REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE IBN KHALDOUN - TIARET

MEMOIRE

Présenté à :

FACULTÉ MATHEMATIQUES ET INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

MASTER

Spécialité : [Génie informatique]

Par :

Figuir Djamila

Sur le thème

Développement d'un système de recommandation à base de connaissances pour le domaine de Tourisme

Soutenu publiquement le 13 / 07 / 2019 à Tiaret devant le jury composé de :

MERATI Medjeded	Grade	Maitre de conférences	Président
BOUDAA Boudjemaa	Grade	Maitre de conférences	Encadreur
KOUADRIA Abderrahmane	Grade	Maitre-Assistant	Examinateur

Acknowledgement

First and foremost, praises and thanks to Allah, the Almighty and Merciful who gave me strength and patience throughout my research work to complete the research successfully.

I would like to express my deep and sincere gratitude to my "framer" Mr. Boudaa Boudjemaa for his availability, support, encouragement and the quality of his advice along this work. I also thank him for the help he has provided and the knowledge he has given me.

A big thank you to my mother and father, for their love, their advices as well as their unconditional support, both moral and economic, which allowed me to carry out the studies that I wanted and consequently this memory.

This project could not have been accomplished without the support of the “DATAtourisme” team, “Stack Overflow” community, and “Google”.

I would like to express my gratitude to my sisters, brothers and friends, who gave me their moral and intellectual support throughout my journey.

I want to take this opportunity to thank Mr. MERATI Medjeded and Mr. KOUADRIA Abderrahmane for accepting to judge and to evaluate this work and to participate in the thesis jury. That they find here the expression of my highest consideration.

Finally, I thank all those who contributed by their advice or encouragement to the completion of this work.

Figuir Djamila

Table of Contents

Abstract

With the rise use of the Internet, recommender systems (RS) are growing progressively and become more popular in our habitual life by helping people to find relevant items (books, movies, hotels, etc.). by using user data, item data, user opinions, preferences, meta-data, demographic information, user behavior or combination of these.

This research work aims to develop a knowledge-based recommender system that recommends Tourism services (hotel, restaurant, cultural sites, etc.) to tourists. In this purpose, we have used a constraint-based approach as type of knowledge-based recommender systems, and the DATAtourisme ontology as a recent rich knowledge about Tourism domain.

The achieved recommender system has proved the capability of recommending complex products and services such as in tourism domain. In addition, the obtained results confirm advantages of these kind of recommendation systems mainly the absence of cold-start problem

Keywords: Knowledge-based recommender system, DATAtourisme ontology, constraint-based approach, explanation, relaxation.

List of Figures

List of Tables

List of Algorithms

GENERAL INTRODUCTION

General Introduction

Context and Motivation

Given the increase in the amount of information and the number of users on the Internet, it has become difficult to find the data; even conventional information retrieval tools do not always provide relevant results because of the information overload problem.

To tackle this problem, the need is increased for new techniques and tools to help users find what they are looking for without much efforts and time. The last two decades witnessed the emergence of new software techniques for many application domains (e-commerce, tourism …). These techniques are called the Recommender Systems (SR, in short).

A Recommender or recommendation System automatically identifies user preferences through their interactions with the system based on either the implicit feedback, explicit feedback or both of them, to suggest recommendations to users.

There are several types of recommender systems, such as, Collaborative Filtering (CF), Content-Based Filtering (CBF), Knowledge-Based Recommender System (KBRS) and Hybrid approaches between these types. In the literature, we find a richness of researches works about the first two type (CF and CBF). However, a little focus about KBRS and few works found in this field. This scarcity of work is due mainly to difficulties meet the developers in the construction of the Knowledge base with domain experts, which is the backbone in the software architecture of any KBRS.

Very recently a new knowledge base about “Tourism domain” has been established by different experts in the domain. This knowledge base is called “DATAtourisme” and constructed on ontological models with all know advantages of the ontology formalism (expressivity, share, semantic richness, extensibility, etc.).

Our present work falls within the of Knowledge-Based Recommender Systems and aims to explicit how to develop them for community researchers by showing their pros and cons. This trend is motivated firstly by the scarcity of researches in KBRS field and secondly, by the availability of DATAtourisme knowledge base.

Problematic and objectives

With the noticed lack of works in the field of KBRS (as we showed previously), the major problematic of this work is how to build Knowledge-Based Recommender Systems and what will be their advantages and limits

In order to contribute in this problematic, we have traced some objectives to be reached in this research work, namely:
At first, to design and implement a knowledge-based recommender system using the new DATAtourisme ontology;
Second, to validate the results in recommendations form our KBRS by real case studies;
Finally, to note the gained benefits of using KBRS in business domains (For instance, Tourism) and the limits of these systems to be tackled in futures works.

Organization

After this introduction, this document is organized into four chapters as follows:

- The **first chapter** presents a general overview on recommendation systems from the past to future passing by the present, and their types such as CF, CBF and KBRS.

- The **second chapter** is dedicated to explain knowledge-based recommender systems, the concept of knowledge base using the ontology formalism and the approaches used to build this type of RSs, namely, Case-based and Constraint-based recommender systems.

- The **third chapter** details Constraint-base recommender system which will be used to build the target recommender system, the DATAtourisme ontology and finished by proposing the architectural design of our KBRS and its functioning process using explanation and relaxation algorithms.

- The **fourth chapter** shows the realization of our application (KBRS) where it gives the development environment, tools and programming languages used in our work, then show some graphical interfaces of our application. It ends by mentioning some pros and cons of knowledge-based recommender systems learned from our experience.

At the end, we conclude this document with an assessment of our contributions, opening the door to certain perspectives envisaged to fill the limits of KBRS.

CHAPTER I

Overview on Recommendation Systems

« Who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast. »
Leonardo da Vinci

1. Introduction

Nowadays, there is a wide trend of users to use recommendations applications in order to facilitate their different social and professional tasks. This chapter aims to present recommender systems and underlying concepts. After exposition of the historical aspect, it gives various classifications of recommender systems found in the literature in which the most important kinds of these systems are detailed.

2. What Is A Recommender System?

In the literature, several definitions of a "Recommender system (RS)" have been introduced. The most popular one is given by (Bo Xiao, 2007):

"*RS are software agents that elicit the interests and preferences of individual consumers […] and make recommendations accordingly. They have the potential to support and improve the quality of the decisions consumers make while searching for and selecting products online.*"

As far as we can tell is that the recommender system is the advisor of users in overwhelming number of available items to help find which they are likely to prefer using user data, item data, user opinions, preferences, meta-data, demographic information, user behavior or combination of these. So, it considerably reduces the user's time to find the most interesting items for him.

In addition, recommender system can also be used to determine the similarity of different products. If the products are very similar to each other, they could interest the same users.

3. Past, Present and Future of Recommender Systems

The preludes of the recommendation systems stem from research into the construction of models representing user choices. in order to facilitate search through the web or e-services, and deal with the problem of overload and wealth of information.

3.1. Past

- **1979**, Grundy (Rich, 1979), a librarian system, is the first recommender engine that described models of users by using stereotypes based on a short interview, and used these stereotypes to produce book recommendations. This work represents an interesting first attempt in the field of recommendation systems. However, its use has remained very limited.
- **1990,** Collaborative filtering (Ekstrand, 2011) appears as a solution to deal with information overload.

- **1992,** the appearance of the Tapestry document recommendation system (D. Goldberg, 1992), it was developed by the "Xerox" research center in the United States, their purpose was to recommend to groups of users' documents from newsgroups that might be of interest to them. The approach used was of the nearest neighbors' type from the user's history. As well as the creation of GroupLens research laboratory, which works explicitly on the automatic recommendation problem within the Usenet newsgroups framework.
- **1994,** GroupLens (P. Resnick, 1994) uses automatic collaborative filtering to identify items in Usenet that may be of interest to a particular user.
- **1995-2006,** successively appear Ringo (Shardanand & Maes, 1995), a music recommendation system and Bellcore (Hill, Stead, Rosenstein, & Furnas, 1995) a video recommendation system. Also, bookseller such as Amazon which is online retailer, and manufacturer of electronic book readers. In addition, Netflix introduces a personalized movie recommendation system, which uses Netflix members' ratings to accurately predict choices for all Netflix members.

3.2. Present

With the rise of the social networks and the rapid development of web services, everything has become today via the Internet and many applications become popular in our lives. People search for jobs on LinkedIn, look for places to spend holidays on tourism web sites, book a Flight Online, share fantastic pictures with there friends on Flickr. Artists also upload their paintings to DeviantArt. On the other hand, people not only rate holidays package or TV series, but also interact with each other on Facebook, see the latest updates of their favorite idols on Twitter, this brings the idea of social recommendation.

Moreover, as more and more new applications appear in social media, people are again facing a huge amount of information that may be interesting to them. Thus, the web service providers will have to face a similar circumstance faced by those offering traditional recommendation (Wang, 2011).

Recommender systems are growing progressively more popular in both e-commerce and in research. Several models of recommendations have been established, e.g. knowledge-based and social methods, demographic and hybrid methods, etc., along with new techniques such as stacking Algorithms[1], Matrix factorization[2], Magic Barrier[3] and Ranking[4], etc. As well as recommendation system prove to be an application area for data mining and machine learning.

[1] **Algorithm Stacking**: is an ensemble learning method, to the problem of building hybrid recommendation systems.
[2] **Matrix factorization**: is an algorithm work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.
[3] **Magic Barrier**: represents the lowest error we can expect from any recommendation algorithm.
[4] **Ranking**: is the process of ordering alternatives in relation to others.

All this progress is aimed at upgrading and evolving the quality of the recommendation and to deal with the problem of information overload.

- **2012 - Today,** Netflix becomes the most nominated service, which is available worldwide and hits 100 million members globally ("About Netflix,").

3.3. Future

The recommendation systems continue to be largely geared towards improving the accuracy of output and improvements in all dimensions. Henceforth there will be intense research going on and these efforts are surely shaping the future of recommender systems, to be more and more useable and practical in real life scenarios.

The Table I- 1 represents some of future research directions on recommender system features and recommended techniques.

System features	Future research directions	Recommended techniques
User data/ preference modeling	- Managing uncertainties of preference modeling. - Storing data in ontology-based repositories and discovering semantic similarities and relations. - Exploring alternate options of ranking and recommending items to the users by considering several criteria.	Deep learning Semantic web ontologies Web usage mining Multi-objective optimization
System platform System architecture	- Utilizing distributed and elastic platforms. - Studying mobile applications and security vulnerabilities in decentralized environments.	Cloud computing Intelligent agents
Adaptivity	- Designing a system to operate within dynamic environments and autonomously choose the appropriate recommendation algorithm.	Autonomous and self-directed learning
Security and privacy	- Analyzing the required amount of user data. - Exploring the tradeoff relation between security and privacy to preserve a suitable balance.	Data mining Machine learning
System performance	- Exploring user perception by considering user's privacy concerns, experience, knowledge domain and emotional states.	Data warehouse Implicit evaluation techniques Game theory

Table I- 1 Future research directions in recommender systems (Taghavi, Bentahar, Bakhtiyari, & Hanachi, 2018)

4. How Does Recommender System Work?

In the previous section, we have defined the recommender system as the user leader to find items of interest from an overwhelming number of available items. Now we would to clarify this definition by illustrating how recommender system works.

This is an illustrating scenario: for example, you are looking for a holiday place. You have a profile on a tourism and vacation website (or mobile application), as well as you have previously rated several holiday packages for places you've visited. Now you are looking for recommendation based on your preferences or tastes. So, the system should already know your preferences. Apparently, you seem to like "Archaeological sites and accommodations with forest view", etc. Based on this information, the system should recommend something similar.

Typically, to provide recommendations, the process requires a list of information about users to formulate the user's profile, which includes items liked or disliked, preferences, tastes, historical research, and so on. But the system needs to be able to learn from users' inputs, and retrain periodically to improve the results.

A recommender engine process through three main phases is summarized in Figure I-1 and detailed in the following points.

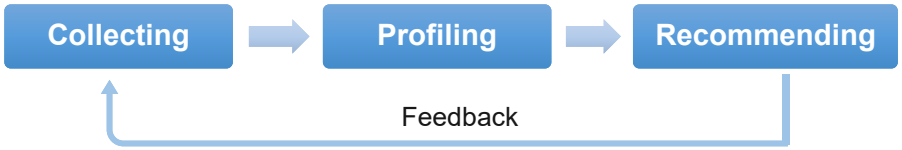

Figure I-1 Phases of Recommendation process

4.1. Collecting Phase

Data is a very important part of all web-based applications, and not only recommendation systems. So, the first step to create a recommender engine is to gather the user information. This collects relevant information of users to generate a user profile or model for the recommendation tasks including user's attribute, behaviors or content of the resources the user accesses. The system needs to know as much as possible from the user in order to provide reasonable recommendation right from the onset.

Recommender systems rely on different types of input such as the explicit feedback, which includes explicit input by users regarding their interest in item or implicit feedback by inferring user preferences indirectly through observing user behavior. Hybrid feedback can also be obtained over the combination of both explicit and implicit feedback (Isinkaye, Folajimi, & Ojokoh, 2015).

4.2. Profiling Phase

Profiling is an important part of recommendation processes since their models are used in order to generate customized recommendations.

User profiles can represent the interests or preferences of both an individual user and a group of users: an individual user profile provides only one user's interests and information, whereas a group user profile describes the common interests or goals of a group of users.

4.2.1. Comparison of The User Profile Types

There are different methods of user profiles depending on the used technique. Each method has advantages and disadvantages. The Table I- 2 in below shows a comparison between the main types.

User Profile Type	Description	Used Techniques	Advantages	Disadvantages
Explicit User Profiles	User manually creates user profile	Questionnaires, Rating	Information gathered is usually of high quality	Requires a lot of efforts from user to update the profile information
Implicit User Profiles	System generates user pro- file from usage history of interactions between user and content	Machine learning algorithms	Minimal user effort is required and easily updatable by automatic methods	Initially requires a large amount of interaction between user and content before an accurate user profile is created
Hybrid User Profiles	Combination of explicit and implicit user profiles	Both explicit and implicit techniques	To reduce weak points and promote strong points of each of the techniques used	N/A

Table I- 2 Comparison of the User Profile Types (Cufoglu, 2014).

4.3. Recommending Phase

The recommendation process ends by offering a list of that items may interest the user. This list should directly respond to key findings issued from collection and profiling phases. A prioritization process is essential to narrowing down finding, and which varies from approach to another. For example, the nearest item to the current user is more prioritized than the others items.

5. Classification of recommendation systems

Recommendation systems can be classified in different ways. Sometimes several terms are used to designate the same method or approach. The objective here is to rely on the best-known classifications on which we base our study. The Figure I- 2 depicts the basic classification of RS.

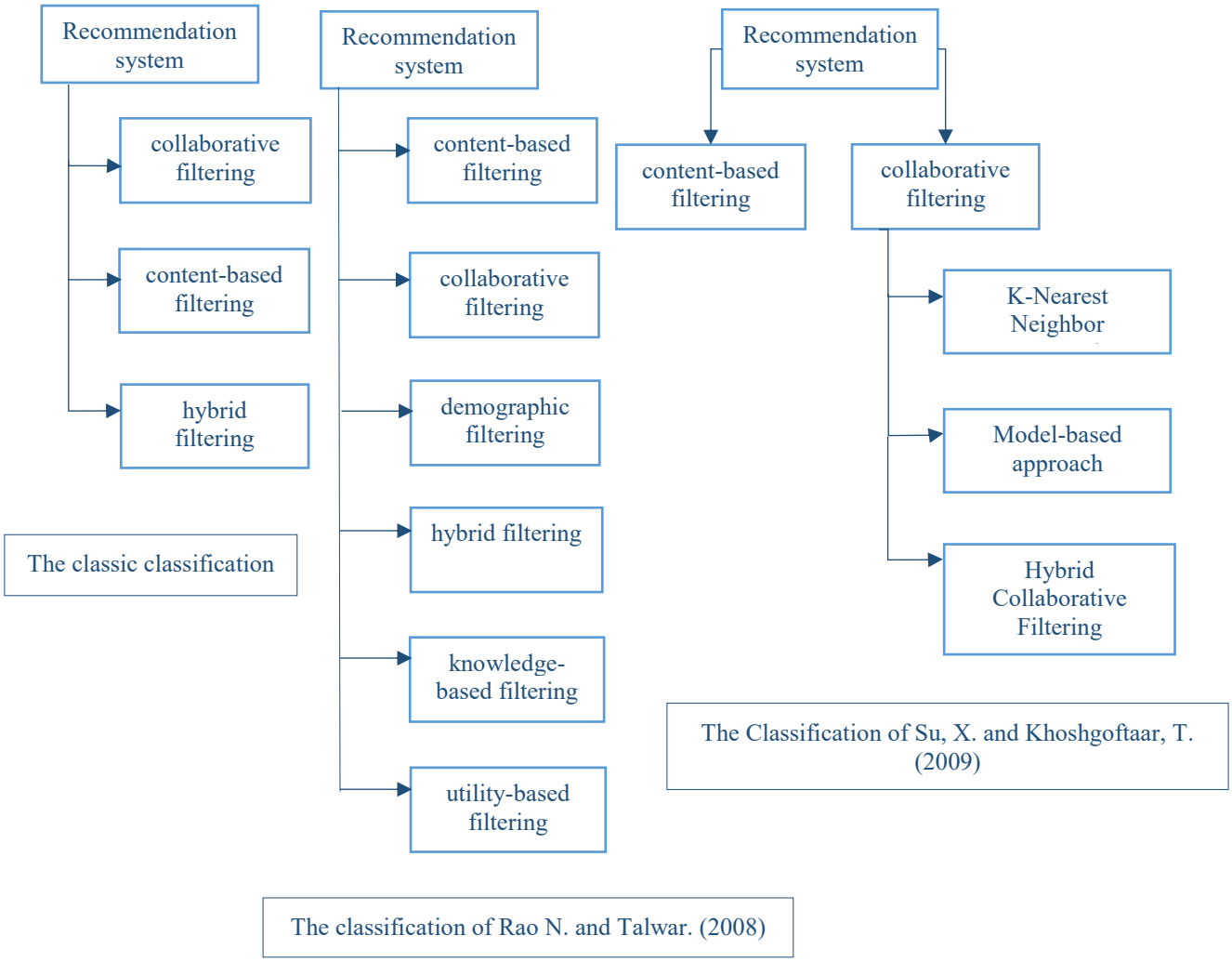

Figure I- 2 Basic Classification of RSs

5.1. Classic classification

This classification of (Adomavicius & Tuzhilin, 2005a) is recognized by three types of filtering: collaborative filtering (CF), content-based filtering (CBF) and hybrid filtering.

5.2. Classification of Rao N. and Talwar. (2008)

It is a classification based on the source of information used.

