRec:** The idea here is to have another table in the end of OntoRec to calculate equivalences between features. Imagine that we had the features A, B and C and the final calculated vector by OntoRec is $v = (1, 1, 0)$; next, we have defined that features A and C are equivalent, and OntoRec should change the final result to $v = (1, 1, 1)$; or we could define that features A and B are opposed, and the final vector should be $v = (1, 0, 0)$. One could extend this proposal by adding some type of weighting to this relationship: A could be 80% similar to C, and we could have $v = (1, 0.8, 0)$ as the solution. We previously called this feature as "equivalences table" and it would be defined by extracting knowledge from the ontology;

- **Explore OntoRec's parameters relations with the ontology format:** In this proposal one can try to explain the relationship between each parameter at OntoRec and the ontology format. This can help to find the best setup for this algorithm for a given problem easily. Some observations that can be considered in this proposal: what happens with each setup if the provided ontology has an infinite height (this easily happens by using web data, for example); what happens if the ontology looks (or not) like a binary tree; investigate if the τ-nth approach is equivalent to the BFS one for all situations; investigate if it is possible to generalize the results for any ontology that does not use cycles on any domain;

- **Compare OntoRec with other ontologies:** We think that there is a relationship between the algorithm and the quality of the ontology. Maybe one could discover another ontology to describe the same domain, and the results may be different with it;

- **Describe UML class diagrams in another way:** Instead of describing the UML class diagrams by using the proposed features, one could use other ways to do it, like by getting information from OCL [60] rules, MDA [24] transformations, or others;

- **Try other UML diagrams:** One could try the same study we have made with another UML diagrams;

- **Use OntoRec in another domains:** OntoRec can be used to evaluate other domains besides UML, whether from software engineering, like relational databases, whether from completely different domains, like music and movies. It is only necessary to define a valid ontology for the algorithm and fulfill their prerequisites;

- **Repeat this experiment by reducing the current threats to validity:** With the results of this work, one could try to reduce the threats to validity that we had here in a future work;

- **Evaluate user history:** We have not evaluated user history. A future work could consider acquiring and analyzing user's past interactions with the UML diagrams database.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender Systems Handbook*, chapter 7, pages 217–253. Springer US, New York, NY, 2011. doi: 10.1007/978-0-387-85820-3.

[2] Charu C Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016.

[3] Xavier Amatriain, Alejandro Jaimes*, Nuria Oliver, and JosepM. Pujol. Data mining methods for recommender systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 39–71. Springer US, 2011. ISBN 978-0-387-85819-7. doi: 10.1007/978-0-387-85820-3_2.

[4] Anupriya Ankolekar, Katia Sycara, James Herbsleb, Robert Kraut, and Chris Welty. Supporting online problem-solving communities with the semantic web. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 575–584, New York, NY, USA, 2006. ACM. ISBN 1-59593-323-9. doi: 10.1145/1135777.1135862.

[5] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001. URL `http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21`.

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual websearch engine. *Seventh International World-Wide Web Conference (WWW 1998)*, 30:107–117, 1998. URL `http://ilpubs.stanford.edu:8090/361/`.

[8] Robin Burke. Knowledge-based recommender systems. In *Encyclopedia of Library and Information Systems*, page 2000. Marcel Dekker, 2000.

[9] Robin Burke. Hybrid web recommender systems. In *The Adaptive Web: Methods and Strategies of Web Personalization., Lecture Notes in Computer Science*, volume 4321, page 377–408. Springer, Berlin-Heidelberg, 2007.

[10] T. Cerqueira. Estudo comparativo entre entre duas abordagens de sistemas de recomendação para diagramas de sequência uml. Relatório de projeto de tese de doutorado (UFCG). 2016.

[11] Thaciana Cerqueira, Leandro Marinho, and Franklin Ramalho. A content-based approach for recommending UML Sequence Diagrams. 2016. doi: 10.18293/SEKE-147.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter Elementary Graph Algorithms. The MIT Press, third edition, 2009.

[13] D. Cubranic, G.C. Murphy, J. Singer, and K.S. Booth. Hipikat: a project memory for software development. *Software Engineering, IEEE Transactions on*, 31(6):446–465, June 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.71.

[14] Brian Dobing and Jeffrey Parsons. How UML is used. *Commun. ACM*, 49(5):109–113, May 2006. ISSN 0001-0782. doi: 10.1145/1125944.1125949.

[15] Brian Dobing and Jeffrey Parsons. Dimensions of UML diagram use: a survey of practitioners. *Journal of Database Management*, 19, 2008. ISSN 10638016.

[16] Richard Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7(7), July 1964. ISSN 0001-0782. doi: 10.1145/364520.364540.

[17] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003. ISBN 0321193687.

[18] Dragan Gašević, Dragan Djurić, and Vladan Devedžić. Mda-based automatic owl ontology development. *International Journal on Software Tools for Technology Transfer*, 9(2):103–117, 2006. ISSN 1433-2787. doi: 10.1007/s10009-006-0002-1. URL `http://dx.doi.org/10.1007/s10009-006-0002-1`.

[19] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–71, December 1992.

[20] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

[21] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, New York, NY, USA, 1st edition, 2010. ISBN 0521493366, 9780521493369.

[22] I. W. Kim and K. H. Lee. A model-driven approach for describing semantic web services: From uml to owl-s. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(6):637–646, Nov 2009. ISSN 1094-6977. doi: 10.1109/TSMCC.2009.2023798.

[23] Rajiv Kishore and Raj Sharman. Computational ontologies and information systems i: Foundations. *Communications of the Association for Information Systems*, 14(8), 2004. URL `http://aisel.aisnet.org/cais/vol14/iss1/8`.

[24] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 032119442X.

[25] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[26] Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Machine Learning Conference (ML95)*, Lake Tahoe, CA, 1995.

[27] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition).* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131489062.

[28] A Lozano, A Kellens, and K. Mens. Mendel: Source code recommendation based on a genetic metaphor. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 384–387, Nov 2011. doi: 10.1109/ASE.2011.6100078.

