UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

MODELO GENÉRICO DE USUARIO PARA SISTEMAS DE RECOMENDACIÓN CONSCIENTES DEL CONTEXTO

TESIS QUE PARA OBTENER EL GRADO DE:

DOCTOR EN INGENIERÍA

PRESENTA:

SERGIO ALBERTO INZUNZA SOBERANES

Bajo la dirección de:
Dr. J. Reyes Juárez Ramírez

Co-dirigido por:
Dr. Guillermo Licea Sandoval

Tijuana, Baja California, México

Junio 2018

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

GENERAL USER MODEL FOR CONTEXT-AWARE RECOMMENDER SYSTEMS

A THESIS PRESENTED TO OBTAIN THE DEGREE OF:

PHD IN ENGINEERING

PRESENTED BY:

SERGIO ALBERTO INZUNZA SOBERANES

Under the direction of:
Dr. J. Reyes Juárez Ramírez

Co-directed by:
Dr. Guillermo Licea Sandoval

Tijuana, Baja California, México

Junio 2018

Universidad Autónoma de Baja California
FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

FOLIO No. 251

Tijuana, B. C., a 4 de junio de 2018

C. Sergio Alberto Inzunza Soberanes
Pasante de: Doctor en Ingeniería
Presente

El tema de trabajo y/o tesis para su examen profesional, en la Opción TESIS

Es propuesto, por los C. Dres. J. Reyes Juárez Ramírez y Guillermo Licea Sandoval

Quienes serán los responsables de la calidad de trabajo que usted presente, referido al tema Modelo Genérico de Usuario para Sistemas de Recomendación Conscientes del Contexto.

el cual deberá usted desarrollar, de acuerdo con el siguiente orden:

I.- INTRODUCCIÓN
II.- ANTECEDENTES
III.- MODELO GENÉRICO DE USUARIO PARA CARS
IV.- MARCO DE MODELADO PARA SISTEMAS DE RECOMENDACION
V.- CONCLUSIONES Y TRABAJO FUTURO
VI.- REFERENCIAS BIBLIOGRAFICAS

Dr. J. Reyes Juárez Ramírez
Director de Tesis

Dr. José Luis González Vázquez
Sub-Director Secretario

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA
FACULTAD DE CIENCIAS QUIMICAS E INGENIERIA

Dr. Guillermo Licea Sandoval
Co-Director de Tesis

Dr. Luis Enrique Palafox Maestre
Director

Dedicado a:

Mi esposa e hija

Por haberme apoyado en todo momento, por saber esperar y por haber soportado mi ausencia durante estos años, pero sobre todo por su amor.

Mis padres

Por brindarme su apoyo desde el inicio de este largo camino y por ser un ejemplo de perseverancia y persistencia.

AGRADECIMIENTOS

Agradezco al Dr. Reyes Juárez Ramírez por haberme aceptado bajo su tutela y luchar para que esto fuese posible, por ser un ejemplo de superación personal, pero sobre todo por su amistad.

Agradezco a mis compañeros del grupo de investigación. A Samantha por decir tantas tonterias y alegrarnos el día a día con sus ocurrencias, y por servir de ejemplo como persona, amiga e investigador. A Angeles por su amistad, su apoyo incondicional y por los waffles. A Carlos por su amistad, y por incitar esas discusiones tan entretenidas. A Raúl por recordame firmar, por esas platicas tan técnicas, y por los cafés con chocolate. A Alan por ser un ejemplo de dedicación y de actuar conforme a una meta de vida.

De igual forma agradezco a los demás compañeros con lo que tuve el placer de convivir, compartir ideas y aprender de ellos. De cada uno me llevo una parte y espero tener la oportunidad de seguir conviviendo con ellos tanto en lo persona como en lo profesional.

Agradezco infínitamente a los miembros de mi comité. A Dr Licea, Dr Tapia, Dr Mario, Dr Juan Ramón por tomarse el timpo de escuchar mis presentaciones, y por sus sugerencias constructivas sobre como mejorar en el dominio del tema. De igual forma agradezco a Dr. Victor, que aun con la poca convivencia me demostró que el éxito debe medirse a nivel personal y que el ser éxitoso no debe interferir con ser humilde y cordial.

Agradezo a UABC por haberme acogido durante todos estos años, por darme la oportunidad de crecer, aprender y tener una profesión. Así mismo agradezco a CONACYT y por lo tanto a México por su apoyo econónico, ya que sin el esto no habría sido posible.

Quiero agradecer también a mis familiares y amigos por todo el apoyo brindado durante estos cuatro años.

A todos los anteriormente mencionados, de corazón un profundo agradicimiento!!

RESUMEN

Los modelos de informacin son indispensables para el correcto funcionamiento de los sistemas de recomendacin conscientes del contexto (CARS), a pesar de ello, hoy en dia existe una necesidad de herramientas y estructuras de información de usuario y de contexto, que faciliten el desarrollo de este tipo de sistemas.