5.3. Classification of Su, X. and Khoshgoftaar, T. (2009)

(Su & Khoshgoftaar, 2009)classify collaborative filtering into three categories: Su, X. and Khoshgoftaar, T. (2009)

- Memory-based CF approaches: for K-nearest neighbors.
- Model-based CF approaches: including a variety of techniques such as: Clustering, Bayesian networks, matrix factorization, decision processes of Markov.
- Hybrid CF: which combines a CF recommendation technique with one or more

other methods.

We present in the following section the Basic Models of Recommendation Systems such as Collaborative Filtering models, content and knowledge-based models, then Demographic Models, and finally the hybrid approaches.

6. Basic Models of Recommender Systems

A recommendation system seeks to associate two entities: users and items. To handle this task many methods are modeled with different concepts but for the same purpose - supply users with recommendations according to their preferences.

6.1. Collaborative Filtering Models

The basic idea of collaborative filtering methods is the use of something humans have been doing for centuries - sharing opinions with others. Which called -word of mouth- to build an opinion about a product or service they don't know (Schafer, Frankowski, Herlocker, & Sen, 2007).

These models use the collaborative power of the ratings provided by multiple users. To translated into numerical values, can be notes, accounts of purchases made, numbers of visits, etc. in order to formulate the user profile.

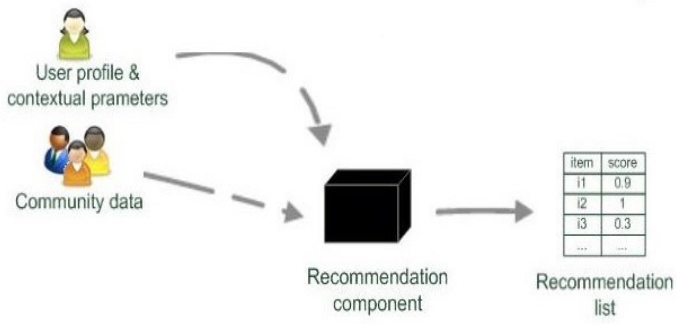

Figure I- 3 Collaboratif Filtering Recommender System

For example, we consider two users named A and B, if A likes the items 1, 2, 3 and B likes 2,3,4, then these users seem to be very similar in their preferences. It is very likely that the ratings in which only one of them has specified a value, are also similar. Thus, similarities are used to make inferences about incompletely specified values. So, A should like item 4 and B should like item 1.

Most of collaborative filtering models focus on leveraging either inter-item correlations or inter-user correlations for the prediction process. Some models use both types of correlations. On the whole a classifier creates two models are referred to as memory-based methods and model-based methods (Aggarwal, 2016):

6.1.1. Memory-based methods

Memory-based methods are also referred to as neighborhood based collaborative filtering algorithms. These neighborhoods can be defined in one of two ways (Aggarwal, 2016):

- User-User Collaborative Filtering: In this case, we have to determine the users who are similar to the current user, then calculate a prediction value for each candidate item for the recommendation by analyzing the notes that the neighbors of the current user have expressed on this item. User-based collaborative filtering was introduced for the first time in the GroupLens system (P. Resnick, 1994).
- Item-Item Collaborative filtering: With regard to conduct the rating predictions for target item P by user A, the first step consists in identifying a set S of items that are most similar to target item P. The ratings in item set S, which are specified by A, are used to predict whether the user A will like item P. The Similarity functions are computed between the columns of the ratings matrix to discover similar items.

6.1.2. Model-based methods

In model-based methods, machine learning and data mining methods are used within the sphere of predictive models. These models are learned to impute the missing or unobserved values in the rating matrix. Some examples of such model-based methods include decision trees, Bayesian methods, rule-based and latent factor models (Aggarwal, 2016).

6.2. Content-Based Models

Content-based models (CBM) attempts to recommend items to the active user similar to those bought, visited, viewed, rated positively in the past, or is examining in the present. It is based on the concept that items with similar contents will be rated similarly. The term "content" refers to the descriptions, attributes, and futures of objects intended for recommendation (J. Bobadilla, 2013).

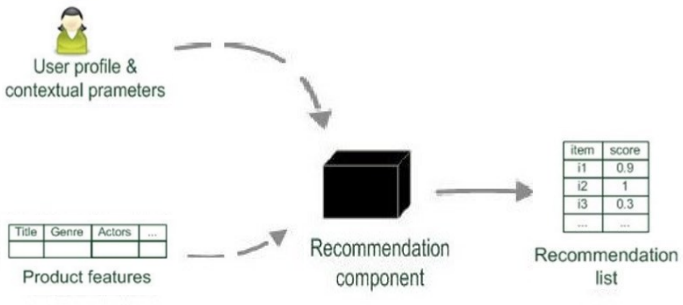

Figure I- 4 Content-Based Recommender System

For example, if a user reads a book A with attributes X, Y and Z, then the CBM will recommend books that are more similar to contents X, Y and Z. That's why two sets must be made: items profiles and user profile. The first set must include the full description and features of a specific item, and the second one must contain all interactions (e.g. feedback) between the user and the system, such as comments, critiques, ratings, opinions and all information which increase the accuracy of predictions and recommendations.

These methods are best suited to situations where there is known data on an item (name, location, description, etc.), but not on the user. Content-based recommenders handle recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on product features (Aggarwal, 2016).

6.2.1. Feature Extraction

Broadly, feature extraction entails reducing the amount of resources required to describe a large set of data. The first stage in all content-based models is to extract discriminative features for representing the items. Discriminative features are those, which are highly predictive of user interests. Although it is possible to use any kind of representation, such as a multidimensional data representation, the most common approach is to extract keywords from the underlying data. In many cases, the items may have multiple fields describing various aspects of the item. The various fields need to be weighted appropriately in order to facilitate their use in the classification process (Aggarwal, 2016).

6.3. Knowledge-Based Recommender Systems

In knowledge-based recommender systems, the ratings are not used for the purpose of recommendations. Rather, the recommendation process is performed in the context of knowledge bases, by using similarities between customer requirements and item descriptions, or the use of constraints specifying user requirements (Aggarwal, 2016).

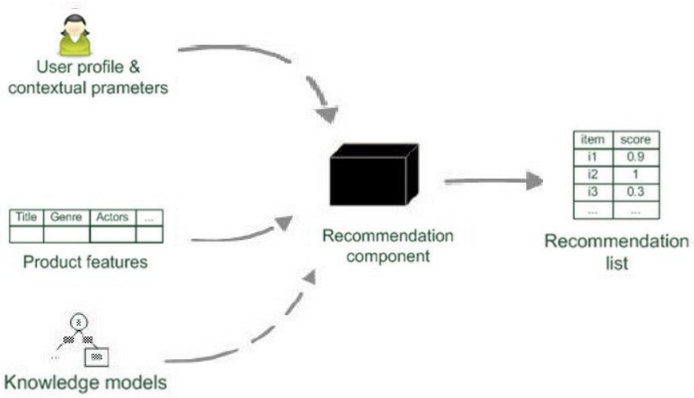

Figure I- 5 Knowledge-Based Recommender System

6.4. Hybrid Recommender Systems

We have mentioned above that collaborative filtering systems rely on community ratings, content-based methods rely on textual descriptions and the target user's own ratings, and knowledge-based systems rely on interactions with the user in the context of knowledge bases. In many cases where a wider variety of inputs is available, one has the flexibility of using different types of recommender systems for the same task. In such cases, many opportunities exist for hybridization, where the various aspects from different types of systems are combined to achieve the best of all worlds (Aggarwal, 2016).

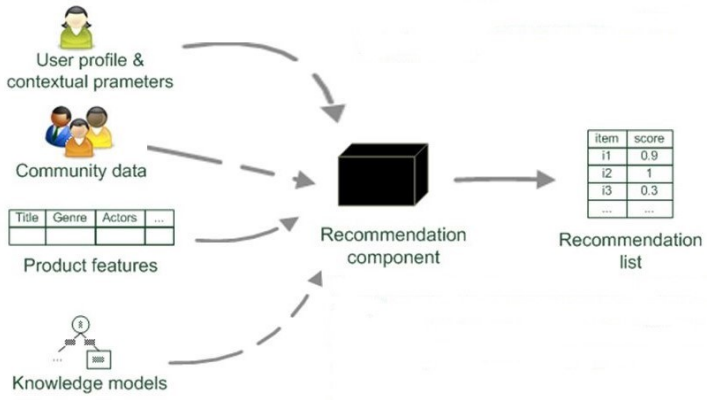

Figure I- 6 Hybrid-Based Recommender System

7. Issues and Challenges in RSs

This section describes the most common issues and challenges that facing RS deployment

- Sparsity: Majority of users do not rate most of items and therefore the ratings matrix becomes very sparse. Due to this, the data sparsity problem arises that declines the chances of finding a set of users with similar ratings (Kumar & Sharma, 2016).
- Cold-start: This problem occurs when new users enter the system or new items are added to the catalogue. In such cases, neither the taste of the new users can be predicted nor can the new items be rated or purchased by the users leading to less accurate recommendations (Khusro, Ali, & Ullah, 2016).
- Grey Sheep: This problem occurs in pure CF systems where opinions of a user do not match with any group and therefore, is unable to get benefit of recommendations(Khusro et al., 2016).
- Scalability: The rate of growth of nearest-neighbor algorithms shows a linear relation with number of items and number of users. It becomes difficult for a typical recommender to process such large-scale data (Khusro et al., 2016).
- Privacy: RSs are bound to gather as much user data as possible and to exploit it to the fullest. But on the other side, this may create a negative impression on the users' mind about their privacy because the system knows too much about them(Balraj Kumar, 2016).
- Robustness of RSs: Another major challenge in RSs is their robustness to attacks. Robustness is a performance measure of RSs. To gain certain profits, an attacker may generate some fake user profiles based on some attack models, such as Push/Nuke Attacks to make some target items more/less popular respectively. Such attacks are collectively called shilling attacks or profile injection attacks (Kumar & Sharma, 2016).

8. Conclusion

In this chapter we have discussed the recommender systems as an important kind of software applications in the last years. We have exposed the past, present and the future of these systems. Also, a focus is given on the most recommendations methods such as, filtering collaborative, content-based and knowledge-based. The latter (knowledge-based recommender system) will be the topic of the next chapter.

CHAPTER II

Knowledge-Based Recommender Systems

«Knowledge is knowing that a tomato is a fruit. Wisdom is knowing not to put it in a fruit salad. »
Brian O'Driscoll

1. Introduction

In this work we aim to propose a knowledge-based recommender system (KBRS) in the field of Tourism. This chapter introduces KBRS by defining the underlying concepts and notions allowing building that latter. We will tackle the knowledge notion which is considered as the main component, the ontology formalism to represent knowledges and the types of this kind of recommender systems, namely, case-based recommender systems and constraint-based recommender systems.

2. What is Knowledge?

Defining the meaning of knowledge requires to determine the distinctive differences between Data, information, knowledge and wisdom, which they are the major elements of human thinking and reasoning process. (Bellinger & Castro, 2004) Proposes the following definitions:

- Data: is raw, it simply exists and has no significance beyond its existence (in and of itself). It can exist in any form, usable or not. It does not have meaning of itself.
- Information: is data that has been given meaning by way of relational connection. This "meaning" can be useful, but does not have to be.
- Knowledge: is the appropriate collection of information, such that its intent is to be useful. Knowledge is a deterministic process. When someone "memorizes" information (as less-aspiring test-bound students often do), then they have amassed knowledge. This knowledge has useful meaning to them, but it does not provide for, in and of itself, an integration such as would infer further knowledge.
- Wisdom: is an extrapolative and non-deterministic, non-probabilistic process. It calls upon all the previous levels of consciousness. It beckons to give us understanding about which there has previously been no understanding.

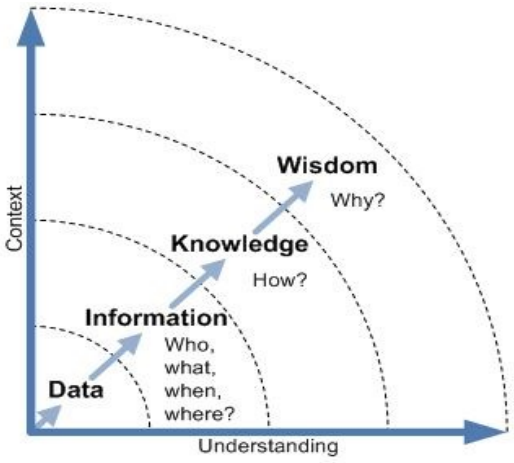

Figure II- 1 The Dick Whittington Model for Knowledge Management

The definition of "knowledge" belongs to the domain of philosophy or epistemology (Bachimont, 2004) clarify that:

"Knowledge is the ability to exert an action to achieve a goal".

This definition raises the ideal character of knowledge, and the importance of the finality of knowledge.

2.1. Types of knowledge

The below table gives different types of Knowledge (Akerkar, 2010):

Knowledge Type	Description
Domain knowledge	Domain knowledge is valid knowledge for a specified domain. Specialists and experts develop their own domain knowledge and use it for problem solving.
Meta knowledge	Meta knowledge can be defined as knowledge about knowledge.
Heuristic knowledge	Heuristic is a specific rule-of-thumb or argument derived from experience.
Explicit knowledge	Explicit knowledge can be easily expressed in words/numbers and shared in the form of data, scientific formulae, product specifications, manuals, and universal principles. It is more formal and systematic.
Tacit knowledge	Tacit knowledge is the knowledge stored in subconscious mind of experts and not easy to document. It is highly personal and hard to formalize, and hence difficult to represent formally in system. Subjective insights, intuitions, emotions, mental models, values and actions are examples of tacit knowledge.

Table II- 1 Type of Knowledge.

3. What is Knowledge-base?

Knowledge-bases support collecting, organizing, retrieving, and sharing knowledge, if we define knowledge-base as a centralized database for spreading information or data plus their meaning. Then we must consider the differences between knowledge-base and database.

3.1. Knowledge-base vs Database

First of all, a database is a collection of information organized in such a way that a computer program can quickly select desired pieces of data (Beal), it is mostly also limited on just these functionalities. In contrast to that, a knowledge-base is a collection of knowledge in the form of subject-problem-solution information that pertains to a specific topic or subject of interest (Beal). A knowledge-base can use many databases or can enrich it with information from public data sources. One popular example of a knowledge base is the Microsoft Help & Support Knowledge-Base, and the data stored inside it provides answers, not just a list of data resources. More technically, RDFS and OWL provide the most popular data models for knowledge bases, but using them does not prevent someone to represent knowledge in a wrong way. The test is the "ontological commitment[5]" of the classes and properties used.

What is the difference again? Simply the difference is that a knowledge-base stores knowledge, while a database stores and organizes data.

[5]See https://en.wikipedia.org/wiki/Ontological_commitment

4. Knowledge representations

The aim of the knowledge representation and reasoning is to express knowledge in a computer tractable form, moreover to depict or understand the behavior of systems with regard to the knowledge it has, etc.
The knowledge manipulation goes through three successive steps: Knowledge acquisition, Knowledge reasoning, and finally the decision or the action.

- Knowledge acquisition: is the process of collecting or absorbing and storing the knowledge.
- Knowledge reasoning: is the use of the knowledge representation in order to derive or, deduce new knowledges.
- Decision/action: is the process of *making* choices and decide what to do next.

4.1. Forms of Representing Knowledge

In this section we will look at the possibility of using Natural Language, Databases, and First Order Logic as knowledge representations (John A. Bullinaria, 2005).

4.1.1. Natural Language as a Knowledge Representation

Natural language is so far the most comprehensive tool for humans to encode and reason with knowledge (how many text books are not written in natural language?). Therefore, it could be viewed as the best knowledge representation formalism available. so why not use that to represent knowledge in the knowledge-base systems? we quote here some disadvantages of natural language:

- Both the syntax and semantics are very complex and not fully understood.
- There is little uniformity in the structure of sentences.
- It is usually ambiguous.

4.1.2. Database as a Knowledge Representation

We have already defined database as a collection of information organized in this way a computer program can quickly select desired pieces of data. Databases are clearly very powerful, but they are rather limited. The important issues are:

- Only simple aspects of the problem domain can be accommodated.
- We can represent entities, and relationships between entities, but not much more.
- Basically, the only reasoning possible is simple lookup, and we usually need more sophisticated processing than that.

4.1.3. First Order Logic as a Knowledge Representation

First-order logic[6] is symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. The predicate modifies or defines the properties of the subject. A sentence in first-order logic is written in the form P(x), where P is the predicate, and x is the subject, represented as a variable. We can also manipulate the logic representations to generate new knowledge, e.g.:

$$\frac{\neg man\ (Djamila) \qquad \neg man(x) \Rightarrow woman(x)}{woman\ (Djamila)}$$

First-order logic as a knowledge representation is very expressive, and has unambiguous syntax and semantics. Except that there is no generally efficient procedure for processing knowledge.

4.1.4. Ontologies as a knowledge representation

Ontology is the formal way to represent knowledge. It facilitates the management of unstructured information and helps to detect ambiguities, inconsistencies and contradictions while building representation of a large and complex domain.
Using an Ontology, the knowledge is represented as a set of concepts within a domain and relationships between pairs of concepts.

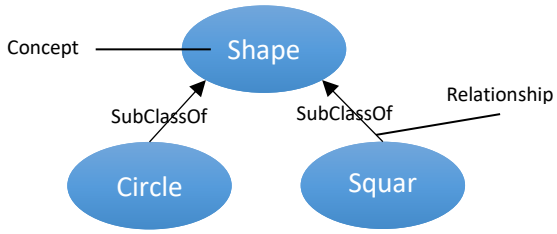

Figure II- 2 Concept and Relationship of Ontology

In the next section we introduce the semantic web, and the ontologies as well as the relationship between them.

5. The semantic Web and Ontologies

The increase in information on the web has led to the appearance of diverse portals, thus the semantic web allows machines to understand semantics, the meaning of information on the Web, so the latter can perform many tasks instead of humans, for example finding, sharing, and combining information.

[6] See https://whatis.techtarget.com/definition/first-order-logic

5.1. What is the Semantic Web?

In 1999, Tim Berners-Lee published the book "Weaving the Web" in which he designed a portrait of the Web and the paths for its future. whereat the term semantic web appeared. in the same year he enunciated his famous quote (Berners-Lee, 1999):

"I have a dream for the Web in which computers become capable of analyzing all the data on the Web - the content, links, and transactions between people and computers. A « Semantic Web », which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines."