[29] Daniel Lucrédio, Renata P. de M. Fortes, and Jon Whittle. Moogle: a metamodel-based model search engine. *Software & Systems Modeling*, 11(2):183–208, 2012. ISSN 1619-1366. doi: 10.1007/s10270-010-0167-7. URL `http://dx.doi.org/10.1007/s10270-010-0167-7`.

[30] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, England, 2009. URL `http://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf`.

[31] F. McCarey. Agile software reuse recommender. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 652–, May 2005. doi: 10.1109/ICSE.2005.1553632.

[32] Donald Metzler. Beyond bags of words: Effectively modeling dependence and features in information retrieval. *SIGIR Forum*, 42(1):77–77, June 2008. ISSN 0163-5840. doi: 10.1145/1394251.1394271. URL `http://doi.acm.org/10.1145/1394251.1394271`.

[33] Stuart E Middleton, Nigel R Shadbolt, and David C De Roure. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22 (1):54–88, 2004.

[34] Russ Miles and Kim Hamilton. *Learning UML 2.0.* O'Reilly Media, Inc., 2006. ISBN 0596009828.

[35] Raymond J Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[36] M A S N Nunes. Towards to Psychological-based Recommenders Systems: A survey on Recommender Systems. *Scientia Plena*, 6(8):1028, 2010. URL `http://www.scientiaplena.org.br/ojs/index.php/sp/article/viewFile/119/48`.

[37] OMG. Meta object facility (MOF) core specification, version 2, 2006. URL `http://www.omg.org/spec/MOF/2.0/PDF/`.

[38] OMG. XML metadata interchange (XMI) specification, version 2.1, 2009. URL `http://www.omg.org/spec/XMI/2.1/PDF/`.

[39] OMG. Unified modeling language: Infrastructure, version 2.4.1, 2011. URL `http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/`.

[40] OMG. Unified modeling language: Superstructure, version 2.4.1, 2011. URL `http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/`.

[41] F. Palma, H. Farzin, Y. Gueheneuc, and N. Moha. Recommendation system for design patterns in software development: An dpr overview. In *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, pages 1–5, June 2012. doi: 10.1109/RSSE.2012.6233399.

[42] Roberto Poli, Michael Healy, and Achilles Kameas. *Theory and Applications of Ontology: Computer Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 9048188466, 9789048188468.

[43] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997. ISSN 0001-0782. doi: 10.1145/245108.245121. URL `http://doi.acm.org/10.1145/245108.245121`.

[44] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings*

of the 1994 ACM Conference on Computer Supported Cooperative Work, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM. ISBN 0-89791-689-1. doi: 10.1145/192844.192905. URL `http://doi.acm.org/10.1145/192844.192905`.

[45] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 0387858199, 9780387858197.

[46] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook*, chapter 1, pages 1–35. Springer US, New York, NY, 2011. doi: 10.1007/978-0-387-85820-3.

[47] Elaine Rich. User modeling via stereotypes. *Cognitive science*, 3(4):329–354, 1979.

[48] M.P. Robillard, R.J. Walker, and T. Zimmermann. Recommendation systems for software engineering. *Software, IEEE*, 27(4):80–86, 2010. ISSN 0740-7459. doi: 10.1109/MS.2009.161. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5235134`.

[49] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.

[50] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. ISBN 0470025700.

[51] William Swartout and Austin Tate. Guest editors' introduction: Ontologies. *IEEE Intelligent Systems*, 14(1):18–19, January 1999. ISSN 1541-1672. doi: 10.1109/MIS.1999.747901. URL `http://dx.doi.org/10.1109/MIS.1999.747901`.

[52] Henry C. Thode. *Testing for Normality*. Marcel Dekker, New York, NY, USA, 2002.

[53] Shari Trewin. Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):180, 2000.

[54] A. Vargha and H. D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal on Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[55] W3C. Owl web ontology language guide, February 2004. URL `https://www.w3.org/TR/2004/REC-owl-guide-20040210/`.

[56] W3C. Owl 2 web ontology language document overview (second edition), December 2012. URL `https://www.w3.org/TR/owl2-overview/`.

[57] W3C. Resource description framework (rdf), February 2014. URL `https://www.w3.org/RDF/`.

[58] W3C. Rdf 1.1 concepts and abstract syntax, February 2014. URL `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.

[59] W3C. Rdf 1.1 xml syntax, February 2014. URL `https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/`.

[60] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition, 2003. ISBN 0321179366.

[61] David A. Watt. *Programming Language Design Concepts*. John Wiley & Sons, 2004. ISBN 0470853204.

[62] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. ISBN 0-7923-8682-5.

[63] Yunwen Ye and Gerhard Fischer. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 513–523, New York, NY, USA, 2002. ACM. ISBN 1-58113-472-X. doi: 10.1145/581339.581402. URL `http://doi.acm.org/10.1145/581339.581402`.

Appendix A

OntoRec formalizations

This appendix presents some formalizations for the OntoRec algorithm (Chapter 4). Section A.1 presents the information model, Section A.2 presents the formal definition for the parameter τ and Section A.3 presents some other details in the preparation of the user vectors.

A.1 Information Model

The main OntoRec concepts are formally described next:

- **Users set** $U = \{u_1, u_2, \cdots, u_n\}$: Contains all the system users;
- **Items set** $I = \{i_1, i_2, \cdots, i_m\}$: Contains all the system items (the elements to be recommended);
- **Feature set** $F = \{f_1, f_2, \cdots, f_g\}$: Contains all features considered in the problem domain;
- **System classes set** $C = \{c_1, c_2, \cdots, c_h\}$: Contains all the concepts that can be used to represent the system. Each one of these concepts can be related to other concepts at this set as specialization or generalization of the other. We call each one of these concepts as a "class";
- **Ontology** O **for the classes** C: We model an ontology O by using the classes set C previously defined. For each class $c_k \in C$ there is a set $A_l = \{a_1, a_2, \cdots, a_m\}, m \geq 0$

of attributes;

- **All attributes' set** $B = \bigcup_{c=1}^{|C|} A_c$: Contains all the attributes for all classes of O, where C is the set of ontology classes and A_c is defined as previously presented;

- **Mappable elements set** $M = M_\alpha \cup M_\beta$: Contains all elements that can be mapped from features, *i.e.* classes and attributes. $M_\alpha \in C$ is the subset of classes that are mapped by the system, and $M_\beta \in B$ is the subset of attributes that are mapped by the same system. M_α and M_β will be formally defined next;