En este trabajo se presenta un modelo genérico de usuario para sistemas de recomendación conscientes de contexto (CARS por sus siglas en inglés), que proporciona una estructura estática para una amplia variedad de información de usuario, contexto, y de los elementos que los CARS recomiendan. La información de dichos elementos fue obtenida mediante una revición sistemática de literatura siguiendo la metodología de Kitchenham.

La calidad estructural del modelo fue validada utilizando métricas de calidad de software, los resultados sugieren que los elementos contenidos en el modelo están correctamente organizados. En una segunda validació cuantitativa se demostró que el modelo es capaz de almacenar casi toda la información contenida en los conjuntos de datos de CARS, así como de soportar información de diferentes dominios de recomendación.

Así mismo, como extensión del modelo de usuario se presenta un marco de modelado de usuario para sistemas de recomendación como una herramienta capaz de administrar la información requerida por los algoritmos de recomendación, y cuya finalidad es incrementar la productividad de los desarrolladores durante la fase de implemenaciíon de este tipo de sistemas.

La calidad estructural del marco de trabajo fue validada utilizando métricas de calidad de software, los resultados sugieren que los elementos que componen el marco de trabajo están correctamente organizados. Finalmente, mediante una validación cualitativa del marco de trabajo en la cual desarrolladores utilizaron el marco propuesto para la implementación de tareas de administración de datos, se demostró que el uso del marco de trabajo disminuye el esfuerzo y las líneas de código requeridas para implementar las tareas, lo que se traduce en un aumento de la productividad de los desarolladores.

ABSTRACT

Context-Aware Recommender Systems (CARS) are extensions of traditional recommender systems that use information about the context of the user to improve recommendation accuracy and user experience. Whatever the specific algorithm exploited by a CARS, it can provide high-quality recommendations only after having modeled the user and context aspects.

Despite the importance of data models in CARS, nowadays there is a lack of tools that support and facilitate the development of CARS, especially structures for user and context information management, which leaves designer, developers, and researchers the work of designing and implementing their own context-aware models to manage the information needed by recommendation algorithms, based on their knowledge and with no model to use as a reference, often resulting in overspecialized, inefficient, or incomplete models.

In this work, a general user model for context-aware recommender systems is presented, that provides a static structure for an extensive set of user, context and items information. As an extension of the user model, a User Modeling framework for Recommender Systems (UM4RS) is presented to increase developers productivity when implementing data management functionality in CARS.

The structural correctness of the user model was validated using software quality metrics, the results show that the elements contained in the user model are correctly organized. Then, in another quantitative validation, the model was proved to be capable of storing almost all the information contained in CARS datasets and to support information from different recommendation domains.

In a third quantitative validation, the structural correctness of the framework was validated showing that the structural correctness of the model was kept when implementing the model in the modeling framework. Finally, in a qualitative validation of UM4RS where developers used the framework during the implementation of some data management task showed that UM4RS decrease the effort and lines of code required to implement the task, which translates in an increase in developers productivity.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The rapid growth of potentially interesting products available through online services, frequently overwhelm users surpassing their ability, knowledge or time to make good decisions [1]. The large availability of choices, instead of producing a benefit, started to decrease users' well being. This information overload problem served as driving force for the development of intelligent software systems to help the users in making a decision over what products will better fit their needs and interests [2]. Such software systems received the name of Recommender Systems.

Recommender Systems (RS) are defined as software systems that attempts to identify a subset of items from a –typically large– information space that meet a user's interest and preferences best among all alternatives, and which subsequently presents those items to the user in a suitable manner [3]. In their simplest form, a personalized recommendation if presented to user as a ranked list of items. This recommendation task is achieved by exploiting various types of knowledge, like data about users, the available items, and information about previous transactions stored in customized databases [4]. Extensive research of recommender systems started over two decades ago and yielded a wide variety of recommendation techniques, such as content-based filtering [5], collaborative filtering, [6] and multiple hybridizations [7]. A recent research trend in recommender systems is the inclusion of context information in the recommendation algorithms, as contextual information has been proved to help increasing the prediction accuracy of recommender systems [8, 9]. This type of recommender systems are known as context-aware recommender system.

Context-Aware Recommender Systems (CARS) are an evolution of traditional recommender systems that in addition to considering information about the items and the history of interactions between the user and the system, includes extra information that describes users characteristics and their surround environment to generate better predictions [10].