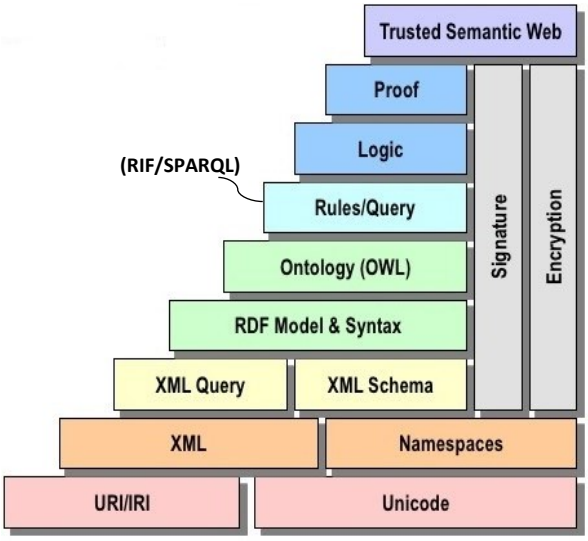

Figure II- 3 The layers of the Semantic Web

Berners-Lee[7] suggested the Semantic Web layers illustrated in Figure II- 3, which is discussed in depth for instance in (Patel-Schneider & Fensel, 2002) and (Patel-Schneider & Siméon, 2002). The lower levels define the common syntax. Uniform resource identifiers (URIs) identify resources in the web, while Unicode is a standard for symbols exchange. The Extensible Markup Language (XML) is for transfer and store data, and XML Schema represent the grammars for valid XML documents. The upper levels introduce formal representations of High-level (RDF and OWL). The following point highlight their descriptions.

5.2. What is RDF, RDFs, OWL and SPARQL?

- RDF: The Resource Description Framework (RDF) is a model for describing resources in the World Wide Web with a triplet (subject, predicate, object), so that the subject related to an object via a predicate.

Figure II- 4 Example of RDF

- RDFs: RDF Schema[8] is a semantic extension of RDF, which describe the basic concepts and abstract syntax of RDF (classes and properties). The RDF Schema class and property is similar to the type systems of object-oriented programming languages such as Java.

[7] See http://www.w3.org/DesignIssues/Semantic.html.
[8] See https://www.w3.org/TR/rdf-schema/

- OWL: Web Ontology Language (OWL)[9] is a Semantic Web language aims to represent rich and complex knowledge about things, groups of things, and relations between things. It is more expressive than RDF since it intended to formulate, exchange and reason with knowledge about a domain of interest. Some fundamental notions should first be explained to understand how knowledge is represented in OWL. These basic notions are in the following points:
  - Axioms: the basic statements that an OWL ontology expresses.
  - Entities: elements used to refer to real-world objects.
  - Expressions: combinations of entities to form complex descriptions from basic ones
- SPARQL: (simple protocol and RDF query language) is a query language for accessing and manipulating data stored in RDF structures. There are some similarities with SQL because it shares several keywords such as SELECT, WHERE, etc. A simple example of a SPARQL query:

```
SELECT DISTINCT? film_URI WHERE {
? film_URI rdf: type <http://dbpedia.org/ontology/Film>.
} LIMIT 10
```

In the context of the Semantic Web, ontologies play a particularly fundamental role. it helps computers to process the web content on a semantic level.

5.3. Ontologies

Ontologies are computational artefacts has appeared in computer science, particularly in artificial intelligence. In its original meaning in philosophy, the term "ontology" refers to the study of being or existence and the organization of reality (Guarino, 1995) on this paper. Guarino and Giaretta (Guarino, 1995) proposed to consider the ontology as:

"a logical theory that gives an explicit and partial account of a conceptualization."

The meaning of conceptualization in this definition below (R.Gruber, 1993):

"*A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose."*

So far, we have known the meaning of ontology in philosophy. the above definitions can be summarized in the following figure:

Figure II- 5 From Reality to Ontology.

[9] See https://www.w3.org/TR/owl-primer/

The notion ontology, has been defined from many viewpoints. In 1998, Studer et al (Studer, Benjamins, & Fensel, 1998) introduced a global definition that captures several features of an ontology:

"An ontology is a formal, explicit specification of a shared conceptualization."

- Ontologies are formal because they are designed to be processed by the computers.
- Ontologies are explicit, their concepts and relations are explicitly defined.
- Ontologies relate to a specific domain of interest.
- Ontologies are shared (or shareable) between users who have already agreed on ontological commitments.
- Ontologies are conceptualizations, they describe the real world in abstract models composed of concepts.

5.3.1. Ontology Components

Technically, the main components of an ontology break down into two essential design (Bullinger-Hoffmann, 2008): First constructs represent classes (subclasses), attributes and relations between objects. Second, rules and axioms to describe general facts and conditions. According to (Bullinger-Hoffmann, 2008) we illustrate these elements in the subsequent points.

- Classes: (concepts) are an abstract definition of objects which are similar to identical in their structure and behavior.
- Objects: are a concrete individual of a class.
- Attributes: are the properties of objects that constitute the object's structure.
- Relations: represent an association or interaction between two or more classes and consequently associations between the instances of these classes.
- Axioms and Inference Rules: axioms are used to describe the ontological assertions that will be considered true. Inference rules allow to deduce new information that is not explicitly stored.

5.3.2. Types of ontologies

Influenced by the publication of (Guarino, 1997), ontologies can be divided into four types, summarized in the Figure II-6.

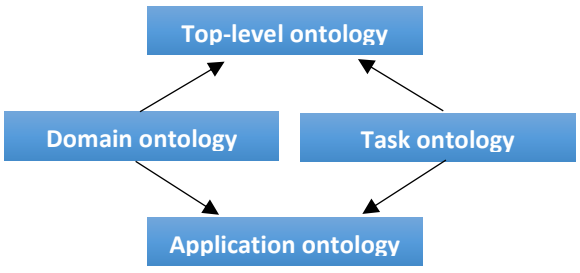

Figure II- 6 Types of Ontologies

- Top-level ontologies: aim to describe very abstract and general concepts, such as space, time, matter, objects, events, actions, etc., that must be consensual for a large community of users.
- Domain ontologies: provide knowledge within a specific domain, and a commonly agreed understanding of a domain.
- Task ontologies: visualizes the systematized vocabulary and types of knowledge required for the task.
- Application ontologies: tailored for a specific application, its concepts are related to a particular domain and task, it can not be reused.

6. Knowledge-based recommender systems

In general, traditional recommendation methods rely on previous users' interactions such as ratings. The more users interact with the recommender engine, the more accurate the recommendations will be. For example, in the content-based approach, the user will be recommended items similar to the ones she preferred in the past, while for the collaborative filtering the user will be recommended items that people with similar tastes and preferences liked in the past (Adomavicius & Tuzhilin, 2005b). Thus, in some cases, the recommendation process often suffers from a lack of product evaluation due to the greater complexity of the product domain. for this situation, it is better to give the user access to control the recommendation process through direct requirement specification. In other cases, the ratings may be time-sensitive. Since they evolve with changing product availability and corresponding user requirements.
Knowledge-based recommender systems does not use users' evaluations, but will instead rely on similarities between customer requirements and item descriptions (case-based) or through the use of constraints specifying user requirements (constraint-based). What makes this type of system special as it allows users to explicitly specify what they want.

Approach	**Conceptual Goal**	**Input**
Collaborative	Give me recommendations based on a collaborative approach that leverages the ratings and actions of my peers/myself.	User ratings + Community ratings
Content-based	Give me recommendations based on the content (attributes) I have favored in my past ratings and actions.	User ratings + Item attributes
Knowledge-based	Give me recommendations based on my explicit specification of the kind of content (attributes) I want.	User specification + Item attributes + Domain knowledge

Table II- 2 The conceptual goals of various recommender systems (Aggarwal, 2016).

There are different interaction forms between user and the knowledge-based recommender, which can be used either in isolation, or in combination, (Aggarwal, 2016) defined them as follows:

- Conversational systems: In this case, the user preferences are determined in the context of a feedback loop. The main reason for this is that the item domain is complex, and the user preferences can be determined only in the context of an iterative conversational system.
- Search-based systems: In search-based systems, user preferences are elicited by using a preset sequence of questions such as the following: "Do you prefer a house in a suburban area or within the city?"
- Navigation-based recommendation: In navigation-based recommendation, the user specifies a number of change requests to the item being currently recommended. Through an iterative set of change requests, it is possible to arrive at a desirable item. An example of a change request specified by the user, when a specific house is being recommended is as follows: "I would like a similar house about 5 miles west of the currently recommended house."

6.1. Case-based recommendation

In case-based recommendation, specific cases are explicitly determined by the user as targets. Similarity metrics are defined on the item attributes to retrieve examples similar to these targets, which are iteratively modified through the process of critiquing. Critiques can be simple, compound, or dynamic (Aggarwal, 2016).

6.2. Constraint-based recommendation

Constraint-based recommendation enable users to set hard requirements or constraints on the item attributes, these constraints and item attributes are matched with domain-specific rules to provide recommendations. In addition, users can add or relax constraints depending on the size of the output (Aggarwal, 2016).

7. Conclusion

In this chapter, we have presented the main elements which enter into the construction of knowledge-based recommender systems (KBRS) such as knowledge, ontologies and briefly the both kinds of KBRS.

The next chapter will detail one of them, namely, the constraint-based recommender systems, and describe our recommendation approach in the context of knowledge-based recommender systems.

CHAPTER III

The Proposed Knowledge-Based Recommender System

« The best way to escape from a problem is to solve it. »

Alan Saporta

1. Introduction

The literature witnessed a rarity of Knowledge-based recommender systems (KBRS) which still difficult to build. This chapter aims to propose a software architecture for this kind of recommendation systems, in which the main components are detailed. Our KBRS uses constraint-based approach working on DATAtourisme ontology. Also, principal algorithms for explanation and relaxation are presented in this chapter.

2. The Ontological Representation of e-Tourism Domain

Today, tourism has become one of the main income sources for a country, that's why it is the most important and growing sector in the world.

Generally, when a tourist wants to plan a trip, she/he will need to use internet as a rich source of information to search and select Point of Interests (POIs). This is the core problem in the sphere of tourism: overwhelming number of different POIs. Therefore, we need to incorporate recommender systems in E-tourism platforms to help find the most interesting items for him, based on his preferences or requirements. For example, the income level of the tourist, because he tends to lose money by making the wrong choices.

Currently, there are many E-tourism applications for instance TripAdvisor, Kayak, Touropia, and so on. In the wake of the rapid technological development of mobile devices such as smartphones, tourism has reached a new higher level, according to the use of the functional content of these modern devices.

Among the mobile features: webcam, GPS, Dynamic maps, and others which make the tourism applications more and more intelligent by providing recommendations based on the user location and/or time-sensitive recommendations as examples. Moreover, they may provide additional information about the place where the user is located and the objects in view (Augmented Reality), through the use of a webcam, GPS interface, machine vision algorithms and information about current locations.

In this context, the large diversity of tourism vocabulary has led to design the "DATAtourisme"[10] ontology that has been chosen in order to develop our Tourism knowledge-based recommender system.

2.1. DATAtourisme Ontology

Nowadays, Ontologies have a major role in knowledge representation and modeling. By using ontologies, we can benefit from several advantages, namely (McGuinness):

- To share common understanding of the structure of information among people or software agents.
- To enable reuse of domain knowledge, and to make domain assumptions explicit.
- To separate domain knowledge from the operational knowledge

[10] See http://www.datatourisme.fr/

DATAtourisme[11] is a national system supported by the DGE (the Directorate-General for Enterprises) with the Tourism and Territories network in France, and the winner of the Future Investment Program (PIA) in 2015.

It aims to gather within a national platform the tourist information produced by the Tourist Offices, Departmental Agencies and Regional Tourism Committees, in order to disseminate them in open-data and so facilitate the creation of innovative tourist services by start-ups, digital agencies, media and other public or private actors.

In January 2017, the DATAtourisme project reached a new milestone with the publication of version 1.0 of its ontology. Then, they released version 2.0 on January 15, 2019.

2.1.1. Basic Concepts

The central concept of ontology is the concept: PointOfInterest. It is defined as any tourist element that deserves to be described and valued. A POI (Point of Interest) is a tourist item that is managed by an Agent and that can be consumed via products and services. This is the minimum class to instantiate for a product to be managed in the DATAtourisme information system.
As POI examples in this ontology, we find: Restaurant, Hotel, Practice, Heritage Object, and Event.
A POI is broken down into 4 different subtypes ("Ontologie DATAtourisme v2.0_Documentation," 2019), as shown in the Figure III- 1.

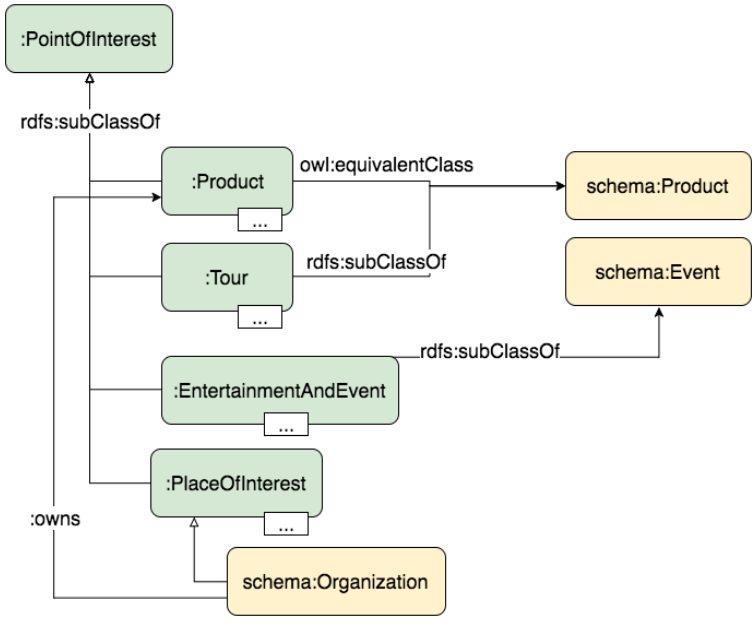

Figure III- 1 Basic Schema of DATAtourisme Ontology

[11] See: http://www.datatourisme.fr/

- Product

: Product: a tourist object that can be consumed (e.g. a hotel room, a practice activity, a guided tour, ...).

- Touristic itinerary

: Tour: an itinerary is a POI which proposes a route composed of stages forming a path.

- Entertainment and event

: EntertainentAndEvent: events, festivals, exhibition, or any other event having a beginning and an end.

- Place of interest

: PlaceOfInterest: a place with a tourist interest (for example, a natural site, a cultural site, a village, a restaurant ...).

2.1.2. Main Properties and Relationships

In general, POI aggregates several information ("Ontologie DATAtourisme v2.0_Documentation," 2019) (such as: location, features, themes, etc.). Figure III- 2 displays the main properties and relationships.

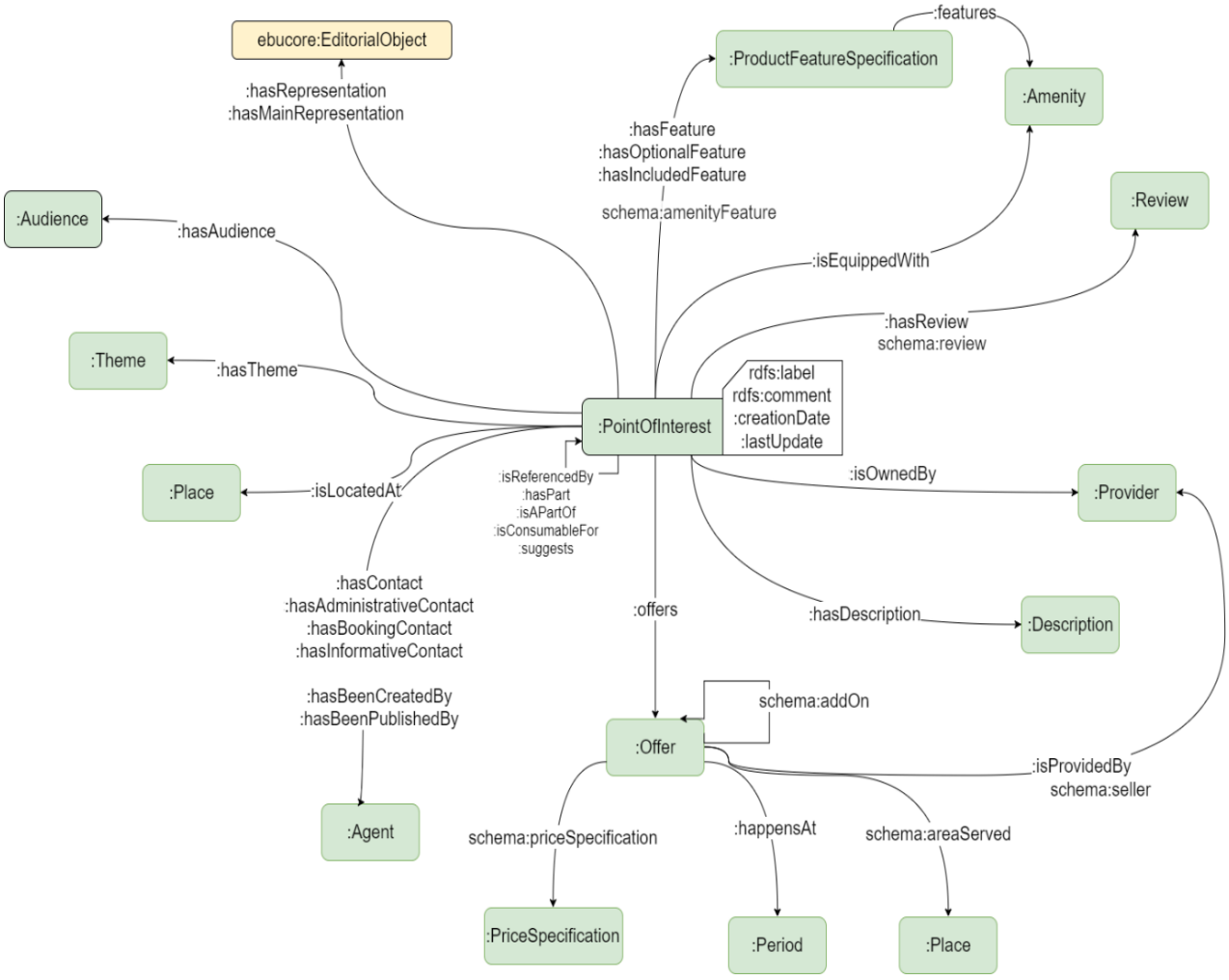

Figure III- 2 General schema of DATAtourisme Ontology

The Table III- 1 describes main properties and relationships of DATAtourisme ontology depicted in previous figure.