- **Mapping discovery function** $map : F \to M$: Function that maps the feature set F to elements in the set M, according to a previous defined mapping table related to the problem instance;

- **Mapped classes set** $M_\alpha = \{c_j \in C : c_j \in map(f_k)\forall(f_k \in F)\}$: Contains all the classes mapped by the system;

- **Class attributes function** $getAttrs : C \to 2^B$: Returns all the attributes that are owned by a class, directly or by inheritance[1];

- **Mapped attributes set** $M_\beta = \{(c_o \in C, b_p \in B) : b_p \in getAttrs(c_o), b_o \in map(f_p)\forall(f_p \in F)\}$: Contains all the attributes mapped by the system. All the attributes mappings are linked to a class, since each attribute can be mapped to the owner class or its descendants;

- **Owner's definition function** $owner : B \to C$: Function that, given an attribute $b_q \in B$, returns the class $c_r \in C$ for which that attribute is mapped;

- **Ancestry's definition function** $\omega_{ancestors} : (\omega, M, \lambda) \to 2^C$: Function that, considering the ontology O, returns the ancestors at ω level of ancestry for a reference element $m \in M$. $\lambda \in (0, 1)$ and, if is 1, the function will ignore the existence of levels that contains just one direct descendant, otherwise these levels will not be ignored (this is a parameter that will be better explained in future sections). To exemplify this behavior, $\omega = 1$ returns the immediate parents, and $\omega = 2$ will return the grandparents, for example;

[1] 2^B is the power set of B.

- **Mapped descendant's definition function** $\mu_{desc} : (C, \upsilon) \rightarrow 2^M$: Function that, considering the ontology O, returns the descendant classes for a given class $c \in C$ that are mapped to features. $\upsilon \in (0, 1)$ is a parameter that controls if classes in M_α or attributes for any element in M_β are returned by this function (the value 1 enables this behavior, the value 0 disables it). The parameter υ will be better explained in future sections.

These search strings were adapted to the content of the database's files in this work and can include non-standard XMI code and compatibility with UML 1.x and 2.x.

A.2 τ formal definition

Let $F_{interest} \subseteq F$ be the subset of features of interest to the user. We define τ as the **influence range factor** that a feature $f_w \in F_{interest}$ can have over F.

For each $f \in F$ there is a correspondent mapped element on the ontology O that is represented by an element $m \in M$. This result allow us to consider that τ defines the set of paths $P = \{p : \forall_{e_{t_\xi} \in \omega_{ancestors}(\xi, e_{t_0}, \lambda)} p = (e_{t_0}, e_{t_1}, \cdots, e_{t_\xi}) \wedge \forall_{y \in \mathbb{N}} e_{t_{y+1}} \in \omega_{ancestors}(1, e_{t_y}, \lambda)\}$, for any value of λ, considering that $\forall_{m_x \in M} e_{x_0} = m_x$. If $e_{x_0} \in M_\beta$, *i.e.* is an attribute, $e_{x_1} \in M_\alpha$ and is a class. Figure 4.3 contains an example of path for $\tau = 3$.

A.3 User vector preparation details

For OntoRec, each user u_t has a vector v_t with length $|F|$. Each position of this vector represents a feature $f \in F$, and is related to the information presented in ontology O by means of the mapping table previously defined at Section 4.3. If the user is interested in a particular feature, it will have initially the value 1 at the vector. Each one of the other features (that the user has not interest) will receive the value 0.

In order to get that ancestors, we use the $\omega_{ancestors}$ function defined in Section A.1. If $\tau = 2$, $\omega_{ancestors}$ will return "StructuralFeature", "DeploymentTarget" and "ConnectableElement" for "PSA" (that is connected to the attribute "isStatic" on "Property") in the Figure 4.1, for instance, and all the features mapped to their children are related to

"PSA" for this value of τ. The larger is the τ parameter, the higher are the chances to have a greater number of related elements.

Also, the BFS and the τ-nth approaches will compute a weight for all reached mapped feature by running the Source Code A.1; In that code appear a function that we have not defined yet, the "getDistances()", that calculates the distances from a feature to all other mapped and reachable ones; these distances will be used to calculate the final weight for all other features. It is different for each approach, and their respective implementations are presented for each one in the Sections A.3.1 and A.3.2.

Source Code A.1: Common code for both algorithm approaches

```
public Map getFeaturesWeight(Set $F_{interest}$, Set $M_\alpha$, Set $M_\beta$, Integer $\tau$,
    Boolean $\lambda$, Boolean $\upsilon$) {
    Map result;

    for ($f \in F_{interest}$) {
        Integer pathsSum = 0;
        Map distancesToMappedElems = getDistances($map(f)$, $F_{interest}$, $\tau$, $\lambda$, $\upsilon$)
            ;
        for (element : distancesToMappedElems.keySet()) {
            pathsSum += distancesToMappedElems[element];
        }
        for (element : distancesToMappedElems.keySet()) {
            Integer distance = distancesToMappedElems[element];
            if (!result.hasKey(element)) {
                result[element] = 1 - (distance / pathsSum);
            } else {
                Integer currentResult = 1 - (distance / pathsSum);
                result[element] = (result[element] + currentResult) / 2;
            }
        }
    }
    return result;
}
```

The vector expansion is the most expensive part of the knowledge-based approach. It is related to the number f of features in the proposal. In the worst case, τ is the height of the tree and all features will be explored. As will be presented in the next subsections, OntoRec uses BFS for searching in the ontology graph. Considering $|V|$ the number of vertices and $|E|$ the number of edges in the graph, the cost of the BFS is $O(|V| + |E|)$ [12]. The Source Code A.1 does a loop for each feature in line 4, but the function "getDistances()" in line 6 will also iterate for each feature. Thus, the cost for the entire approch is about $\theta(f^2)$, since the cost of the BFS is lower than that.

A.3.1 The Breadth-First Search (BFS) approach source code

The Source Code A.2 is an implementation of the function "getDistances()" in the Source Code A.1 and implements the behavior described in Subsection 4.5.3. The "BFSDistanceTo(m, λ)" function computes the distance by using the BFS algorithm, considering the λ parameter's behavior as described in Subsection 4.5.1.