In the beginning, the research in CARS was leaned toward the development of new, and optimizing the existing algorithms to support contextual information and to generate better recommendation by improving the accuracy [11]. In recent years, researchers have become more aware of the fact that effectiveness of recommender systems goes beyond recommendation accuracy. Thus, research in the context and human factors has gained increased interest [12] as a potential opportunity to increase the user satisfaction with the recommendation results. Based on that premise, studies like [1, 3, 13–15] has focused on demonstrating the benefits of including user and context information, on which they all conclude that including contextual information inside the recommendation generation function brings a positive impact on both accuracy and user satisfaction.

Contextual information plays an important role in CARS, as user behavior is affected by the user current context [16], e.g., time, location, mood, and weather. CARS are based on the idea that similar users, in similar contexts, like similar items, and that user's preferences change according to his contextual situation [14]. From the beginning of CARS research, Adomavicius [1] laid the groundwork for techniques on how traditional recommender system can include and process all this new information. According to Adomavicius [1], CARS recommendation techniques can be classified into *pre-filtering*, *post-filtering* and *contextual modeling*. Such techniques were adopted as a de facto standard by CARS researchers, and several recommendation algorithms were created around such techniques.

Whatever the specific technique or recommendation algorithm exploited by a recommender system, it can provide high quality recommendations to users only after having modeled their preferences [17, 18]. Therefore, in CARS as in most personalization systems, a user model is an essential component used to store the information about the user and his interactions with the systems, which can later be used to adapt and personalize the system in order to improve the experience of the user in his future interactions [19, 20].

User models allow CARS to store the collected information about the user and their preferences, the contextual information that described the user environment, the items' characteristics, and the transactional records resulting from previous interactions with the system. Later this information is used by the recommendation algorithm to generate better predictions [21], and can be used to enhance the user experience when interacting with the user interface [14].

1.1 Problem Description

Despite the advances in user modeling and context information management [22], the design and development remain significantly more challenging for context-aware recommender systems that traditional systems, especially without supporting tools that facilitate those process [23], mainly because adding context-aware capabilities to a software system brings design and development overhead inherited from the complexity of managing (acquiring, organizing and storing) the user and context information [24, 25].

Nowadays there is a lack of tools that support and facilitate the development of CARS [11], especially structures for user and context information management [19, 26], which leaves developers and researchers the work of designing and implementing their own context-aware models to manage the information needed by recommendation algorithms, based on their knowledge and with no model to use as a reference, often resulting in overspecialized inefficient or incomplete models [25, 27].

In addition, considering that the creation of models for user and context information is a complex [25] and effort consuming [23] task, it becomes specially important to provide developers and designers with data models that facilitate the creation of CARS, especially for non-experienced designers and developers.

Even when some proposals of user model (like [28]) and context model (like [29]) exist in the literature, they were designed not designed specifically for context-aware recommender systems, which means that such models do not consider key information for CARS, like items data or ratings history. Also, some of the proposals are too abstract, designed at ontological level (e.g. [30, 31]); others present the information categories, but not the specific attributes of user or context (e.g. [29]), and getting them to implementation implies a lot of work.

In this work, we propose a Generic User Model for Context-Aware Recommender Systems (GUMCARS) that address the problem of lack of structures for recommender system data, by providing a generic structure that models the information of user, context and items required by CARS algorithms to perform their predictions.

In order to address the lack of tools that support and facilitate the development of CARS, this work also propose a user modeling framework (UM4RS) that increase developers productivity when implementing data management functionality in CARS, by providing a working ready modeling tool capable of structure, store, and retrieve GUMCARS information to and from a database solution.

1.2 Research Objectives

The development of the proposal is guided by the following objective:

Develop a generic user model that structure the user, context, and item information of CARS, to increase developers productivity in the implementation phase of these type of systems.

To achieve such objective, the development of this thesis is guided by the following list of specific objectives:

Obj1 Identify the user, context, and items information aspect that has been used in context-aware recommender systems literature.

Obj2 Design a user model for context aware recommender systems that structure the user, context, and items information.

Obj3 Design and implement a modeling framework around the proposed user model, to increase CARS developers productivity.

Goals

The goals for the **Obj1** are:

Goal 1: Design a taxonomy of the information used by context-aware recommender systems algorithms.

Goal 2: Perform a systematic literature review based on Kitchenham methodology to identify the information aspects about the user, context, and items used by CARS literature as input information for recommendation algorithms.

The goals for the **Obj2** are:

Goal 3: Create a data structure that organizes the information of user, context, and items following object-oriented design paradigm.

Goal 4: Design the data structure to support information from different recommendation domains.

Goal 5: Design the data structure to comply with object-oriented quality rules.

The goals for the **Obj3** are:

Goal 6: Create a user modeling framework that automatically store and retrieve information from the user model classes into a database solution.

Goal 7: Design the modeling framework to comply with object-oriented quality rules.

Goal 8: Design framework API to be easy to use by software developers.

Goal 9: Create a web-based documentation to allow developers explore and learn the functionality provided by the modeling framework.