Information	Semantic relationship	Description
The localization	[:isLocatedAt]	Where is the POI located and what schedules are applied.
Contacts	[:hasContact]	Who to contact for what needs.
The owner	[:isOwnedBy]	A POI can belong to an Agent (a person or an organization) via this relationship.
The consumption	[:offers]	Price and period to consume the product. Note that consumption is only possible through an instance of **:Offer**. Depending on their type, not all POIs can directly reference tariffs (POIs not merchants).
The audience	[:hasAudience]	The target audience for the POI.
Multimedia	[:hasRepresentation]	Documents that are representations of the POI.
The equipment's	[:hasFeature]	What equipment is available and according to which cardinalities.
Classifications and labels	[:hasReview]	Which rankings and labels evaluate the product and with how much score.
The themes	[:hasTheme]	Which themes are associated with the POI.
Suggests	[:suggests]	Allows you to link a POI with another complementary POI that may appeal to the consumer. Ex: A ski resort with a ski hire.

Table III- 1 Main properties and relationships of the "DATAtourisme" ontology

2.1.3. Interconnection with Other Ontologies

The ontology DATAtourisme is modeled in such a way that it is interconnected with standard or authoritative ontologies in their field of competence, such as DublinCore, SKOS, EBUCore, etc. So that they do not reinvent twice a concept, a property or a relationship that already exists ("Ontologie DATAtourisme v2.0_Documentation," 2019).

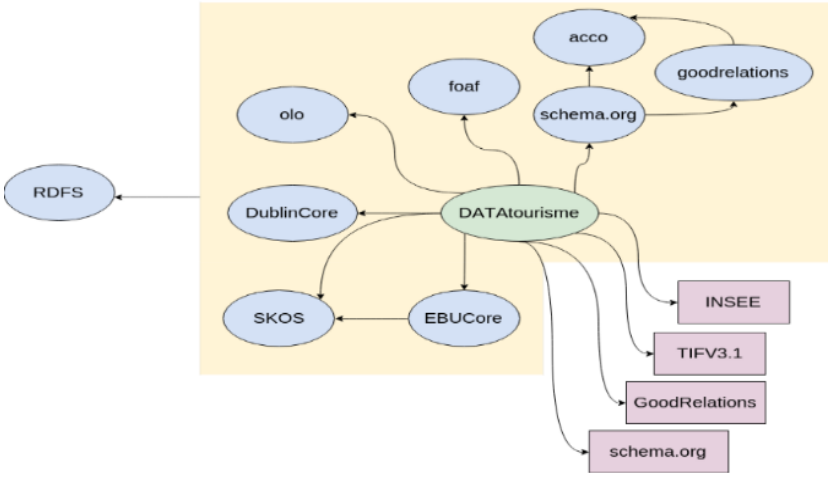

Figure III- 3 Interconnection between DATAtourisme and standard ontologies

3. Constraint-Based Recommendation

As cited in Chapter II, the constraint-based approach can be used to build knowledge-based recommender systems. It attempts to make recommendations for domains where items are more complex and many customers do not know all the technical features in detail.

This approach captures the requirements of the current user in order to derive new solutions. Moreover, it proposes repairs in situations where no solution could be found, and supports explanations as why a system has recommended a specific solution.

Knowledge-base of a constraint-based recommender system is built on three types of knowledge; knowledge about the users, knowledge about the items and knowledge about the matching between the items and user's need.

Deeply, the major ingredients of a constraint satisfaction problem can be defined through two sets of variables (VC, VPROD), a set of finite domains for these variables (D) and three different sets of constraints (CCOMP, CF, CPROD). These variables and constraints are discussed in depth for instance in (Felfernig & Burke, 2008), (Felfernig, Friedrich, Jannach, & Zanker, 2015) and (Jannach, Zanker, & Fuchs, 2009). In conformity with these three references we give the following definitions:

- Customer Properties: V_C describes all possible requirements which can be specified by customers.
- Product Properties: V_{PROD} is a set of variables describing item features or properties.
- Products: C_{PROD} represents one constraint in disjunctive normal form that defines elementary restrictions on the possible instantiations of variables in V_{PROD}.

- Filter Conditions: C_F define the relationship between potential customer requirements and the given product assortment V_{PROD}.
- Compatibility Constraints: C_{COMP} are (in)compatibility constraints restricting the set of possible requirements.

3.1. Recommendation Task

Felfernig and Burke declared that the task of deriving recommendations for a customer is denoted as recommendation task. Given a set of customer requirements, we can calculate a recommendation (result) (Felfernig & Burke, 2008).

The recommendation task can be defined as a constraint satisfaction problem (V_C, V_{PROD}, $C_R \cup C_F \cup C_{COMP} \cup C_{PROD}$) where $C_R \in V_C$ is a set of customer requirements (Felfernig et al., 2015).

A solution to a given recommendation task (V_C, V_{PROD}, $C_R \cup C_F \cup C_{COMP} \cup C_{PROD}$) is a complete assignment to the variables of (V_C, V_{PROD}) such that this assignment is matched with the constraints in ($C_R \cup C_F \cup C_{COMP} \cup C_{PROD}$) (Felfernig & Burke, 2008).

3.2. Preferred Conflicts and Relaxations

When the customer requirements (C_R) include a conflict, or no items might match the customer requirements, we have to support the customer in getting out of these situations. Constraint-based approach is interested in repair actions which indicate interesting and minimal changes to the requirements (C_R) to restore consistency. Thence, the calculation of a recommendation becomes possible (Felfernig et al., 2015).
We define now preferred relaxation and preferred conflict as explained by (Junker, 2004):

- Definition 1: a subset R of C is a relaxation of a problem $P := (\beta, C)$ iff $\beta \cup R$ has a solution.

Where β is a background containing the constraints that cannot be relaxed such as (C_F, C_{COMP}, C_{PROD}); C is a set of customer constraints (C_R). Note that only customer requirements or constraints can be relaxed.

A relaxation exists iff β is consistent. Over-constrained problems can have an exponential number of relaxations. A customer typically prefers to keep the important constraints and to relax less important ones. That means that the customer is at least able to compare the importance of some constraints. Therefore, Junker (Junker, 2004) assumes a strict partial order between the constraints of C, denoted by $\prec$. We write $c_1 \prec c_2$ iff (the selection of) constraint c_1 is preferred to (the selection of) c_2.

- Definition 2: A subset $\complement$ of C is a conflict of a problem $P:=(\beta, C)$ iff $\beta \cup C$ has no solution.

S.t. $C_R \cup C_F \cup C_{COMP} \cup C_{PROD} \cup V_C \cup V_{PROD}$ is inconsistent, iff $\exists$ a conflict $\complement : \complement = \{c_1, c_2, \ldots, c_n\} \subseteq C_R$.

There are two kinds of conflicts in a given constraint system:

- Conflict 1 involves only very important constraints.
- Conflict 2 involves less important constraints.

The user will have to resolve the first conflict, and thus, he/she will have to relax at least one important constraint. Concerning the second conflict, a less important constraint can be relaxed and the user will consider such a modification as very easy to do.

The definition of preferred relaxations and preferred conflicts can be made constructive, thus providing the basis for the explanation and relaxation algorithms. Consider a totally ordered problem $P := (\beta, C, \prec)$ s.t. β is consistent, but not $\beta \cup C$. We enumerate the elements of C in increasing order $c_1, \ldots, c_n$. We construct the preferred relaxation of P by $R_0 := \emptyset$ and

$$R_i := \begin{cases} R_{i-1} \cup \{c_i\} & \text{if } \beta \cup R_{i-1} \cup \{c_i\} \text{ has a solution} \\ R_{i-1} & \text{otherwise} \end{cases}$$

The preferred conflict of P is constructed in the reverse order. Let $C_n := C$ and

$$C_i := \begin{cases} C_{i+1} - \{c_i\} & \text{if } \beta \cup C_{i+1} - \{c_i\} \text{ has no solution} \\ C_{i+1} & \text{otherwise} \end{cases}$$

Adding a constraint to a relaxation thus corresponds to the retraction of a constraint from a conflict. As a consequence of this duality, algorithms for computing relaxations can be reformulated for computing conflicts and vice versa.

3.3. Computing Preferred Explanations

Explaining recommendations is an important aspect of any RS, it focuses on providing clarifications that justify the recommendations the user has received. Its secret is to maintain a higher degree of user confidence in the results generated by the system.

There are many types of explanations (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013), such as human style explanations. For example, we recommend movie i because it was liked by the users who rated movies j, k, m, ... very positively (j, k, m, ... are movies rated well by the active user). Item style explanations. For example, we recommend the vacation destination i because you liked the vacation destinations g, c, r, ... (g, c, r, ... are vacation destinations similar to i and rated well by the active user).

In constraint-based approach, we use the sets of constraints C_{COMP} and C_{PROD} to provide explanations of inconsistent requirements, and to justify the recommended repairs. In conformity with (Felfernig & Burke, 2008), C_{COMP} helps to ensure consistency of customer requirements and to decrease costs related to correction processes. On the other hand, C_{PROD} used to enumerate the offered set of products. Thus, the customers will learn about specific properties of the item domain and insert consistent inputs (C_R).

Algorithm QUICKXPLAIN($\mathcal{B}, \mathcal{C}, \prec$)

1. **if** *isConsistent*($\mathcal{B} \cup \mathcal{C}$) return 'no conflict';
2. **else if** $\mathcal{C} = \emptyset$ **then** return $\emptyset$;
3. **else** return QUICKXPLAIN'($\mathcal{B}, \mathcal{B}, \mathcal{C}, \prec$);

Algorithm QUICKXPLAIN'($\mathcal{B}, \Delta, \mathcal{C}, \prec$)

4. **if** $\Delta \neq \emptyset$ and not *isConsistent*($\mathcal{B}$) **then** return $\emptyset$;
5. **if** $\mathcal{C} = \{\alpha\}$ **then** return $\{\alpha\}$;
6. let $\alpha_1, \ldots, \alpha_n$ be an enumeration of $\mathcal{C}$ that respects $\prec$;
7. let k be *split*(n) where $1 \leq k < n$;
8. $\mathcal{C}_1 := \{\alpha_1, \ldots, \alpha_k\}$ and $\mathcal{C}_2 := \{\alpha_{k+1}, \ldots, \alpha_n\}$;
9. $\Delta_2 :=$ QUICKXPLAIN'($\mathcal{B} \cup \mathcal{C}_1, \mathcal{C}_1, \mathcal{C}_2, \prec$);
10. $\Delta_1 :=$ QUICKXPLAIN'($\mathcal{B} \cup \Delta_2, \Delta_2, \mathcal{C}_1, \prec$);
11. return $\Delta_1 \cup \Delta_2$;

Algorithm III- 1 Algorithm Divide-and-Conquer for Explanations (Junker, 2004)

(Junker, 2004) proposed a basic algorithm which chooses (arbitrarily) one order $\prec$ between the constraints of C, thus fixing the resulting conflict or relaxation. It then inspects one constraint after the other and determines whether it belongs to the preferred conflict or relaxation of $\prec$. It thus applies a consistency checker isConsistent(C) to a sequence of subproblems.

This basic algorithm is predicated on the following main propositions (Junker, 2004).

- Proposition 1: Let P:=(β, C, $\prec$). If β is inconsistent then the empty set is the only preferred conflict of P and P has no relaxation. If $\beta \cup$ C is consistent then C is the only preferred relaxation of P and P has no conflict.
  If C is not empty, then the algorithm follows the constructive definition of a preferred relaxation. In each step, it chooses a $\prec$- minimal element α and removes it from C. If $\beta \cup$ R $\cup\{\alpha\}$ is consistent, α is added to R. A preferred relaxation can be computed by iterating these steps.

When the first inconsistency is obtained, we have detected the best element α_{k+1} which should be taken out from the preferred relaxation. According to (Junker, 2004), α_{k+1} is the worst element of the preferred conflict. Hence, the preferred conflict is a subset of $R_k \cup\{\alpha_{k+1}\}$ and C_{n-k} is equal to $\{\alpha_{k+1}\}$.

- Proposition 2: Suppose C_1 and C_2 are disjoint and that no constraint of C_2 is preferred to a constraint of C_1:
  1. If Δ_1 is a preferred relaxation of (β, C_1, $\prec$) and Δ_2 is a preferred relaxation of ($\beta \cup \Delta_1$, C_2, $\prec$), then $\Delta_1 \cup \Delta_2$ is a preferred relaxation of (β, $C_1 \cup C_2$, $\prec$).
  2. If Δ_2 is a preferred conflict of ($\beta \cup C_1$, C_2, $\prec$) and Δ_1 is a preferred conflict of ($\beta \cup \Delta_2$, C1, $\prec$), then $\Delta_1 \cup \Delta_2$ is a preferred conflict of (β, $C_1 \cup C_2$, $\prec$).

Consequently, we divide an inconsistent problem (C) until we obtain subproblems of the form P':= (β, {α}, $\prec$), where all except one constraint are in the background. We then know that $\beta \cup \{\alpha\}$ is inconsistent.

Algorithm QUICKXPLAIN (cf. Algorithm II- 1) exploits propositions 1 and 2. It is parameterized by a split-function that selects the subproblems for a chosen order $\prec$ (see line 6).

3.3.1. Algorithm QUICKXPLAIN

Initially, algorithm QUICKXPLAIN (Divide-and-Conquer[12] for Explanations) checks whether the background β is inconsistent. If C is empty, then it returns an empty set as the only preferred conflict. A sub-procedure QUICKXPLAIN' is only called if C is a non-empty conflict and if a part of the background, namely $\beta - \Delta$ has a solution.

QUICKXPLAIN spends most of its time in the consistency checks, which will search for a solution to prove the consistency of a set X of constraints. We can diminish the number of consistency checks if we remove whole blocks of constraints. We thus divide C into subsets C_1 and C_2. If the remaining problem C_1 is inconsistent, then we can ignore all constraints in C_2.

4. The Proposed Knowledge-Based Recommender System

In the previous sections, we have introduced the e-Tourism domain, the modern functionalities of mobile devices and how they elevated this domain, thereafter we have given the full descriptions of both DATAtourisme ontology which is the representation of tourism knowledge, and the constraint-based approach as a type of knowledge-based recommendation.

Now, we coordinate these sections to bring the working application of the proposed knowledge-based recommender system for e-Tourism, by explaining how we instantiate each ingredient of constraint-based approach, the designed modules, and how they communicate with each other.

4.1. Specification of Our Constraint Satisfaction Problem

In this section, we specify our constraint satisfaction problem in accordance with DATAtourisme ontology, and the representation of the chosen approach (cf. 3. Constraint-Based Recommendation).

Formally, a constraint satisfaction problem is defined as a triple V, D, C, where:

- ✓ V is a set of variables, includes V_C, V_{PROD}.
- ✓ D is a set of the respective domains of values, 'DATAtourisme ontology'.
- ✓ C is a set of constraints, includes C_{PROD}, C_F, C_{COMP}.

- Customer Properties V_C

V_C describes possible requirements of customers; requirements are instantiations of customer properties. In our system, we have defined an instance of this set as highlighted in Table III- 2.

[12] **Divide-and-Conquer**: is a well-known Algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type.

Properties	Description
1. Type	Ex : Hotel, Restaurant, Archeological site, festival, Tour, Transport, etc.
2. Theme	Ex: Fitness Trail, Art Gallery, Cycling Tour De France, Sea Food, French Gastronomy Day, etc.
3. Means of Payment	Ex: Check, Cash, Blue Card, Eurocard Mastercard, etc.
4. Equipment	Ex: Game room, Kids Club, Wifi, Sauna, Park, Garden, Coffee Maker, etc.
5. Architectural Style	Ex : Antique, Roman, Classical, Modern, Xith Century, Xivth Century, etc.
6. Types of Kitchen	Ex : Traditional Cuisine, Halal Cuisine, European Cuisine, Asian Cuisine, Crepery, etc.
7. Takeaway	A restaurant or shop selling cooked food to be eaten elsewhere.
8. Geographic Environment	Ex: By the Sea, Close to Shops, In the Forest, etc.
9. Classification	Ex: Luxury, medium, economical.
10. Tour Type	Ex: Loop Itinerary, Open Jaw Itinerary, Round Trip.
11. Hike Path Distance	To express distance of Hiking. Ex: Small, Medium or Long Path Distance.
12. Global Path duration	Duration of tour (minutes).
13. Max Price	A number value to express maximal price of an item.
14. Location	Address and the position of an item.
15. Reduce Mobility Access	Accessible to people with reduced mobility, ex: handicap.
16. Available Language	Ex: English, French, German, Arabic, etc.

Table III- 2 Instantiations of Customer Properties

- Product Properties V_{PROD}

V_{PROD} respects main properties of items specified by the DATAtourisme ontology, such as title, description, futures, location, review value, contacts, start date and end date for events and festivals, etc.

- Products C_{PROD}

C_{PROD} represents the possible instantiations of variables in V_{PROD}. DATAtourisme team provide thesauri[13] containing all the possible instantiations of items properties in the ontology DATAtourisme, thus we have decided to use them as a specification of our C_{PROD}.

- Filter Conditions C_F

C_F defines the relationship between customer requirements C_R and products C_{PROD}. The table below shows how we define this set.

Identifier	Constraint
1.	The price of an item has to be lower (equal) then (to) the maxprice imposed by the customer.
2.	Small Path Distance Is limited between 5 and 10. Medium Path Distance Is limited between 11 and 25. Long Path Distance Is bigger than 25.
3.	luxury Classification express 5 or 4 Review Value. Medium Classification express 3 or 2 Review Value. Economical Classification express 2 or 1 Review Value.

Table III- 3 Instantiations of Filter Conditions

- Compatibility Constraints C_{COMP}

(In) Compatibility constraints can be used to model difficult constraints and to enhance solving efficiency. C_{COMP} can be modeled as shown in Table III- 4.

[13] **Thesaurus :** is a form of controlled vocabulary that seeks to dictate semantic manifestations of metadata in the indexing of content objects. See https://framagit.org/datatourisme/ontology/tree/master/thesaurus.

Properties	Compatible-with
1. Accommodation	Equipment, Geographic Environment, Means of Payment, Reduce Mobility Access, Classification, Kitchen Types.
2. Food establishment	Theme, Equipment, Geographic Environment, Means of Payment, Reduce Mobility Access, Classification, Kitchen Types, Takeaway.
3. Cultural site	Theme, Equipment, Geographic Environment, Means of Payment, Reduce Mobility Access, Classification, Architectural Style.
4. Natural heritage	Theme, Equipment, Geographic Environment, Means of Payment, Reduce Mobility Access.
5. Festival and event	Theme, Equipment, Geographic Environment, Means of Payment, Reduce Mobility Access, Classification.
6. Tour	Geographic Environment, Means of Payment, Reduce Mobility Access, Tour Type, Hike Path Distance, Global Path duration.
7. Small Path Distance	Path duration between 2 and 4 minutes.
8. Medium Path Distance	Path duration between 5 and 7 minutes.
9. Long Path Distance	Path duration between 8 and 45 minutes.
10. Transport	Means of Payment, Reduce Mobility Access.