Source Code A.2: BFS approach code snippet

```
public Map getDistances(Element referenceElement, Set $F_{interest}$, Integer $\tau$,
    Boolean $\lambda$, Boolean $\upsilon$) {
    Set result = {};
    Set reachableMappedElems = {};
    Set ancestors = $\omega_{ancestors}(\tau$, referenceElement, $\lambda)$

    for ($m_{ancestor} \in$ ancestors) {
        reachableMappedElems = reachableMappedElems $\cup$ $\mu_{desc}(m_{ancestor}, \upsilon)$;
    }

    for ($m \in$ reachableMappedElems) {
        if ($\upsilon$ || (!$\upsilon$ && !($m \in F_{interest}$))) {
            Integer distance = referenceElement.BFSDistanceTo($m$, $\lambda$);
            result[$m$] = distance;
        }
    }
    return result;
}
```

A.3.2 τ-nth Ancestor approach source code

The Source Code A.2 is an implementation of the function “getDistances()” in the Source Code A.1 and implements the behavior described in Subsection 4.5.4. The “$BFSDistanceTo(m, \lambda)$” function computes the distance by using the BFS algorithm, considering the λ parameter’s behavior as described in the Subsection 4.5.1. The λ parameter is also considered at $\omega_{ancestors}$ function as presented in the Section A.1.

The cost of this approach is greater than the BFS approach, once it needs to achieve the τ-*nth* ancestors. Nevertheless, in the worst case, the previously presented cost $\theta(f^2)$ is higher than the cost for that and this approach differ from the BFS one by a constant factor.

Source Code A.3: τ-nth approach code snippet

```
public Map getDistances(Element referenceElement, Set $F_{interest}$, Integer $\tau$,
    Boolean $\lambda$, Boolean $\upsilon$) {
    Set result = {};
    Set reachableMappedElems = {};
    Set ancestors = $\omega_{ancestors}(\tau$, referenceElement, $\lambda)$

    for ($m_{ancestor} \in$ ancestors) {
        reachableMappedElems = reachableMappedElems $\cup$ $\mu_{desc}(m_{ancestor}, \upsilon)$;
    }

    for ($m \in$ reachableMappedElems) {
        if ($\upsilon$ || (!$\upsilon$ && !($m \in F_{interest}$))) {
            Integer lesserDistance = $\infty$;
            for ($m_{ancestor} \in$ ancestors) {
                Integer distance = $\tau$ + $m_{ancestor}$.BFSDistanceTo($m$, $\lambda$);
                if (distance < lesserDistance) {
                    lesserDistance = distance;
                }
            }
            result[$m$] = lesserDistance;
        }
    }
    return result;
}
```

Appendix B

Bag-of-words algorithm search strings

The following tables contain the search strings for each user selected feature for the *bag-of-words* algorithm, presented at Subsection 3.2.2.

Table B.1: Search strings for the bag-of-words approach (1)

User selected feature	**Search string**
PSA	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (ownedAttribute || "UML:Attribute") + ("isStatic=true" || "ownerScope=classifier") + (packagedElement || "UML:Model")
PADV	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (ownedAttribute || "UML:Attribute") + (defaultValue || Attribute.initialValue) + (packagedElement || "UML:Model")
PASC	("uml:AssociationClass" || "UML:AssociationClass") + (packagedElement || "UML:Model")
PRI	("uml:Interface" || "UML:Interface" || name="interface") + (interfaceRealization || "Dependency.supplier") + Interface + (packagedElement || "UML:Model")

Table B.2: Search strings for the bag-of-words approach (2)

User selected feature	**Search string**
PABC	("uml:Class" || "UML:Class") + "isAbstract=true" + (packagedElement || "UML:Model")
PTC	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (ownedTemplateSignature || TemplateParameter) + (packagedElement || "UML:Model")
PEN	("uml:Enumeration" || UMLEnumeration) + (packagedElement || "UML:Model")
POP	("uml:Class" || "UML:Class") + ownedAttribute + "uml:Port" + (packagedElement || "UML:Model")
PSO	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (ownedOperation || "UML:Operation") + ("isStatic=true" || "ownerScope=classifier") + (packagedElement || "UML:Model")
PAO	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (ownedOperation || "UML:Operation") + "isAbstract=true" + (packagedElement || "UML:Model")
PNAA	("uml:Association" || "UML:Association") + (navigableOwnedEnd || "isNavigable=true") + (packagedElement || "UML:Model")
PDA	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + ownedAttribute + "isDerived=true" + (packagedElement || "UML:Model")

Table B.3: Search strings for the bag-of-words approach (3)

User selected feature	Search string
POQ	(("uml:Class" || "UML:Class") || ("uml:Interface" || "UML:Interface" || name="interface")) + (qualifier || "AssociationEnd.qualifier") + (packagedElement || "UML:Model")
PSHA	("uml:Association" || "UML:Association") + (ownedEnd || AssociationEnd) + ("aggregation=shared" || "aggregation=aggregate") + (packagedElement || "UML:Model")
PCOA	("uml:Association" || "UML:Association") + (ownedEnd || AssociationEnd) + "aggregation=composite" + (packagedElement || "UML:Model")
PGS	"uml:GeneralizationSet" + (packagedElement || "UML:Model")
POD	(("uml:Dependency" || "UML:Dependency") || ("uml:Usage" || "UML:Usage") || ("uml:Abstraction" || "UML:Abstraction") || "uml:InterfaceRealization" || "uml:ComponentRealization" || "UML:Permission" || "Dependency.supplier") + (packagedElement || "UML:Model")
POI	("uml:Interface" || "UML:Interface" || name="interface") + (packagedElement || "UML:Model")
POG	("uml:Generalization" || "UML:Generalization") + (packagedElement || "UML:Model")

Appendix C

Vargha Delaney and Wilcoxon tables for all data

Table C.2 contains the *Vargha Delaney* effect size metric [54] for all combinations of approaches. Table C.1 presents the *Wilcoxon* pairwise test [62] to the same combinations. Both tables are analysed at Subsection 5.2.4.