Table III- 4 Set of Compatibilities

4.2. Architectural Design of the Proposed Recommender System

The system design satisfies the requirements of the proposed constraint-based recommender system. It includes controller components such as compatibility checker and results checker to ensure consistency of customer requirements, as well as it proposes repairs by using QUICKXPLAIN algorithm presented in the last subsection.

The system design will also capture the major functional building modules needed to understand the functioning process of our Tourism recommender system. Our knowledge-base which includes in particular the DATAtourisme ontology as knowledge about items and the several sets of constraints and variables as discussed previously.

The architectural design of the proposed system is illustrated in Figure III- 4. We describe its modules and its functioning process in the next subsections.

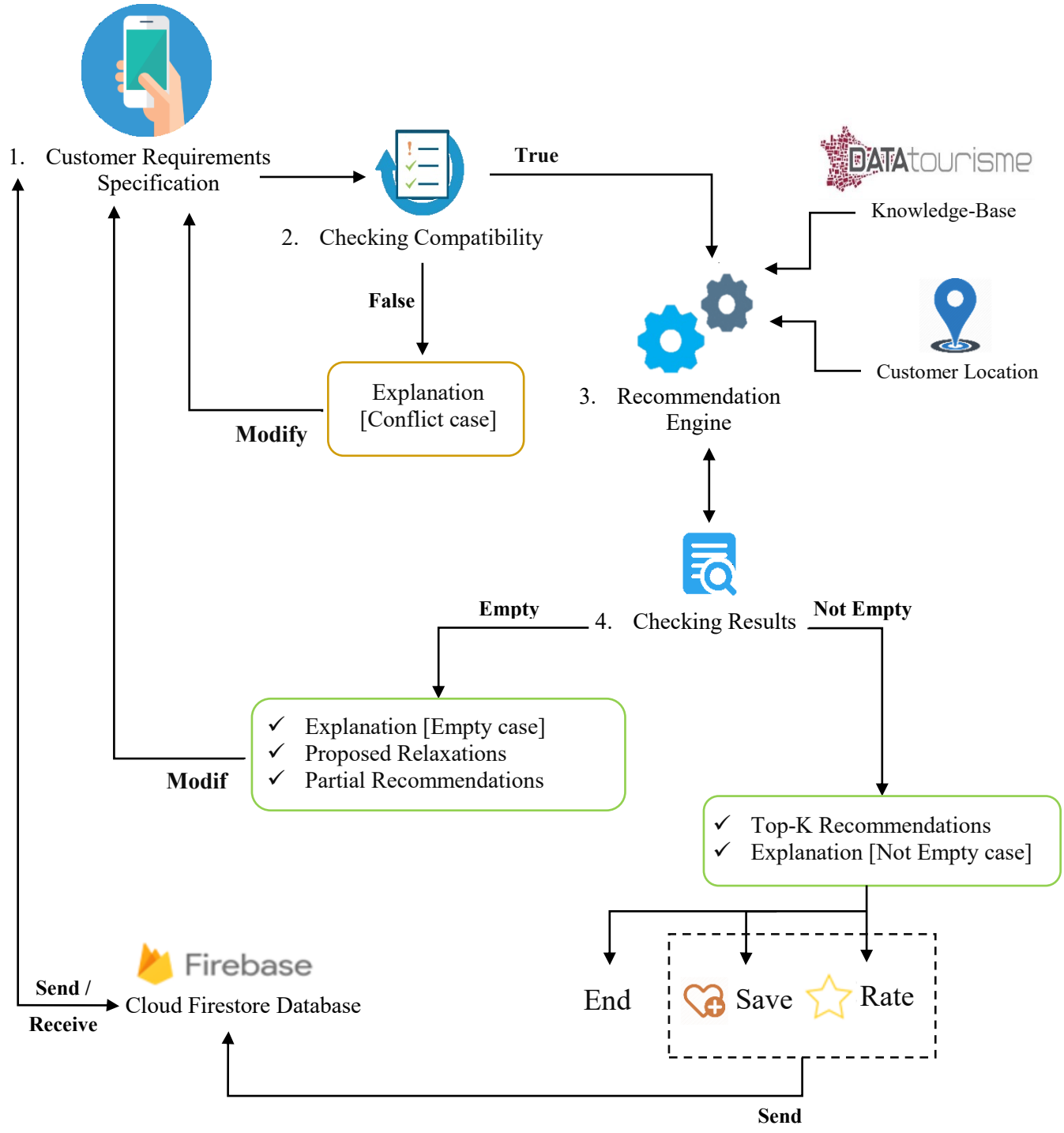

Figure III- 4 Architecture of the Proposed Knowledge-based Recommender System

4.2.1. Requirements Specification

At this initial stage, customers will select their requirements (C_R) such as Types, Themes, Means of Payment, etc. As well as they may add or modify them through an iterative set of change requests, these requirements will be saved in the Cloud Firestore Database (Firebase) which uses real-time processing to handle requests changes. We explain this concept at the end of this chapter.

4.2.2. Checking Compatibility

After the specification of customer requirements, the test of compatibility will start to check whether the specific requirements are compatible or not, by using the same concept of QUICKXPLAIN algorithm.

Compatibility checker is a procedure parametrized with three variables are: M which depicts the Type or category (Ex: Hotel, Restaurant, Festival, etc.) selected by the current customer, C_R which represent the customer requirements list, $\prec$ which is the order between constraints in C_R. we have specified that the first added constraint (by the current user) is more important then the second added, and this second constraint is more important then the third one, and so on.

M is considered as a very important constraint for which we don't have to propose a repair. Therefore, we select set of properties from C_{COMP} that are compatible with M, in order to reduce the number of compatibility checks. Then we initialize B (background knowledge) with this set. These actions sequence

Algorithm CompatibilityChecker ($M, C_R, \prec$)

1. $B := SelectSetOfCompatibility (M, C_{COMP})$;
2. if isCompatible ($B \cup C_R$) return 'True';
3. else return CompatibilityChecker' ($B, B, C_R, \prec$);

Algorithm CompatibilityChecker' ($B, \Delta, C, \prec$)

4. if $\Delta \neq \emptyset$ and not isCompatible (B) then return $\emptyset$;
5. if $C = \{\alpha\}$ then return $\{\alpha\}$;
6. let $\alpha_1, \ldots, \alpha_n$ be an enumeration of C that respects $\prec$;
7. let k be split(n) where $1 \leq k < n$;
8. $C_1 := \{\alpha_1, \ldots, \alpha_k\}$ and $C_2 := \{\alpha_{k+1}, \ldots, \alpha_n\}$;
9. $\Delta_2 :=$ CompatibilityChecker' ($B \cup C_1, C_1, C_2, \prec$);
10. $\Delta_1 :=$ CompatibilityChecker' ($B \cup \Delta_2, \Delta_2, C_1, \prec$);
11. return $\Delta_1 \cup \Delta_2$;

Algorithm III- 2 Compatibility Checker Algorithm.

- Explanation (Conflict Case)

Table III- 5 Example of Compatibility Checker, where M: "Horse Tour", C_R= {Tour Type: "Loop itinerary", Means of Payment: "Cash", Hike Path Distance: "Medium", Global Path duration: "10"}. And the order ≺: {Tour Type, Means of Payment, Global Path duration, Hike Path Distance}.

We consider that r1: Tour Type, r2: "Means of Payment", r3: "Global Path duration", r4: "Hike Path Distance".

Step	B	Δ	C	C_1	C_2	Return
1	Set of property compatibility with Tour (cf. Table III- 4).	B	{r1, r2, r3, r4}	{r1, r2}	{r3, r4}	{r4}
2	B ∪ {r1, r2}	{r1, r2}	{r3, r4}	{r3}	{r4}	{r4} ∪ ∅
3	B ∪ {r3}	{r3}	{r4}	/	/	{r4}
4	B ∪ {r4}	{r4}	{r3}	/	/	∅

Table III- 5 Example of Compatibility Checker

The constraint that causes the conflict case is "Hike Path Distance"; thus, the system gives the following explanation to the customer, and allows the customer to choose the right repair.

Constraint	Compatible-with
Long Path Distance	Path duration between 8 and 45 minutes.
Medium Path Distance	Path duration between 5 and 7 minutes.

Table III- 6 Explanation (Conflict case).

4.2.3. Recommendation Engine

When customer requirements become consistent, the recommendation engine will execute the recommendation task (cf. 3.1. Recommendation Task). Finding items that match with the current requirements.

In addition, our engine will sort the list of recommendation according to the current user's location. The nearest item will be at the top of recommendation list.

4.2.4. Checking Results

Result Checker is a procedure Not much different than compatibility checker. Just that the Result Checker will look for the constraint that caused the empty result case, and accordingly we propose a relaxation and give a partial recommendation which respects only the consistent requirements.

Algorithm ResultChecker ($B, C_R, \prec$)

1. if hasResult(B) return 'Not Empty';
2. else return ResultChecker' (B, B, C, $\prec$);

Algorithm ResultChecker' ($B, \Delta, C, \prec$)

3. if $\Delta := \emptyset$ and Not hasResult(B) then return $\emptyset$;
4. if $C = \{\alpha\}$ then return $\{\alpha\}$;
5. let $\alpha_1, \ldots, \alpha_n$ be an enumeration of C that respects $\prec$;
6. let k be split(n) where $1 \leq k < n$;
7. $C_1 := \{\alpha_1, \ldots, \alpha_k\}$ and $C_2 := \{\alpha_{k+1}, \ldots, \alpha_n\}$;
8. $\Delta_2 :=$ ResultsXPLAIN' ($B \cup C_1, C_1, C_2, \prec$);
9. $\Delta_1 :=$ ResultsXPLAIN' ($B \cup \Delta_2, \Delta_2, C_1, \prec$);
10. return $\Delta_1 \cup \Delta_2$;

Algorithm III- 3 Results Checker Algorithm.

4.2.5. Cloud Firestore data model

Cloud[14] Firestore is Firebase's newest flagship database for mobile and web apps. It is a successor to the Realtime Database with a new and more intuitive data model. Cloud Firestore is richer, faster, and more scalable than the Realtime Database ("Realtime Database vs. Cloud Firestore," 2018; Sharma, 2018).

Firestore is NoSQL data model (supports any data structure), we store data in documents that contain fields mapping to values. These documents are stored in collections, which are containers for our documents. Documents support all possible data types, from simple strings and numbers, to complex, nested objects.

Additionally, querying in Cloud Firestore is expressive, efficient, and flexible. we can sort, filter our queries to paginate our results. Moreover, adding real-time listeners to our app notify us with a data snapshot whenever the data our client apps are listening to changes, retrieving only the new changes.

Our data model encompasses different collections and documents which are fully illustrated in Figure III- 6.

[14] See https://fr.wikipedia.org/wiki/Cloud_computing

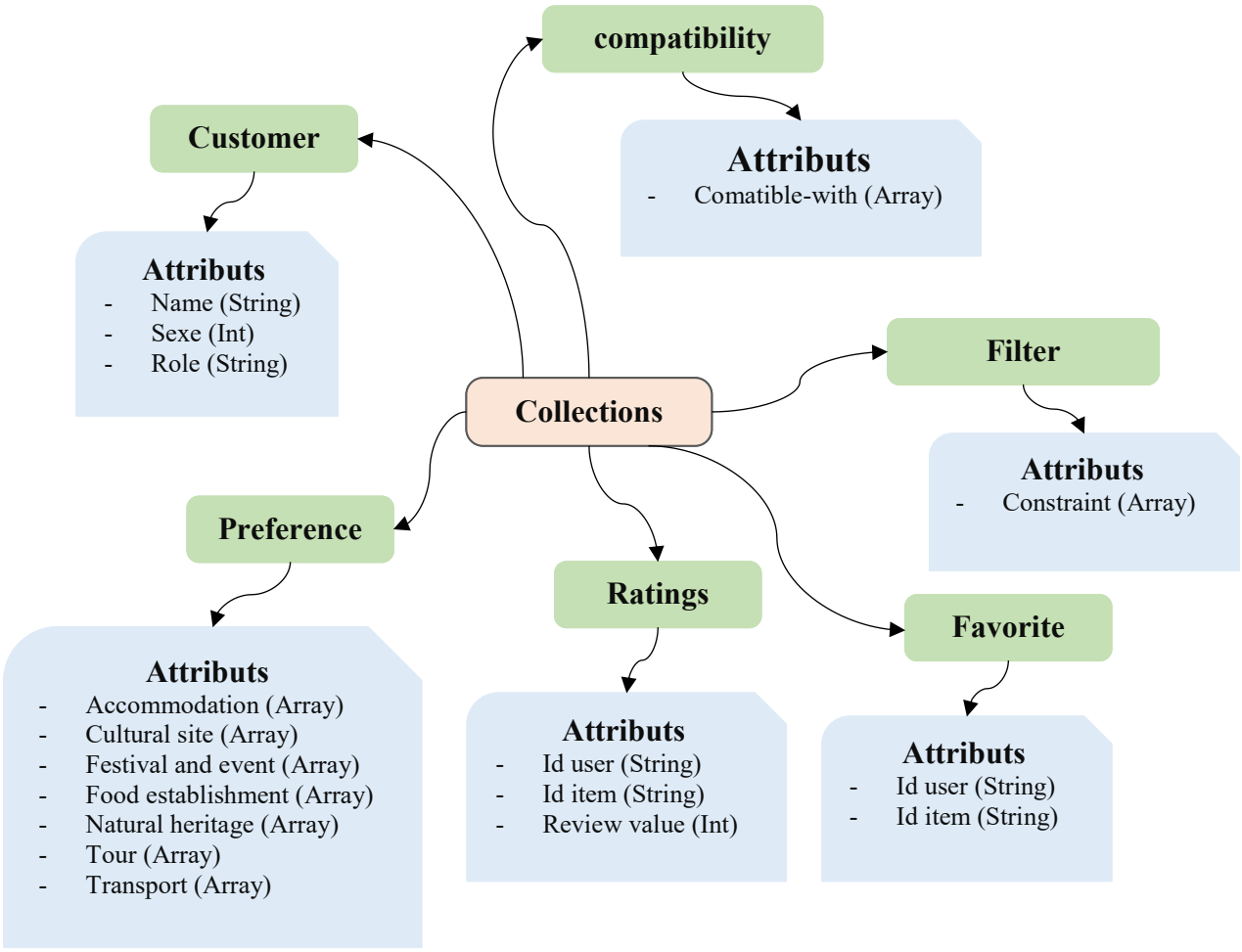

Figure III- 5 Cloud Firestore Data Model

5. Conclusion

This chapter has introduced our knowledge-based recommender system (more precisely, constraint-based recommender system) by designing an architecture and its functioning process. To achieve the recommendation task, this recommender system uses several algorithms for explanation, relaxation and recommendation. The realization of this system will be established in the following chapter.

CHAPTER IV
Realization and Testing

« talk is cheap. show me the code. »
Linus Torvalds

1. Introduction

This chapter aims to implement the proposed architecture of our Constraint-Based Recommender System (cf. Chapter III) for showing the obtained results and testing the proposed recommendations in several execution scenarios.
The application execution will be discussed after presenting the tools and technologies that are used in this implementation. Finally, a summary of advantages and limits of our application will be listed at the end of this chapter.

2. Development Environment

In order to realize the different functional building components of the proposed architectural design of our constraint-based recommender system (cf. Chapter III), we have used a set of new tools and technologies, namely:

2.1. Flutter

Flutter[15] is Google's open-source user interface (UI) toolkit for building high-performance, natively-compiled applications for mobile, web, desktop, and even embedded devices from a single codebase.

Figure IV- 1 Flutter Logo

- **2015**, The first version of Flutter was unveiled at the 2015 Dart developer summit. It was known as codename "Sky" and ran on the Android operating system("Google's Dart language on Android aims for Java-free, 120 FPS apps," 2015).
- **December 4th, 2018**, Flutter 1.0 was released at the Flutter Live event, denoting the first "stable" version of the Framework .
- **February 26th, 2019**, publication of Flutter 1.2 stable version,.
- **May 7th, 2019**, At Google I/O 2019 developer conference, Google launched version 1.5 of Flutter, its open source mobile UI framework that helps developers build native interfaces for Android and iOS. But that's no longer true: The mobile framework is now a multi-platform UI framework, supporting the web, desktop, mobile, and even embedded devices. Flutter's mission has expanded to building "the best framework for developing beautiful experiences for any screen."("Flutter SDK releases,").

2.1.1. The Engine Architecture

The main components of Flutter include: Dart platform, Flutter engine, Foundation library, and Design-specific widgets, ("Flutter SDK releases,")summarizes the description of each components as follows.

[15] See: https://flutter.dev

- Dart platform: Flutter apps are written in the Dart[16] language and make use of many of the language's more advanced features. A notable feature of the Dart platform is its support for "hot reload" where modifications to source files can be injected into a running application.
- Flutter engine: Flutter's engine, written primarily in C++, provides low-level

Figure IV- 2 Dart Logo

rendering support using Google's Skia graphics library. It is a portable runtime for hosting Flutter applications. The Flutter Engine implements Flutter's core libraries, including animation and graphics, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain.

- Foundation library: The Foundation library, written in Dart, provides basic classes and functions which are used to construct applications using Flutter.
- Design-specific widgets: The Flutter framework contains two sets of widgets which conform to specific design languages. Material Design widgets implement Google's design language of the same name, and Cupertino widgets imitate Apple's iOS design.

2.1.2. Benefits of Flutter

The major benefits of flutter are:

- High productivity: Develop a single codebase for cross-platform, this can certainly save you time and resources. This means, do more with less code, and with a modern, expressive language and a declarative approach.
- Highly-customized, beautiful user experiences: Benefit from a wealthy set of Material Design and Cupertino widgets built using Flutter's own framework. Moreover, Realize custom, beautiful, brand-driven designs.

2.2. VScode

Visual Studio Code is a source-code editor, developed by Microsoft for Windows, Linux and macOS, so we can hit the ground running, no matter the platform. It includes support for debugging, syntax highlighting, intelligent code completion, embedded Git control and GitHub.

Figure IV- 3 Visual Studio Code

[16]See https://en.wikipedia.org/wiki/Dart_(programming_language)

In the Stack Overflow 2019 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool, with 50.7% of 87,317 respondents claiming to use it. It supports major programming language such as web programming language (PHP, JavaScript, etc.) and mobile programing language (Dart, java, etc.).

2.3. DATAtourisme API

The "DATAtourisme" system is enriched with new tools in order to help users to exploit the tourist data made available in Open Data. The Datatourism API[17] is a PHP library that allows the user to easily query a semantic database containing tourist data from the DATAtourisme platform. It relies on a GraphQL query language whose data schema is based on the ontology DATAtourisme.

Figure IV- 4 DATAtourisme Logo

With regard to implement the API, the user must:

- Recover DATAtourisme data in semantic format on the diffuser platform.
- Load semantic data into a semantic database (triplestore) with a SPARQL access point.
- Use the DATAtourisme API to submit GraphQL queries to the database and retrieve the results.

This API translates our GraphQL queries into a SPARQL query, in the API response of the executed query we will find (in addition to the results) the SPARQL query that was used. The Figure IV- 5 shows the list of available fields includes this SPARQL query, for more information see the documentation[18] of DATAtourisme API.

```
{
  poi {
    total # <= total number of results
    results {
      # ... <= Fields and subfields of a POI
    }
    query # <= SPARQL query generated and executed by the API
  }
}
```

Figure IV- 5 List of Available Fields

[17] See https://datatourisme.frama.io/api/#/start/getting_started
[18] See https://datatourisme.frama.io/api/#/api/fields

2.4. Semantic Database - Blazegraph

Blazegraph is an open-source semantic database management system. It provides a SPARQL access point that allows the API to query its contents to respond to GraphQL queries. Blazegraph is a product developed by Systap since 2006, formerly known as Bigdata.
Regarding the launch of the Blazegraph server is done simply by the command "java -jar blazegraph.jar", the Blazegraph server is then launched, its administration interface is accessible from the address "http: // localhost: 9999/".

Figure IV- 6 Blazegraph Logo

2.5. Composer

Composer[19] is a tool for dependency management in PHP. It allows us to declare the libraries our project depends on and it will manage (install/update) them. Composer requires PHP 5.3.2+ to run, and it works equally well on Windows, Linux and macOS (multi-platform).

Figure IV- 7 Composer Logo

Composer offers several parameters such as :

- **install**: install all libraries from composer.json. it's the command to use to download all php repository dependencies.
- **update**: update all libraries from composer.json, according to the allowed versions mentioned into it.
- **require**: add the library in parameter to the file composer.json, and install it
- **remove**: uninstall a library and remove it from composer.json.

2.6. PHP 7

PHP[20] (an acronym for: Hypertext Preprocessor) is a scripting language that's generally used in server-side web development. It was created by Rasmus Lerdorf in 1994.

[19] See https://getcomposer.org/
[20] See https://www.php.net/

PHP code may be executed with a command line interface (CLI), and it is usually processed by a PHP interpreter implemented as a module in a web server or as a Common Gateway Interface (CGI) executable. PHP 7 is the latest stable release.

Figure IV- 8 PHP Logo

2.7. Apache

Apache HTTP Server[21] is free and open-source cross-platform web server software. It was launched in 1995 and it has been the most popular web server on the Internet since April 1996. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

Figure IV- 9 Apache Logo

2.8. Firebase

Firebase[22] is a Backend-as-a-Service (BaaS) application development platform developed by Firebase, Inc. in 2011, then acquired by Google in 2014(PROTALINSKI, 2019). It offers hosted backend services such as a real-time database, authentication, cloud storage, machine learning, remote configuration, and hosting for your static files.

Figure IV- 10 Firebase Logo

In October 2018, the Firebase platform has 18 products, which are used by 1.5 million apps("Flutter (software)," 2019). In the following sub-sections, we provide a brief description of some firebase products.

2.8.1. Firebase Authentication

Firebase Authentication is a service that can authenticate users using only client-side code. It supports social login providers Facebook, Twitter, GitHub and Google.

[21] See https://httpd.apache.org/
[22] See https://firebase.google.com/

2.8.2. Realtime Database

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client.

2.8.3. Cloud Firestore

Cloud Firestore is a cloud-hosted, NoSQL database that your iOS, Android, and web apps can access directly via native SDKs. Cloud Firestore is also available in native Node.js, Java, Python, and Go SDKs, in addition to REST and RPC APIs.

2.8.4. Firebase Analytics

Google Analytics for Firebase helps you understand how people use your iOS or Android app. The SDK automatically captures a number of events and user properties and also allows you to define your own custom events to measure the things that uniquely matter to your business.

2.8.5. Firebase Prediction

Firebase Predictions applies machine learning to your analytics data to create dynamic user segments based on the predicted behavior of users in your app. By default, Predictions provides two types of predictions:

- **churn**, which helps you identify users likely to stop using your app.
- **spend**, which helps you find users who are likely to spend money in your app. You can also create your own predictions based on custom conversion Analytics events that you collect in your app.

2.9. Leaflet Map

Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps, it has all the mapping features most developers ever need. Leaflet is designed with simplicity, performance and usability in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, it has a readable source code that is a joy to contribute to("Technical Overview,").

Figure IV- 11 Leaflet maps Logo

3. Execution and Results

In this section, we show the execution process of our application by testing the obtained results in two different scenarios (without and with conflicts in user's requirements).

3.1. Registration / Authentication

Application Home Page Figure IV- 12 is the first page of our application (FRANCE Advisor); it allows users to visualize the registration and authentication pages (Figure IV- 13 and Figure IV 14 , respectively).

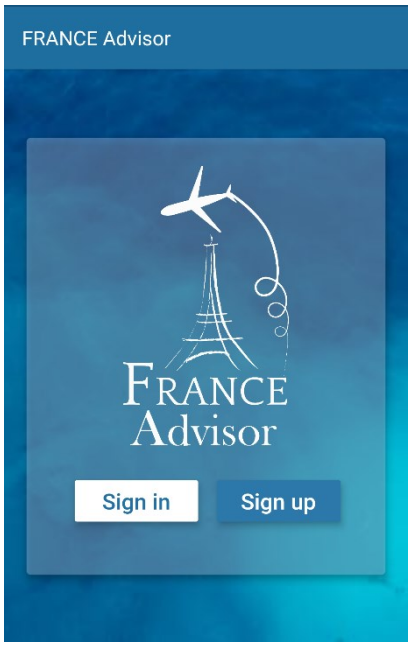

Figure IV- 14 Application Home Page

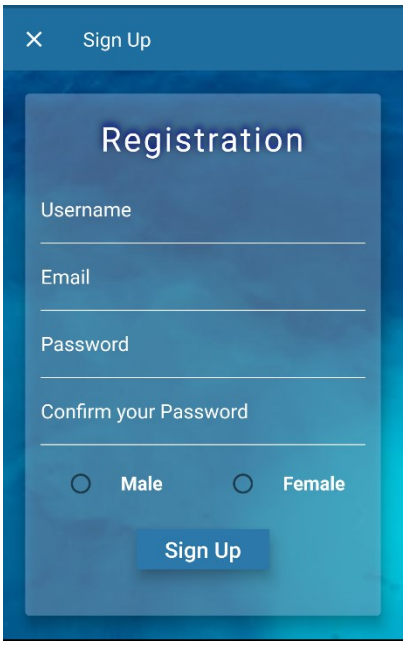

Figure IV- 13 Registration Page

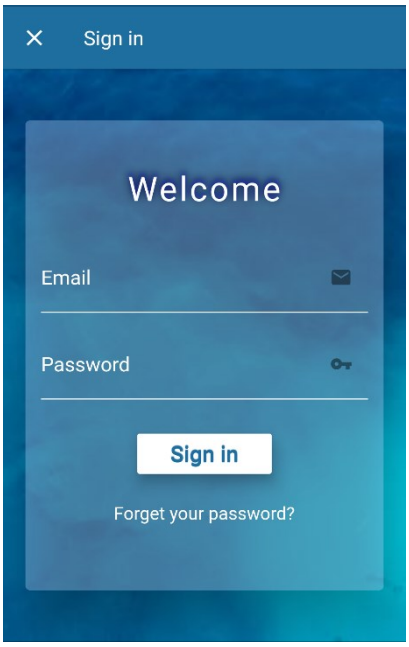

Figure IV- 12 Authentication Page

3.2. User Profile

The user profile Figure IV- 15 allows the users to specify their preferences and needs, it contains all possible requirements for a Tourist such as Accommodation, Food establishment, Cultural site, etc.

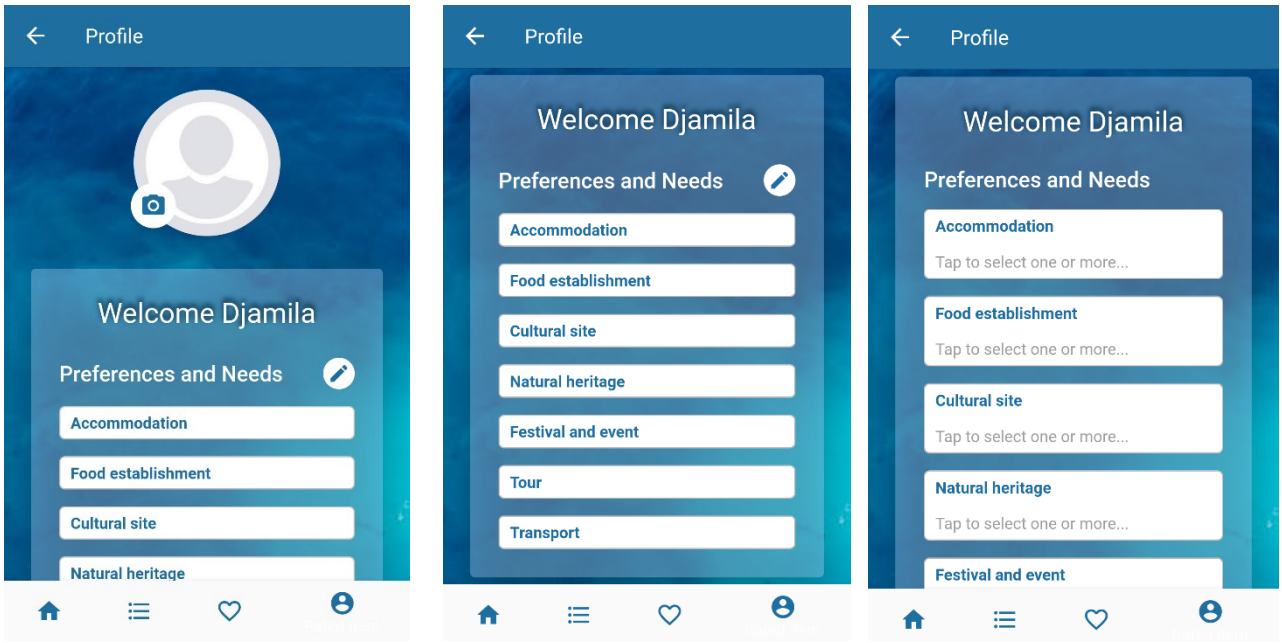

Figure IV- 15 User Profile Page

3.2.1. Example of User Preferences / Needs

Figure IV- 16 displays a representations sequence to express the different preferences/needs. For instance, the user "Djamila", has as preferences: Hotel, Fluvial or sea tour, Museum, etc.

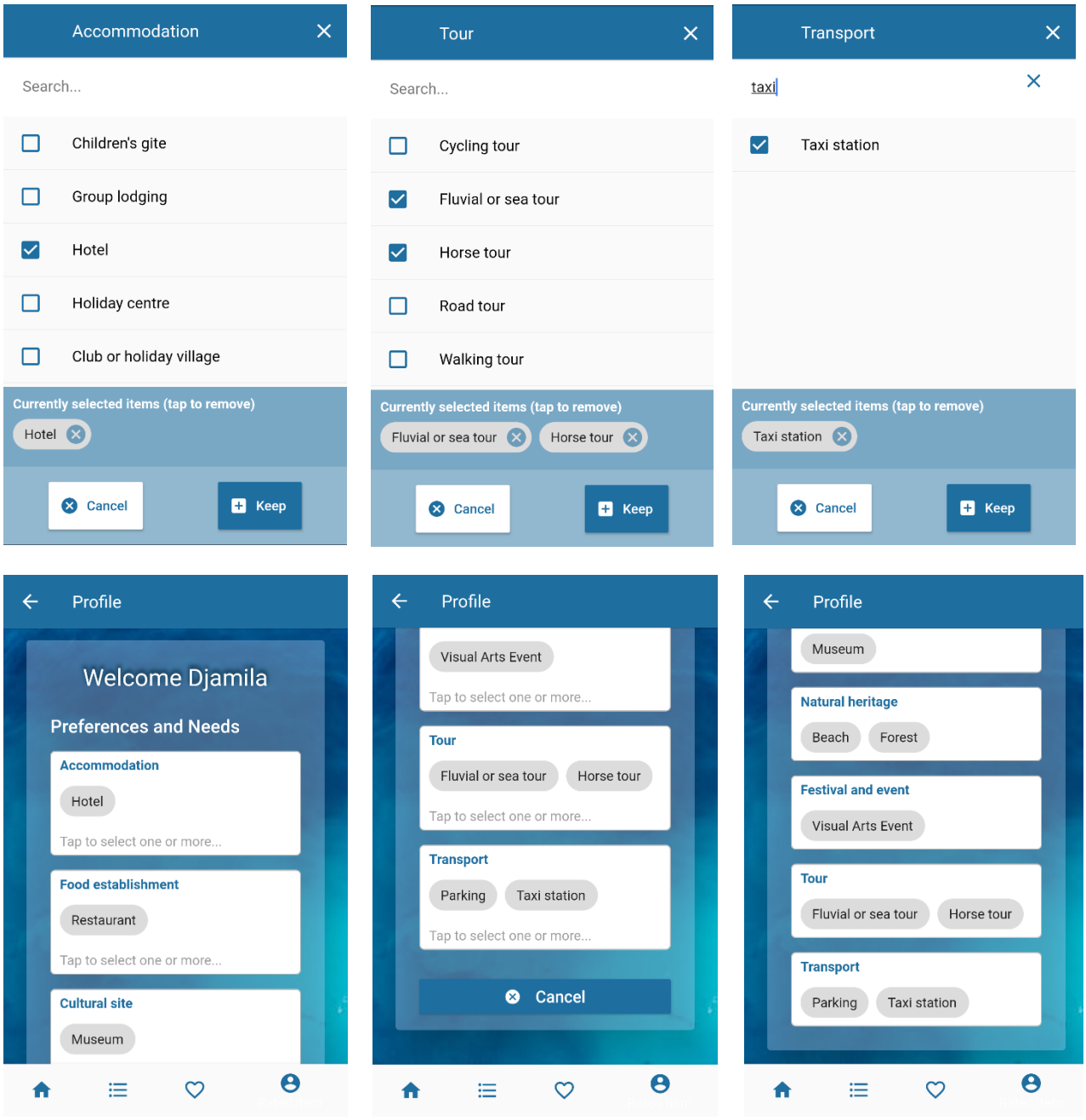

Figure IV- 16 Example of User Profile

These steps are intended to explicitly define user preferences / needs, and narrow down available Tourism services. Figure IV- 17 visualizes selected services by the current user.

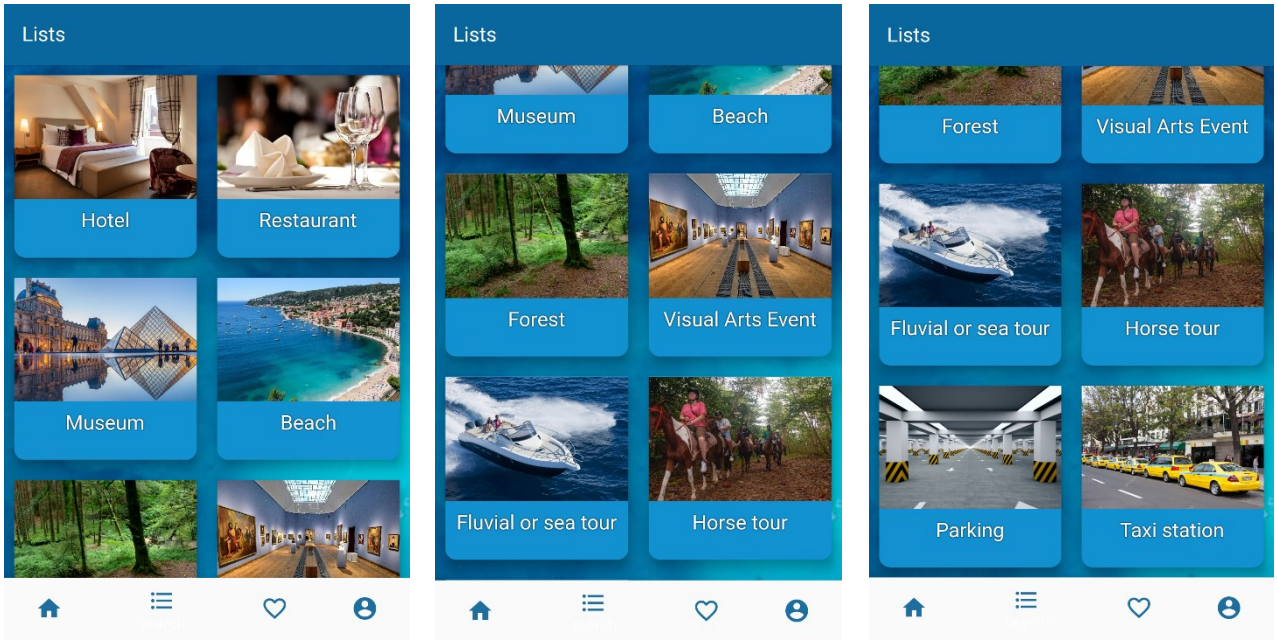

Figure IV- 17 The selected Tourism services List.

3.3. Tourism services

Each tourism service contains Guidance which show how many items are in the current service, how many items are matched with the specific filters, and help the user to select the best filters choices. Figure IV- 18 depicts some examples of Tourism services with default results.