Table C.1: Wilcoxon test results for all data

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	1103	0.17
Knowledge-based	Bag-of-words	1248.5	0.72
Knowledge-based	Random	1227.5	0.61
Content-based	Knowledge-based	1498	0.17
Content-based	Bag-of-words	1453	0.28
Content-based	Random	1432.5	0.35
Bag-of-words	Knowledge-based	1352.5	0.72
Bag-of-words	Content-based	1148	0.28
Bag-of-words	Random	1278.5	0.88
Random	Knowledge-based	1373.5	0.61
Random	Content-based	1168.5	0.35
Random	Bag-of-words	1322.5	0.88

Table C.2: Vargha Delaney test results for all data

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.42	Small	Content-based	(0.32, 0.53)
Knowledge-based	Bag-of-words	0.48	Small	Bag-of-words	(0.37, 0.59)
Knowledge-based	Random	0.47	Small	Random	(0.37, 0.58)
Content-based	Knowledge-based	0.58	Small	Content-based	(0.47, 0.68)
Content-based	Bag-of-words	0.56	Small	Content-based	(0.45, 0.66)
Content-based	Random	0.55	Small	Content-based	(0.44, 0.65)
Bag-of-words	Knowledge-based	0.52	Small	Bag-of-words	(0.41, 0.63)
Bag-of-words	Content-based	0.44	Small	Content-based	(0.34, 0.55)
Bag-of-words	Random	0.49	Small	Random	(0.39, 0.60)
Random	Knowledge-based	0.53	Small	Random	(0.42, 0.63)
Random	Content-based	0.45	Small	Content-based	(0.35, 0.56)
Random	Bag-of-words	0.51	Small	Random	(0.40, 0.61)

Appendix D

Vargha Delaney and Wilcoxon tables for the top-3 results of the auxiliary experiment

The following tables contains the *Vargha Delaney* effect size metric [54] and the *Wilcoxon* pairwise test [62] for all combinations of approaches, grouped by the number of features at user profile and considering the top-3 results. For more information about this data see the Section 5.3.

Table D.1: Vargha Delaney test results for top-3 results and profile length 1

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.31	Large	Content-based	(0.17, 0.51)
Knowledge-based	Bag-of-words	0.39	Medium	Bag-of-words	(0.2, 0.62)
Knowledge-based	Random	0.81	Large	Knowledge-based	(0.52, 0.95)
Content-based	Knowledge-based	0.69	Large	Content-based	(0.49, 0.83)
Content-based	Bag-of-words	0.56	Small	Content-based	(0.44, 0.68)
Content-based	Random	1	Large	Content-based	-
Bag-of-words	Knowledge-based	0.61	Medium	Bag-of-words	(0.38, 0.8)
Bag-of-words	Content-based	0.44	Small	Content-based	(0.32, 0.56)
Bag-of-words	Random	0.91	Large	Bag-of-words	(0.59, 0.98)
Random	Knowledge-based	0.19	Large	Knowledge-based	(0.05, 0.48)
Random	Content-based	0	Large	Content-based	-
Random	Bag-of-words	0.09	Large	Bag-of-words	(0.02, 0.41)

Table D.2: Wilcoxon test results for top-3 results and profile length 1

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	20	0.08
Knowledge-based	Bag-of-words	25	0.37
Knowledge-based	Random	52	0.03
Content-based	Knowledge-based	44	0.08
Content-based	Bag-of-words	36	0.38
Content-based	Random	64	0
Bag-of-words	Knowledge-based	39	0.37
Bag-of-words	Content-based	28	0.38
Bag-of-words	Random	58	0
Random	Knowledge-based	12	0.03
Random	Content-based	0	0
Random	Bag-of-words	6	0

Table D.3: Vargha Delaney test results for top-3 results and profile length 2

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.43	Small	Content-based	(0.36, 0.5)
Knowledge-based	Bag-of-words	0.54	Small	Knowledge-based	(0.44, 0.64)
Knowledge-based	Random	0.85	Large	Knowledge-based	(0.73, 0.92)
Content-based	Knowledge-based	0.57	Small	Content-based	(0.5, 0.64)
Content-based	Bag-of-words	0.61	Medium	Content-based	(0.53, 0.68)
Content-based	Random	0.89	Large	Content-based	(0.79, 0.95)
Bag-of-words	Knowledge-based	0.46	Small	Knowledge-based	(0.36, 0.56)
Bag-of-words	Content-based	0.39	Medium	Content-based	(0.32, 0.47)
Bag-of-words	Random	0.78	Large	Bag-of-words	(0.64, 0.88)
Random	Knowledge-based	0.15	Large	Knowledge-based	(0.08, 0.27)
Random	Content-based	0.11	Large	Content-based	(0.05, 0.21)
Random	Bag-of-words	0.22	Large	Bag-of-words	(0.12, 0.36)

Table D.4: Wilcoxon test results for top-3 results and profile length 2

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	336	0.04
Knowledge-based	Bag-of-words	427	0.4
Knowledge-based	Random	666	0
Content-based	Knowledge-based	448	0.04
Content-based	Bag-of-words	476	0.01
Content-based	Random	700	0
Bag-of-words	Knowledge-based	357	0.4
Bag-of-words	Content-based	308	0.01
Bag-of-words	Random	611.5	0
Random	Knowledge-based	118	0
Random	Content-based	84	0
Random	Bag-of-words	172.5	0

Table D.5: Vargha Delaney test results for top-3 results and profile length 3

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.49	Small	Content-based	(0.47, 0.51)
Knowledge-based	Bag-of-words	0.6	Small	Knowledge-based	(0.54, 0.66)
Knowledge-based	Random	0.92	Large	Knowledge-based	(0.86, 0.96)
Content-based	Knowledge-based	0.51	Small	Content-based	(0.49, 0.53)
Content-based	Bag-of-words	0.61	Medium	Content-based	(0.55, 0.66)
Content-based	Random	0.93	Large	Content-based	(0.87, 0.96)
Bag-of-words	Knowledge-based	0.4	Small	Knowledge-based	(0.34, 0.46)
Bag-of-words	Content-based	0.39	Medium	Content-based	(0.34, 0.45)
Bag-of-words	Random	0.81	Large	Bag-of-words	(0.72, 0.88)
Random	Knowledge-based	0.08	Large	Knowledge-based	(0.04, 0.14)
Random	Content-based	0.07	Large	Content-based	(0.04, 0.13)
Random	Bag-of-words	0.19	Large	Bag-of-words	(0.12, 0.28)