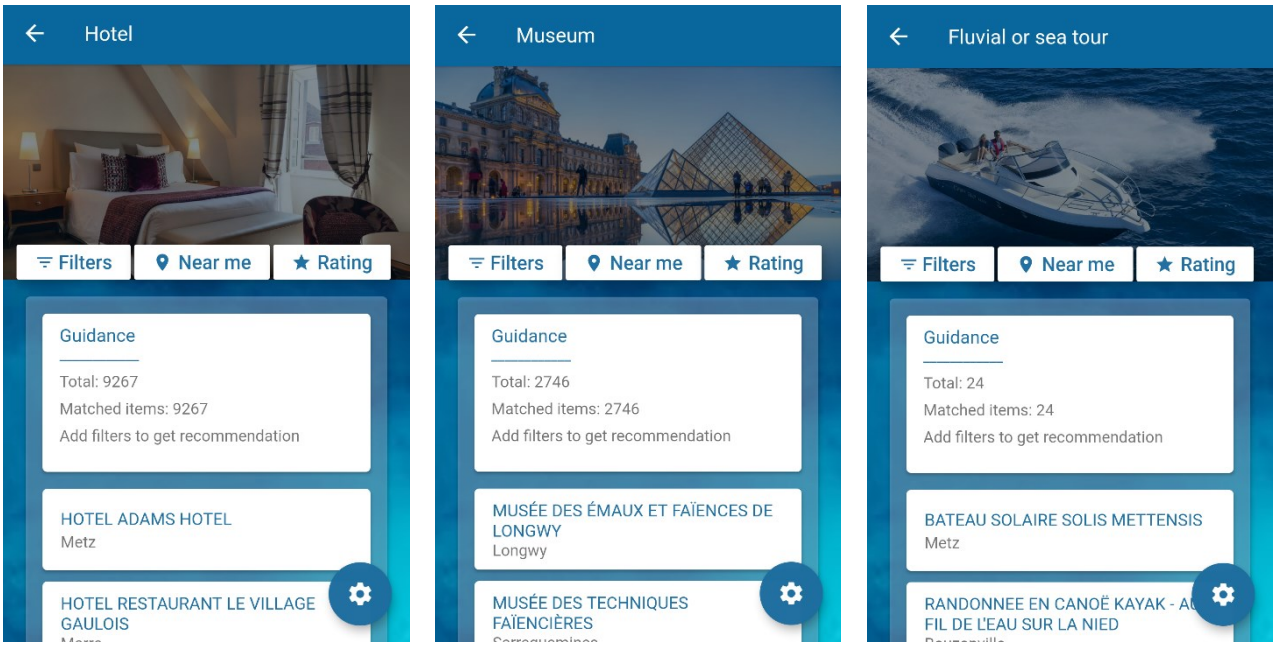

Figure IV- 18 Tourism Services Exemples

3.4. Execution Scenarios

In order to present the execution capabilities of our application (Constraint-based recommender system), we have processed for two use's scenarios. The first scenario concerns the execution of the user's request without conflict in its requirements. In contrast, the second one deals with the appearance of conflicts into user's requirements and how these can be explained (cf. Chapter III).

3.4.1. Scenario 1 (without conflict)

Figure IV- 19 depicts example scenario without conflict case. A list of recommended items which respects all user constraint (Equipped with, Geographic environment, means of payment, and Mobility access). We usually choose the top 20 items.

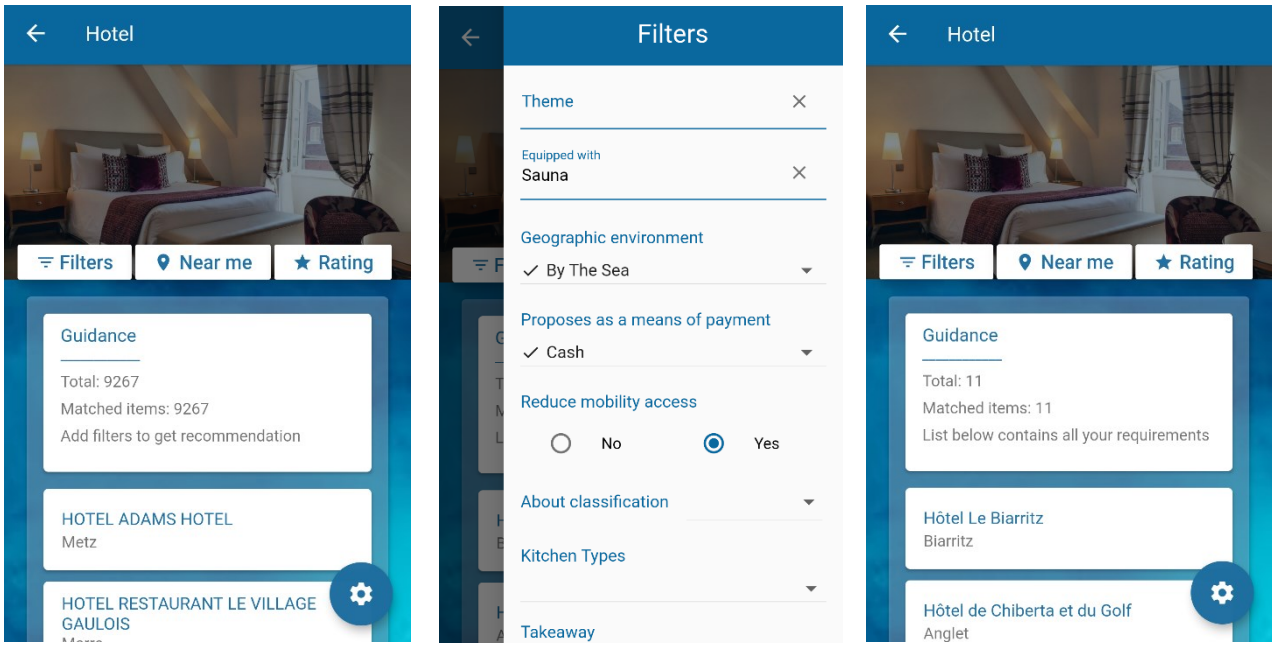

Figure IV- 19 Scenario 1 (Without conflict)

3.4.2. Scenario 2 (with conflict)

Figure IV- 20 shows an example of user's requirements that containing a conflict (Theme incompatible with Accommodation).

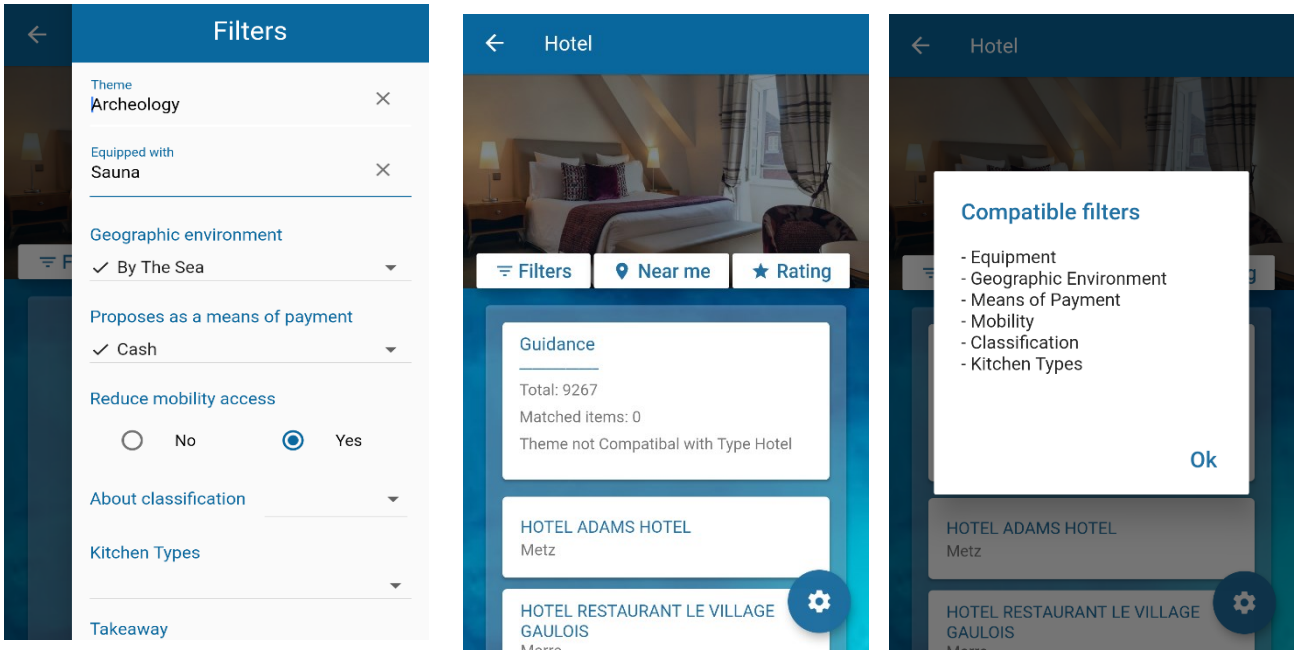

Figure IV- 20 Scenario 2 (With conflict)

- Details on recommended items

We give here (Figure IV- 21) an example of hotel which is one of the results finding through the previous requirements (or constraints).

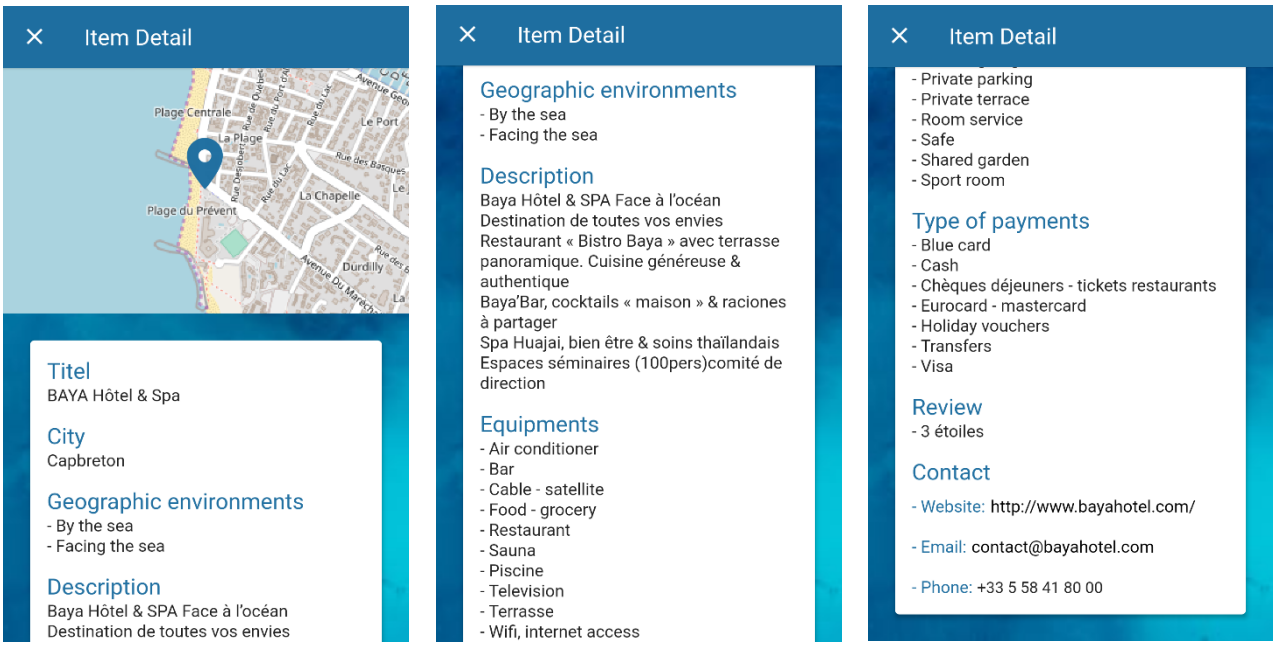

Figure IV- 21 Example Hotel

4. Advantages and Limits

Our experience has enhanced the major advantages of Knowledge-based recommender systems (KBRS) found in the literature (such as, No Cold-start issue). However, we can note some limits of the current application (FRANCE Advisor) in order to be tackled in the near future.

4.1. Advantages

- KBRS (precisely, Constraint-Based Recommender System) do not suffer from either the "cold start problem" with a new user (cf. Chapter 1), or "rump-up problem" with a new item.
- They, do not face a user privacy issue that is usually caused by the user information collection phase.
- Explicit dynamic user profile (real-time preferences).
- In case of a conflict within user's requirements, the provided explanations may have many advantages, mainly, enhancing users trust to the recommender system (reliability) by helping them to make coherent constraints.
- Provide suggestions that allows users to know the offering relevant items, and define what they want.
- Last but not least, our recommendation system is multiplatform application with can be executed either on a mobile (android/IOS), web, or desktop platforms

4.2. Limits

- Our proposed sets of constraints are not sufficient to represent all knowledge about the matching between the items and user's needs. Generally, they are made by domain experts.
- In its current version, our application does not sort results by the nearest items to the current user's location, or by the values of their ratings.

5. Conclusion

In this chapter, we have presented the realization of our recommendation system in order to show its feasibility on the DATAtourisme ontology as a recent knowledge-base. For a reproducible implementation, we have mentioned the different tools and technologies used in this application. The execution has been displayed according two different scenarios showing all possibilities of user's requirements cases.
The chapter has been concluded by giving some advantages of our application and its limits for eventual research perspectives.

GENERAL CONCLUSION

General Conclusion

Summary and contributions

Our graduation project was to design and build a knowledge-based recommender system for the Tourism domain, by using the "DATAtourisme" ontology as a new and rich knowledge-base in this domain. Our recommender system can work on mobile, web and desktop platforms. Thanks to the gained benefits by using the Google project "Flutter".

To carry out this work, we had to go through several phases. At first, we did research on the field of study, this research focused on generalities of recommender systems and the basic designed models for them, such as Collaboratif filtering and Content-based models. Then we did a research on the field of knowledge-based recommender systems, we explained what is knowledge, knowledge-based, and we clarified the main forms to represent knowledge. Moreover, we gave a brief explanation of the two types of knowledge-based recommender systems.

Concerning the design phase, we gave a full description of DATAtourisme ontology, and constraint-based recommendation as an adopted approach. This phase ended by highlighting our architectural design for the proposed recommender system with appropriate descriptions for its modules inclosing several formal algorithms.

In implementation phase, we presented our development environment by indicating various tools and technologies which are used, such as Flutter, API DATAtourisme, firebase, etc. then we proved the execution process of our application by depicting the obtained results while presenting some screenshots (interfaces) according to several uses scenarios. Finally, we cited some advantages and limits of this research work.

As far as we can tell is that the Knowledge-based recommender systems requires a big efforts in term of system design, they use a lot of techniques (not simple as in content-based or collaborative-based filtering). In addition, Knowledge-based recommender systems help users to explore and understand the domain knowledge. In this kind of recommender system, users are an integral part of recommendation process knowledge, by developing their information needs during their frequent interactions with the recommender system.

This project was a good opportunity to discover and deepen our knowledge domain, recommender systems, and to push our skills by using further new technologies.

Future work

Nevertheless, any large-scale project requires considerable effort and continuous improvement. For our case, some points remain to be explored, among them we can indicate:

- First, evaluation of the proposed recommender system using standard metrics (such as recall, precision and F-measure).

- Construction of an ontological representation of the user's profile in order to achieve the matching between the ontologies of items (tourism services) and user's needs.
- Finally, we aim to perform some non-functional requirements of this software application (like as performance and maintainability).

Annex

Our Dart Classes to read the ontology DATAtourisme (Json files):