Table D.6: Wilcoxon test results for top-3 results and profile length 3

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	1540	0.33
Knowledge-based	Bag-of-words	1881.5	0
Knowledge-based	Random	2899.5	0
Content-based	Knowledge-based	1596	0.33
Content-based	Bag-of-words	1904	0
Content-based	Random	2912	0
Bag-of-words	Knowledge-based	1254.5	0
Bag-of-words	Content-based	1232	0
Bag-of-words	Random	2548.5	0
Random	Knowledge-based	236.5	0
Random	Content-based	224	0
Random	Bag-of-words	587.5	0

Table D.7: Vargha Delaney test results for top-3 results and profile length 4

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.6	Small	Knowledge-based	(0.55, 0.65)
Knowledge-based	Random	0.86	Large	Knowledge-based	(0.8, 0.9)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.6	Small	Content-based	(0.55, 0.65)
Content-based	Random	0.86	Large	Content-based	(0.8, 0.9)
Bag-of-words	Knowledge-based	0.4	Small	Knowledge-based	(0.35, 0.45)
Bag-of-words	Content-based	0.4	Small	Content-based	(0.35, 0.45)
Bag-of-words	Random	0.73	Large	Bag-of-words	(0.64, 0.8)
Random	Knowledge-based	0.14	Large	Knowledge-based	(0.1, 0.2)
Random	Content-based	0.14	Large	Content-based	(0.1, 0.2)
Random	Bag-of-words	0.27	Large	Bag-of-words	(0.2, 0.36)

Table D.8: Wilcoxon test results for top-3 results and profile length 4

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	2450	NA
Knowledge-based	Bag-of-words	2940	0
Knowledge-based	Random	4200	0
Content-based	Knowledge-based	2450	NA
Content-based	Bag-of-words	2940	0
Content-based	Random	4200	0
Bag-of-words	Knowledge-based	1960	0
Bag-of-words	Content-based	1960	0
Bag-of-words	Random	3577	0
Random	Knowledge-based	700	0
Random	Content-based	700	0
Random	Bag-of-words	1323	0

Table D.9: Vargha Delaney test results for top-3 results and profile length 5

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.6	Small	Knowledge-based	(0.54, 0.65)
Knowledge-based	Random	0.75	Large	Knowledge-based	(0.68, 0.81)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.6	Small	Content-based	(0.54, 0.65)
Content-based	Random	0.75	Large	Content-based	(0.68, 0.81)
Bag-of-words	Knowledge-based	0.4	Small	Knowledge-based	(0.35, 0.46)
Bag-of-words	Content-based	0.4	Small	Content-based	(0.35, 0.46)
Bag-of-words	Random	0.63	Medium	Bag-of-words	(0.53, 0.71)
Random	Knowledge-based	0.25	Large	Knowledge-based	(0.19, 0.32)
Random	Content-based	0.25	Large	Content-based	(0.19, 0.32)
Random	Bag-of-words	0.37	Medium	Bag-of-words	(0.29, 0.47)

Table D.10: Wilcoxon test results for top-3 results and profile length 5

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	1568	NA
Knowledge-based	Bag-of-words	1876	0
Knowledge-based	Random	2352	0
Content-based	Knowledge-based	1568	NA
Content-based	Bag-of-words	1876	0
Content-based	Random	2352	0
Bag-of-words	Knowledge-based	1260	0
Bag-of-words	Content-based	1260	0
Bag-of-words	Random	1961.5	0.01
Random	Knowledge-based	784	0
Random	Content-based	784	0
Random	Bag-of-words	1174.5	0.01

Table D.11: Vargha Delaney test results for top-3 results and profile length 6

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.59	Small	Knowledge-based	(0.52, 0.66)
Knowledge-based	Random	0.86	Large	Knowledge-based	(0.75, 0.92)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.59	Small	Content-based	(0.52, 0.66)
Content-based	Random	0.86	Large	Content-based	(0.75, 0.92)
Bag-of-words	Knowledge-based	0.41	Small	Knowledge-based	(0.34, 0.48)
Bag-of-words	Content-based	0.41	Small	Content-based	(0.34, 0.48)
Bag-of-words	Random	0.71	Large	Bag-of-words	(0.56, 0.83)
Random	Knowledge-based	0.14	Large	Knowledge-based	(0.08, 0.25)
Random	Content-based	0.14	Large	Content-based	(0.08, 0.25)
Random	Bag-of-words	0.29	Large	Bag-of-words	(0.17, 0.44)

Table D.12: Wilcoxon test results for top-3 results and profile length 6

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	392	NA
Knowledge-based	Bag-of-words	462	0.02
Knowledge-based	Random	672	0
Content-based	Knowledge-based	392	NA
Content-based	Bag-of-words	462	0.02
Content-based	Random	672	0
Bag-of-words	Knowledge-based	322	0.02
Bag-of-words	Content-based	322	0.02
Bag-of-words	Random	559.5	0
Random	Knowledge-based	112	0
Random	Content-based	112	0
Random	Bag-of-words	224.5	0

Table D.13: Vargha Delaney test results for top-3 results and profile length 7

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.56	Small	Knowledge-based	(0.44, 0.68)
Knowledge-based	Random	0.63	Medium	Knowledge-based	(0.46, 0.77)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.56	Small	Content-based	(0.44, 0.68)
Content-based	Random	0.63	Medium	Content-based	(0.46, 0.77)
Bag-of-words	Knowledge-based	0.44	Small	Knowledge-based	(0.32, 0.56)
Bag-of-words	Content-based	0.44	Small	Content-based	(0.32, 0.56)
Bag-of-words	Random	0.56	Small	Bag-of-words	(0.36, 0.74)
Random	Knowledge-based	0.38	Medium	Knowledge-based	(0.23, 0.54)
Random	Content-based	0.38	Medium	Content-based	(0.23, 0.54)
Random	Bag-of-words	0.44	Small	Bag-of-words	(0.26, 0.64)

Table D.14: Wilcoxon test results for top-3 results and profile length 7

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	32	NA
Knowledge-based	Bag-of-words	36	0.38
Knowledge-based	Random	40	0.17
Content-based	Knowledge-based	32	NA
Content-based	Bag-of-words	36	0.38
Content-based	Random	40	0.17
Bag-of-words	Knowledge-based	28	0.38
Bag-of-words	Content-based	28	0.38
Bag-of-words	Random	36	0.59
Random	Knowledge-based	24	0.17
Random	Content-based	24	0.17
Random	Bag-of-words	28	0.59

Appendix E

Vargha Delaney and Wilcoxon tables for the top-5 results of the auxiliary experiment

The following tables contains the *Vargha Delaney* effect size metric [54] and the *Wilcoxon* pairwise test [62] for all combinations of approaches, grouped by the number of features at user profile and considering the top-5 results. For more information about this data see the Section 5.3.

Table E.1: Vargha Delaney test results for top-5 results and profile length 1

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.25	Large	Content-based	(0.11, 0.47)
Knowledge-based	Bag-of-words	0.38	Medium	Bag-of-words	(0.17, 0.63)
Knowledge-based	Random	0.81	Large	Knowledge-based	(0.52, 0.95)
Content-based	Knowledge-based	0.75	Large	Content-based	(0.53, 0.89)
Content-based	Bag-of-words	0.63	Medium	Content-based	(0.46, 0.77)
Content-based	Random	1	Large	Content-based	-
Bag-of-words	Knowledge-based	0.63	Medium	Bag-of-words	(0.37, 0.83)
Bag-of-words	Content-based	0.38	Medium	Content-based	(0.23, 0.54)
Bag-of-words	Random	0.91	Large	Bag-of-words	(0.59, 0.98)
Random	Knowledge-based	0.19	Large	Knowledge-based	(0.05, 0.48)
Random	Content-based	0	Large	Content-based	-
Random	Bag-of-words	0.09	Large	Bag-of-words	(0.02, 0.41)

Table E.2: Wilcoxon test results for top-5 results and profile length 1

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	16	0.03
Knowledge-based	Bag-of-words	24	0.36
Knowledge-based	Random	52	0.03
Content-based	Knowledge-based	48	0.03
Content-based	Bag-of-words	40	0.17
Content-based	Random	64	0
Bag-of-words	Knowledge-based	40	0.36
Bag-of-words	Content-based	24	0.17
Bag-of-words	Random	58	0.01
Random	Knowledge-based	12	0.03
Random	Content-based	0	0
Random	Bag-of-words	6	0.01

Table E.3: Vargha Delaney test results for top-5 results and profile length 2

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.41	Small	Content-based	(0.34, 0.48)
Knowledge-based	Bag-of-words	0.56	Small	Knowledge-based	(0.45, 0.67)
Knowledge-based	Random	0.91	Large	Knowledge-based	(0.79, 0.96)
Content-based	Knowledge-based	0.59	Small	Content-based	(0.52, 0.66)
Content-based	Bag-of-words	0.66	Medium	Content-based	(0.57, 0.74)
Content-based	Random	0.98	Large	Content-based	(0.9, 1)
Bag-of-words	Knowledge-based	0.44	Small	Knowledge-based	(0.33, 0.55)
Bag-of-words	Content-based	0.34	Medium	Content-based	(0.26, 0.43)
Bag-of-words	Random	0.89	Large	Bag-of-words	(0.78, 0.95)
Random	Knowledge-based	0.09	Large	Knowledge-based	(0.04, 0.21)
Random	Content-based	0.02	Large	Content-based	(0, 0.1)
Random	Bag-of-words	0.11	Large	Bag-of-words	(0.05, 0.22)

Table E.4: Wilcoxon test results for top-5 results and profile length 2

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	322	0.02
Knowledge-based	Bag-of-words	442.5	0.28
Knowledge-based	Random	713	0
Content-based	Knowledge-based	462	0.02
Content-based	Bag-of-words	518	0
Content-based	Random	770	0
Bag-of-words	Knowledge-based	341.5	0.28
Bag-of-words	Content-based	266	0
Bag-of-words	Random	698	0
Random	Knowledge-based	71	0
Random	Content-based	14	0
Random	Bag-of-words	86	0

Table E.5: Vargha Delaney test results for top-5 results and profile length 3

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.48	Small	Content-based	(0.46, 0.51)
Knowledge-based	Bag-of-words	0.61	Medium	Knowledge-based	(0.54, 0.67)
Knowledge-based	Random	0.95	Large	Knowledge-based	(0.9, 0.98)
Content-based	Knowledge-based	0.52	Small	Content-based	(0.49, 0.54)
Content-based	Bag-of-words	0.63	Medium	Content-based	(0.57, 0.68)
Content-based	Random	0.96	Large	Content-based	(0.91, 0.99)
Bag-of-words	Knowledge-based	0.39	Medium	Knowledge-based	(0.33, 0.46)
Bag-of-words	Content-based	0.38	Medium	Content-based	(0.32, 0.43)
Bag-of-words	Random	0.89	Large	Bag-of-words	(0.82, 0.94)
Random	Knowledge-based	0.05	Large	Knowledge-based	(0.02, 0.1)
Random	Content-based	0.04	Large	Content-based	(0.01, 0.09)
Random	Bag-of-words	0.11	Large	Bag-of-words	(0.06, 0.18)

Table E.6: Wilcoxon test results for top-5 results and profile length 3

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	1512	0.16
Knowledge-based	Bag-of-words	1903	0
Knowledge-based	Random	2988.5	0
Content-based	Knowledge-based	1624	0.16
Content-based	Bag-of-words	1960	0
Content-based	Random	3024	0
Bag-of-words	Knowledge-based	1233	0
Bag-of-words	Content-based	1176	0
Bag-of-words	Random	2800.5	0
Random	Knowledge-based	147.5	0
Random	Content-based	112	0
Random	Bag-of-words	335.5	0