- Model_results.dart

```
class Tourism {
  Data data;
  Tourism({this.data});
  factory Tourism.fromJson(Map<String, dynamic> parsedJson) {
    var mydata = Data.fromJson(parsedJson['data']);
    return Tourism(data: mydata);
  }
}

//data
class Data {
  Poi poi;

  Data({
    this.poi,
  });
  factory Data.fromJson(Map<String, dynamic> parsedJson) {
    var mypoi = Poi.fromJson(parsedJson['poi']);
    return Data(poi: mypoi);
  }
}

//poi
class Poi {
  List<Results> results;
  int total;

  Poi({this.results, this.total});

  factory Poi.fromJson(Map<String, dynamic> parsedJson) {
    var list = parsedJson['results'] as List;
    List<Results> resultsList = list.map((i) => Results.fromJson(i)).toList();

    var totalFromJson = parsedJson['total'];
    int totalList = totalFromJson;

    return Poi(results: resultsList, total: totalList);
  }
}
```

```
//results
class Results {
  List<String> rdfslabel;
  List<HasDescription> hasDescription;
  List<HasArchitecturalStyle> hasArchiStyle;
  List<ProvidesCuisineOfType> providesCuisineOfType;
  List<HasTheme> hasTheme;
  List<Hascontact>hascontact;
  List<IsLocatedAt> isLocatedAt;
  List<HasReview> hasReview;
  List<Offers> offers;
  List<IsEquippedWith> isEquippedWith;
  Results(
      {this.rdfslabel,
      this.hasDescription,
      this.hasArchiStyle,
      this.providesCuisineOfType,
      this.hasTheme,
      this.hascontact,
      this.isLocatedAt,
      this.hasReview,
      this.offers,
      this.isEquippedWith});

  factory Results.fromJson(Map<String, dynamic> parsedJson) {
    var rdfslabelFromJson = parsedJson['rdfs_label'];
    List<String> rdfslabelList = rdfslabelFromJson.cast<String>();

    var description = parsedJson['hasDescription'] as List;
    List<HasDescription> descriptionList;
    if (parsedJson['hasDescription'] != null) {
      descriptionList =
          description.map((i) => HasDescription.fromJson(i)).toList();
    }

    var archiStyle = parsedJson['hasArchitecturalStyle'] as List;
    List<HasArchitecturalStyle> archiStyleList;
    if (parsedJson['hasArchitecturalStyle'] != null) {
      archiStyleList =
          archiStyle.map((i) => HasArchitecturalStyle.fromJson(i)).toList();
    }

    var pcuisineOfType = parsedJson['providesCuisineOfType'] as List;
    List<ProvidesCuisineOfType> pcuisineOfTypeList;
    if (parsedJson['providesCuisineOfType'] != null) {
      pcuisineOfTypeList =
```

```
          pcuisineOfType.map((i) =>
ProvidesCuisineOfType.fromJson(i)).toList();
    }

    var theme = parsedJson['hasTheme'] as List;
    List<HasTheme> themeList;
    if (parsedJson['hasTheme'] != null) {
      themeList =
          theme.map((i) => HasTheme.fromJson(i)).toList();
    }

    var contact = parsedJson['hasContact'] as List;
    List<Hascontact> contactList;
    if (parsedJson['hasContact'] != null) {
      contactList = contact.map((i) => Hascontact.fromJson(i)).toList();
    }

    var isLocatedAt = parsedJson['isLocatedAt'] as List;
    List<IsLocatedAt> isLocatedAtList = [];
    isLocatedAtList = isLocatedAt.map((i) =>
IsLocatedAt.fromJson(i)).toList();

    var hasReview = parsedJson['hasReview'] as List;
    List<HasReview> hasReviewList = [];
    hasReviewList = hasReview.map((i) => HasReview.fromJson(i)).toList();

    var offers = parsedJson['offers'] as List;
    List<Offers> offersList = [];
    offersList = offers.map((i) => Offers.fromJson(i)).toList();

    var isEquippedWith = parsedJson['isEquippedWith'] as List;
    List<IsEquippedWith> isEquippedWithList =
        isEquippedWith.map((i) => IsEquippedWith.fromJson(i)).toList();

    return new Results(
        rdfslabel: rdfslabelList,
        hasDescription: descriptionList,
        hasArchiStyle: archiStyleList,
        providesCuisineOfType: pcuisineOfTypeList,
        hasTheme: themeList,
        hascontact: contactList,
        isLocatedAt: isLocatedAtList,
        hasReview: hasReviewList,
        offers: offersList,
        isEquippedWith: isEquippedWithList);
  }
}
```

```
//description
class HasDescription {
  List<String> shortDescription;

  HasDescription({
    this.shortDescription,
  });

  factory HasDescription.fromJson(Map<String, dynamic> parsedJson) {
    var shortDescriptionFromJson;
    List<String> shortDescriptionlList;
    if (parsedJson != null) {
      shortDescriptionFromJson = parsedJson['shortDescription'];
      shortDescriptionlList = shortDescriptionFromJson.cast<String>();
    }
    return new HasDescription(
      shortDescription: shortDescriptionlList,
    );
  }
}

//archi style
class HasArchitecturalStyle {
  List<String> rdfslabelArchiStyle;

  HasArchitecturalStyle({
    this.rdfslabelArchiStyle,
  });

  factory HasArchitecturalStyle.fromJson(Map<String, dynamic> parsedJson) {
    var rdfslabelArchiStyleFromJson;
    List<String> rdfslabelArchiStyleList;
    if (parsedJson['rdfs_label'] != null) {
      rdfslabelArchiStyleFromJson = parsedJson['rdfs_label'];
      rdfslabelArchiStyleList = rdfslabelArchiStyleFromJson.cast<String>();
    }
    return new HasArchitecturalStyle(
      rdfslabelArchiStyle: rdfslabelArchiStyleList,
    );
  }

}
//cuisine type
class ProvidesCuisineOfType {
  List<String> rdfslabelCuisineOfType;
```

```
  ProvidesCuisineOfType({
    this.rdfslabelCuisineOfType,
  });

  factory ProvidesCuisineOfType.fromJson(Map<String, dynamic> parsedJson) {
    var rdfslabelCuisineOfTypeFromJson;
    List<String> rdfslabelCuisineOfTypeList;
    if (parsedJson['rdfs_label'] != null) {
      rdfslabelCuisineOfTypeFromJson = parsedJson['rdfs_label'];
      rdfslabelCuisineOfTypeList =
rdfslabelCuisineOfTypeFromJson.cast<String>();
    }
    return new ProvidesCuisineOfType(
      rdfslabelCuisineOfType: rdfslabelCuisineOfTypeList,
    );
  }

}


//hastheme
class HasTheme {
  List<String> rdfslabeltheme;

  HasTheme({
    this.rdfslabeltheme,
  });

  factory HasTheme.fromJson(Map<String, dynamic> parsedJson) {
    var rdfslabelthemeFromJson;
    List<String> rdfslabelthemeList;
    if (parsedJson != null) {
      rdfslabelthemeFromJson = parsedJson['rdfs_label'];
      rdfslabelthemeList = rdfslabelthemeFromJson.cast<String>();
    }
    return new HasTheme(
      rdfslabeltheme: rdfslabelthemeList,
    );
  }
}


//hascontact
class Hascontact {
  List<String> foafhomepage;
  List<String> schematelephone;
  List<String> schemaemail;

  Hascontact({
```

```
    this.foafhomepage,
    this.schematelephone,
    this.schemaemail,
  });

  factory Hascontact.fromJson(Map<String, dynamic> parsedJson) {

    var foafhomepageFromJson;
    List<String> foafhomepageList;
    if (parsedJson['foaf_homepage'] != null) {
      foafhomepageFromJson = parsedJson['foaf_homepage'];
      foafhomepageList = foafhomepageFromJson.cast<String>();
    }

    var schematelephoneFromJson;
    List<String> schematelephoneList;
    if (parsedJson['schema_telephone'] != null) {
      schematelephoneFromJson = parsedJson['schema_telephone'];
      schematelephoneList = schematelephoneFromJson.cast<String>();
    }

    var schemaemailFromJson;
    List<String> schemaemailList;
    if (parsedJson['schema_email'] != null) {
      schemaemailFromJson = parsedJson['schema_email'];
      schemaemailList = schemaemailFromJson.cast<String>();
    }


    return new Hascontact(
      foafhomepage: foafhomepageList,
      schematelephone:schematelephoneList ,
      schemaemail: schemaemailList,

    );
  }
}

//isLocatedAt
class IsLocatedAt {
  List<SchemaAddress> schemaAddress;
  List<Schemageo> schemageo;

  IsLocatedAt({this.schemaAddress, this.schemageo});

  factory IsLocatedAt.fromJson(Map<String, dynamic> parsedJson) {

    var isLocatedAt = parsedJson['schema_address'] as List;
    List<SchemaAddress> schemaAddressList =
```

```
        isLocatedAt.map((i) => SchemaAddress.fromJson(i)).toList();

    var isLocatedAt2 = parsedJson['schema_geo'] as List;
    List<Schemageo> schemageoList =
        isLocatedAt2.map((i) => Schemageo.fromJson(i)).toList();

    return new IsLocatedAt(
        schemaAddress: schemaAddressList, schemageo: schemageoList);
  }
}

//Schema_address
class SchemaAddress {
  List<String> schemaAddressLocality;
  SchemaAddress({
    this.schemaAddressLocality,
  });

  factory SchemaAddress.fromJson(Map<String, dynamic> parsedJson) {
    var schemaAddressLocalityFromJson = parsedJson['schema_addressLocality'];
    List<String> schemaAddressLocalitylList =
        schemaAddressLocalityFromJson.cast<String>();

    return new SchemaAddress(
      schemaAddressLocality: schemaAddressLocalitylList,
    );
  }
}

class Schemageo {
  List<double> schemalatitude;
  List<double> schemalongitude;
  Schemageo({
    this.schemalatitude,
    this.schemalongitude,
  });

  factory Schemageo.fromJson(Map<String, dynamic> parsedJson) {
    var schemalatitudeFromJson = parsedJson['schema_latitude'];
    List<double> schemalatitudeList = schemalatitudeFromJson.cast<double>();

    var schemalongitudeFromJson = parsedJson['schema_longitude'];
    List<double> schemalongitudeList = schemalongitudeFromJson.cast<double>();

    return new Schemageo(
      schemalatitude: schemalatitudeList,
      schemalongitude: schemalongitudeList,
    );
```

```
  }
}

//HasReview
class HasReview {
  List<HasReviewValue> hasReviewValue;

  HasReview({
    this.hasReviewValue,
  });

  factory HasReview.fromJson(Map<String, dynamic> parsedJson) {
    var hasReviewValue = parsedJson['hasReviewValue'] as List;
    List<HasReviewValue> hasReviewValueList =
        hasReviewValue.map((i) => HasReviewValue.fromJson(i)).toList();

    return new HasReview(hasReviewValue: hasReviewValueList);
  }
}

//HasReviewValue
class HasReviewValue {
  List<String> rdfslabelreview;
  HasReviewValue({
    this.rdfslabelreview,
  });

  factory HasReviewValue.fromJson(Map<String, dynamic> parsedJson) {
    var rdfslabelreviewFromJson = parsedJson['rdfs_label'];
    List<String> rdfslabelreviewList = rdfslabelreviewFromJson.cast<String>();

    return new HasReviewValue(
      rdfslabelreview: rdfslabelreviewList,
    );
  }
}

//offers
class Offers {
  List<SchemaacceptedPaymentMethod> schemaacceptedPaymentMethod;

  Offers({
    this.schemaacceptedPaymentMethod,
  });

  factory Offers.fromJson(Map<String, dynamic> parsedJson) {
    var schemaacceptedPaymentMethod =
        parsedJson['schema_acceptedPaymentMethod'] as List;
```

```
    List<SchemaacceptedPaymentMethod> schemaacceptedPaymentMethodList =
        schemaacceptedPaymentMethod
            .map((i) => SchemaacceptedPaymentMethod.fromJson(i))
            .toList();

    return new Offers(
        schemaacceptedPaymentMethod: schemaacceptedPaymentMethodList);
  }
}

//SchemaacceptedPaymentMethod
class SchemaacceptedPaymentMethod {
  List<String> rdfslabelPayment;
  SchemaacceptedPaymentMethod({
    this.rdfslabelPayment,
  });

  factory SchemaacceptedPaymentMethod.fromJson(
      Map<String, dynamic> parsedJson) {
    var rdfslabelPaymentFromJson = parsedJson['rdfs_label'];
    List<String> rdfslabelPaymentList =
rdfslabelPaymentFromJson.cast<String>();

    return new SchemaacceptedPaymentMethod(
      rdfslabelPayment: rdfslabelPaymentList,
    );
  }
}

//is equipped
class IsEquippedWith {
  List<String> rdfslabelEquipped;
  IsEquippedWith({
    this.rdfslabelEquipped,
  });

  factory IsEquippedWith.fromJson(Map<String, dynamic> parsedJson) {
    var rdfslabelEquippedFromJson = parsedJson['rdfs_label'];
    List<String> rdfslabelEquippedList =
        rdfslabelEquippedFromJson.cast<String>();

    return new IsEquippedWith(
      rdfslabelEquipped: rdfslabelEquippedList,
    );
  }
}

class Mydata {
```

```
  String rdfslabel;
  String shortDescription;
  List<String> cuisineType;
  String archiStyle;
  List<String> rdfslabeltheme;
  String foafhomepage;
  String schematelephone;
  String schemaemail;
  String schemaAddressLocality;
  String rdfslabelreview;
  List<String> rdfslabelPayment;
  List<String> rdfslabelEquipped;
  double schemalatitude;
  double schemalongitude;
  int total;
  Mydata(
      this.total,
      this.rdfslabel,
      this.shortDescription,
      this.cuisineType,
      this.archiStyle,
      this.rdfslabeltheme,
      this.foafhomepage,
      this.schematelephone,
      this.schemaemail,
      this.schemaAddressLocality,
      this.rdfslabelreview,
      this.rdfslabelPayment,
      this.rdfslabelEquipped,
      this.schemalatitude,
      this.schemalongitude);
}
```

References

About Netflix. Netflix. Retrieved from https://media.netflix.com/fr/about-netflix

Adomavicius, G., & Tuzhilin, A. (2005a). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Trans. on Knowl. and Data Eng., 17(6), 734-749. doi:10.1109/tkde.2005.99

Aggarwal, C. C. (2016). Recommender Systems. New York: Springer.

Akerkar, P. S. S. a. R. (2010). Chapter 1 Knowledge-Based Systems for Development.

Bachimont, B. (2004). Arts et sciences du numérique: Ingénierie des connaissances et critique de la raison computationnelle. Mémoire de HDR.

Balraj Kumar, N. S. (2016). Approaches, Issues and Challenges in Recommender Systems: A Systematic Review. Indian Journal of Science and Technology, 2-12.

Beal, V. database. Retrieved from https://www.webopedia.com/TERM/D/database.html.

Beal, V. knowledge base. Retrieved from https://www.webopedia.com/TERM/K/knowledge_base.html

Bellinger, G., & Castro, D. (2004). Data, information, knowledge, and wisdom Online.

Berners-Lee, T. (1999). Lee. Retrieved from https://history-computer.com/Internet/Maturing/Lee.html

Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. Knowledge-Based Systems, 46, 109-132. doi:10.1016/j.knosys.2013.03.012

Bullinger-Hoffmann, A. (2008). Innovation and ontologies: Structuring the early stages of innovation management.

D. Goldberg, D. N., B. M. Oki, and D. Terry. (1992). Using collaborative filtering to weave an information tapestry. Commun. ACM, 35, 12. doi:10.1145/138859.138867

Ekstrand, M. D. (2011). Collaborative Filtering Recommender Systems. Foundations and Trends® in Human–Computer Interaction, 4(2), 81-173. doi:10.1561/1100000009

Felfernig, A., & Burke, R. (2008). Constraint-based recommender systems: Technologies and research issues. ACM International Conference Proceeding Series, 3. doi:10.1145/1409540.1409544

Felfernig, A., Friedrich, G., Jannach, D., & Zanker, M. (2015). In Francesco Ricci, Lior Rokach, & Bracha Shapira (Eds.), Recommender Systems Handbook (Second Edition ed.). New York Heidelberg Dordrecht London: Springer.

Flutter (software). (2019, June 19). wikipedia. Retrieved from https://en.wikipedia.org/wiki/Flutter_(software)

Guarino, N. (1995). Ontologies and knowledge bases: towards a terminological clarification. 25-32.

Guarino, N. (1997). Semantic matching: Formal ontological distinctions for information organization, extraction, and integration, Berlin, Heidelberg.

Hill, W., Stead, L., Rosenstein, M., & Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. Paper presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Denver, Colorado, USA.

Isinkaye, F., Folajimi, Y., & Ojokoh, B. (2015). Recommendation systems: Principles, methods and evaluation. Egyptian Informatics Journal, 16. doi:10.1016/j.eij.2015.06.005

J. Bobadilla, F. O., A. Hernando, A. Gutiérrez. (2013). Recommender systems survey. Knowledge-Based Systems, 109-132.

Jannach, D., Zanker, M., & Fuchs, M. (2009). Constraint-Based Recommendation in Tourism: A Multiperspective Case Study. Information Technology & Tourism, 11(2), 139-155. doi:10.3727/109830509789994784

John A. Bullinaria. (2005). IAI : Knowledge Representation.

Junker, U. (2004). QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. Aaai, 167-172.

Khusro, S., Ali, Z., & Ullah, I. (2016). Recommender Systems: Issues, Challenges, and Research Opportunities. 1179-1189. doi:10.1007/978-981-10-0557-2_112

Kumar, B., & Sharma, N. (2016). Approaches, Issues and Challenges in Recommender Systems: A Systematic Review. Indian Journal of Science and Technology, 9. doi:10.17485/ijst/2015/v8i1/94892

McGuinness, N. F. N. a. D. L. Ontology Development 101: A Guide to Creating Your First Ontology. Retrieved from https://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

Ontologie DATAtourisme v2.0_Documentation. (2019). Retrieved from https://framagit.org/datatourisme/ontology/blob/master/Documentation/Ontologie%20DATAtourisme%20v2.0_Documentation.pdf

P. Resnick, N. I., M. Suchak, P. Bergstrom, and J. Riedl. (1994). GroupLens: an open architecture for collaborative filtering of netnews Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (pp. 12). Chapel Hill, North Carolina, USA: ACM.

Patel-Schneider, P. F., & Fensel, D. (2002). Layering the Semantic Web: Problems and Directions, Berlin, Heidelberg.

Patel-Schneider, P. F., & Siméon, J. (2002). Building the Semantic Web on XML, Berlin, Heidelberg.

PROTALINSKI, E. (Producer). (2019, May 07). Google expands Flutter mobile app SDK to the web, desktop, and embedded devices. venturebeat. Retrieved from https://venturebeat.com/2019/05/07/google-expands-flutter-mobile-app-sdk-to-the-web-desktop-and-embedded-devices

R.Gruber, T. (1993). A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2), 199 - 220. doi:https://doi.org/10.1006/knac.1993.1008

Realtime Database vs. Cloud Firestore. (2018). Retrieved from https://medium.com/datadriveninvestor/realtime-database-vs-cloud-firestore-which-database-is-suitable-for-your-mobile-app-87e11b56f50f

Rich, E. (1979). User Modeling via Stereotypes. COGNITIVE SCIENCE, 3, 329-354.

Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In B. Peter, K. Alfred, & N. Wolfgang (Eds.), The adaptive web (pp. 291-324): Springer-Verlag.

Shardanand, U., & Maes, P. (1995). Social information filtering: algorithms for automating “word of mouth&rdquo. Paper presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Denver, Colorado, USA.

Sharma, A. (Producer). (2018, September 29). Realtime Database vs. Cloud Firestore. medium. Retrieved from https://medium.com/datadriveninvestor/realtime-database-vs-cloud-firestore-which-database-is-suitable-for-your-mobile-app-87e11b56f50f

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: principles and methods. Data Knowl Eng 25(1-2):161-197. Data & Knowledge Engineering, 25, 161-197. doi:10.1016/S0169-023X(97)00056-6

Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. Adv. in Artif. Intell., 2009, 2-2. doi:10.1155/2009/421425

Technical Overview. flutter.dev. Retrieved from https://flutter.dev/docs/resources/technical-overview

Wang, X. (2011). Recommendation in Social Media: Utilizing Relationships among Users to Enhance Personalized Recommendation. (Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of Doctor of Philosophy), Zhejiang University. Retrieved from http://summit.sfu.ca/system/files/iritems1/17566/etd10207_XWang.pdf

(n.d.). Retrieved June 2019, from code.visualstudio: https://code.visualstudio.com/

Amadeo, R. (Ed.). (2015, May 1). Google's Dart language on Android aims for Java-free, 120 FPS apps. Retrieved June 2019, from Ars Technica: https://arstechnica.com/gadgets/2015/05/googles-dart-language-on-android-aims-for-java-free-120-fps-apps

Cloud Firestore. (n.d.). Retrieved June 2019, from Firebase: https://firebase.google.com/docs/firestore/

Command-line interface / Commands - Composer. (n.d.). Retrieved June 2019, from Composer: https://getcomposer.org/doc/03-cli.md

Developer Survey Results 2019. (n.d.). Retrieved from StackOverflow Insights: https://insights.stackoverflow.com/survey/2019?utm_source=Iterable&utm_medium=email&utm_campaign=dev-survey-2019#technology-_-most-popular-development-environments

Developer Survey Results 2019. (2019, April 10). Retrieved from stackoverflow Insights.

Firebase Authentication. (n.d.). Retrieved June 2019, from Firebase: https://firebase.google.com/docs/auth/

Firebase Predictions. (n.d.). Retrieved June 2019, from Firebase: https://firebase.google.com/docs/predictions/

Firebase Realtime Database. (n.d.). Retrieved June 2019, from Firebase: https://firebase.google.com/docs/database/

Flutter (software). (2019, June 19). Retrieved June 2019, from wikipedia: https://en.wikipedia.org/wiki/Flutter_(software)

Flutter SDK releases. (n.d.). Retrieved June 2019, from flutter.dev: https://flutter.dev/docs/development/tools/sdk/releases

Google Analytics for Firebase. (n.d.). Retrieved June 2019, from Firebase: https://firebase.google.com/docs/analytics/

history of php. (n.d.). Retrieved june 2019, from php.net: https://php.net/manual/en/history.php.php

Ma, F. (2018, October 29). What's new at Firebase Summit 2018. Retrieved from The Firebase Blog: https://firebase.googleblog.com/2018/10/whats-new-at-firebase-summit-2018.html

Products. (n.d.). Retrieved from Firebase: https://firebase.google.com/products/

PROTALINSKI, E. (2019, May 07). Google expands Flutter mobile app SDK to the web, desktop, and embedded devices. Retrieved June 07, 2019, from venturebeat: https://venturebeat.com/2019/05/07/google-expands-flutter-mobile-app-sdk-to-the-web-desktop-and-embedded-devices

Tamplin, J. (2014, October 21). Firebase is Joining Google! Retrieved June 2019, from The Firebase Blog: https://firebase.googleblog.com/2014/10/firebase-is-joining-google.html

Technical Overview. (n.d.). Retrieved from flutter.dev: https://flutter.dev/docs/resources/technical-overview

Welcome! - The Apache HTTP Server Project. (n.d.). Retrieved June 2019, from httpd.apache: https://httpd.apache.org/