Table E.7: Vargha Delaney test results for top-5 results and profile length 4

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.61	Medium	Knowledge-based	(0.56, 0.65)
Knowledge-based	Random	0.94	Large	Knowledge-based	(0.88, 0.97)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.61	Medium	Content-based	(0.56, 0.65)
Content-based	Random	0.94	Large	Content-based	(0.88, 0.97)
Bag-of-words	Knowledge-based	0.39	Medium	Knowledge-based	(0.35, 0.44)
Bag-of-words	Content-based	0.39	Medium	Content-based	(0.35, 0.44)
Bag-of-words	Random	0.84	Large	Bag-of-words	(0.77, 0.89)
Random	Knowledge-based	0.06	Large	Knowledge-based	(0.03, 0.12)
Random	Content-based	0.06	Large	Content-based	(0.03, 0.12)
Random	Bag-of-words	0.16	Large	Bag-of-words	(0.11, 0.23)

Table E.8: Wilcoxon test results for top-5 results and profile length 4

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	2450	NA
Knowledge-based	Bag-of-words	2975	0
Knowledge-based	Random	4585	0
Content-based	Knowledge-based	2450	NA
Content-based	Bag-of-words	2975	0
Content-based	Random	4585	0
Bag-of-words	Knowledge-based	1925	0
Bag-of-words	Content-based	1925	0
Bag-of-words	Random	4122	0
Random	Knowledge-based	315	0
Random	Content-based	315	0
Random	Bag-of-words	778	0

Table E.9: Vargha Delaney test results for top-5 results and profile length 5

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.6	Small	Knowledge-based	(0.54, 0.65)
Knowledge-based	Random	0.84	Large	Knowledge-based	(0.77, 0.89)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.6	Small	Content-based	(0.54, 0.65)
Content-based	Random	0.84	Large	Content-based	(0.77, 0.89)
Bag-of-words	Knowledge-based	0.4	Small	Knowledge-based	(0.35, 0.46)
Bag-of-words	Content-based	0.4	Small	Content-based	(0.35, 0.46)
Bag-of-words	Random	0.73	Large	Bag-of-words	(0.64, 0.81)
Random	Knowledge-based	0.16	Large	Knowledge-based	(0.11, 0.23)
Random	Content-based	0.16	Large	Content-based	(0.11, 0.23)
Random	Bag-of-words	0.27	Large	Bag-of-words	(0.19, 0.36)

Table E.10: Wilcoxon test results for top-5 results and profile length 5

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	1568	NA
Knowledge-based	Bag-of-words	1876	0
Knowledge-based	Random	2632	0
Content-based	Knowledge-based	1568	NA
Content-based	Bag-of-words	1876	0
Content-based	Random	2632	0
Bag-of-words	Knowledge-based	1260	0
Bag-of-words	Content-based	1260	0
Bag-of-words	Random	2302	0
Random	Knowledge-based	504	0
Random	Content-based	504	0
Random	Bag-of-words	834	0

Table E.11: Vargha Delaney test results for top-5 results and profile length 6

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.59	Small	Knowledge-based	(0.52, 0.66)
Knowledge-based	Random	0.93	Large	Knowledge-based	(0.83, 0.97)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.59	Small	Content-based	(0.52, 0.66)
Content-based	Random	0.93	Large	Content-based	(0.83, 0.97)
Bag-of-words	Knowledge-based	0.41	Small	Knowledge-based	(0.34, 0.48)
Bag-of-words	Content-based	0.41	Small	Content-based	(0.34, 0.48)
Bag-of-words	Random	0.81	Large	Bag-of-words	(0.67, 0.9)
Random	Knowledge-based	0.07	Large	Knowledge-based	(0.03, 0.17)
Random	Content-based	0.07	Large	Content-based	(0.03, 0.17)
Random	Bag-of-words	0.19	Large	Bag-of-words	(0.1, 0.33)

Table E.12: Wilcoxon test results for top-5 results and profile length 6

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	392	NA
Knowledge-based	Bag-of-words	462	0.02
Knowledge-based	Random	728	0
Content-based	Knowledge-based	392	NA
Content-based	Bag-of-words	462	0.02
Content-based	Random	728	0
Bag-of-words	Knowledge-based	322	0.02
Bag-of-words	Content-based	322	0.02
Bag-of-words	Random	633	0
Random	Knowledge-based	56	0
Random	Content-based	56	0
Random	Bag-of-words	151	0

Table E.13: Vargha Delaney test results for top-5 results and profile length 7

Approach 1	**Approach 2**	**A**	**Effect**	**Superior**	**Confidence interval for A ($\alpha = 0.05$)**
Knowledge-based	Content-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Knowledge-based	Bag-of-words	0.56	Small	Knowledge-based	(0.44, 0.68)
Knowledge-based	Random	0.69	Large	Knowledge-based	(0.49, 0.83)
Content-based	Knowledge-based	0.5	NO effect	NONE is superior	(0.5, 0.5)
Content-based	Bag-of-words	0.56	Small	Content-based	(0.44, 0.68)
Content-based	Random	0.69	Large	Content-based	(0.49, 0.83)
Bag-of-words	Knowledge-based	0.44	Small	Knowledge-based	(0.32, 0.56)
Bag-of-words	Content-based	0.44	Small	Content-based	(0.32, 0.56)
Bag-of-words	Random	0.63	Medium	Bag-of-words	(0.4, 0.81)
Random	Knowledge-based	0.31	Large	Knowledge-based	(0.17, 0.51)
Random	Content-based	0.31	Large	Content-based	(0.17, 0.51)
Random	Bag-of-words	0.38	Medium	Bag-of-words	(0.19, 0.6)

Table E.14: Wilcoxon test results for top-5 results and profile length 7

Approach 1	**Approach 2**	**W**	**p-value**
Knowledge-based	Content-based	32	NA
Knowledge-based	Bag-of-words	36	0.38
Knowledge-based	Random	44	0.08
Content-based	Knowledge-based	32	NA
Content-based	Bag-of-words	36	0.38
Content-based	Random	44	0.08
Bag-of-words	Knowledge-based	28	0.38
Bag-of-words	Content-based	28	0.38
Bag-of-words	Random	40	0.3
Random	Knowledge-based	20	0.08
Random	Content-based	20	0.08
Random	Bag-of-words	24	0.3