1.3 Hypothesis

H_1: A user modeling framework that structure user, context, and items information, increase developers productivity in terms of effort and lines of code required to implement a context-aware recommender systems.

1.4 Methodology

To archive the proposed objectives, this thesis work followed a five steps methodology.

Literature Review

The beginning of the research consisted of reviewing and analyzing user modeling, context modeling, and context-aware recommender systems to identify a possible categorization of user, context, and items information respectively. Then, a systematic literature review was performed to identify the specific information aspects used in CARS.

Design of a user model for CARS

A generic user model for context-aware recommender systems was designed, based on the results of the literature review. For the model design, the object-oriented design paradigm was followed, and the Unified Modeling Language (UML) was used to graphically present the model elements.

Validation of the proposed user model

During the validation stage of the model, first, a list of CARS datasets was used to assess the level of completeness and generality of the proposed model by measuring the model capability to structure the information contained in the datasets. Then, the second validation performed consisted of measuring the model adherence to object-oriented design rules by applying a set of software quality metrics.

Design and development of a modeling framework for CARS

Based on an analysis of open challenges in CARS implementations, and following the presented objectives, a user modeling framework was designed and implemented using the proposed user model as the core component of the framework. The framework was implemented following the object-oriented paradigm, using C# language and Microsoft .Net Framework platform. To persist the data contained in the model, Microsoft SQL Server was used, along ADO.Net Entity Framework to perform an automatic mapping of model classes and attributes into database tables and columns.

Validation of the modeling framework

First, a validation of the structural correctness was performed using software quality metrics to ensure the adherence of the framework implementation to object-oriented rules. This validation is similar to the one performed over the model design, but the first was a validation of the conceptual design, and this is a validation of implementation, for this validation more metrics were applied compared to previous validation of the model.

Then, an experiment was performed to assess the impact of the framework on developers productivity. The participants of the experiment were developers currently working in the software industry, as well as college students. During the experiment, the effort and lines of code required to implement a set of tasks with and without the framework were measured. Then the difference between both scenarios was analyzed.

1.5 Thesis Structure

This document is organized based on the following structure.

Chapter 2. Background and Related Work

This following chapter introduces the main topics embraced by this thesis, it defines and present an overview of Recommender Systems, it defines and briefly give an overview of the inner workings of both main types of recommender systems: Traditional and Context-Aware Recommender Systems. This section also introduces the topic of Context, presenting some definitions for the term and describes context modeling. The last topic presented in User Model and the user modeling process. Lastly, this section discuss the related work to this proposal.

Chapter 3. General User Model for Context-Aware Recommender Systems

This chapter presents GUMCARS a General User Model for Context-Aware Recommender Systems. It describes the proposal of a taxonomy for CARS information, and then describes a systematic literature review performed to gather the information aspects that literature uses as input for CARS algorithms. Then, the proposed user model is created and its architecture is described. Last two section of this chapter present two validations performed to GUMCARS proposal, a completeness and generality validation, and a structural correctness validations of the model.

Chapter 4. User Modeling framework for Recommender Systems

This chapter present the second contributions of this thesis works, a user modeling framework called UM4RS intended to increase developers productivity by saving developers and designers the process of structuring, persisting and formatting the user and context information. In this chapter, the structural correctness of the framework is validated. Then, the effect of UM4RS on developers productivity is evaluated in a qualitative experiment, that also allowed us to assess the usability of the framework.

Chapter 5. Conclusions and Future Work

This final chapter discusses the conclusions of this work and describes the impact of the proposals to academy and industry. Then, this chapter presents an analysis of contributions, and finally, it described possible paths for future research in user and context modeling derived of this thesis.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Recommender Systems

Recommender System is a software systems that attempts to identify a subset of items from a information space that better meet the user interest and preference, and present them to the user as a ranked list of options to choose from. The main goal of recommender systems is to help users in the decision making process, by filtering the information space, and present them a handful of options.

Ricci et al. [3] establishes through the following main facts, the growing interest that recommendation systems have acquired:

- Recommender systems play an important role in important Internet sites like Amazon.com, YouTube, Netflix, Yahoo, Tripadvisor, Last.fm, and IMDb. Moreover, many media companies are now developing and deploying RSs as part of the services they provide to their subscribers.
- There are dedicated conferences and workshops related to the field. We refer specifically to ACM Recommender Systems (RecSys), established in 2007 and now the premier annual event in recommender technology research and applications. In addition, sessions dedicated to RSs are frequently included in the more traditional conferences in the area of databases, information systems, and adaptive systems.
- At institutions of higher education around the world, undergraduate and graduate courses are now dedicated entirely to RSs; tutorials on RSs are very popular at computer science

conferences, and some books introducing recommender systems techniques were published.

2.1.1 Traditional Recommender System

Typically, the traditional or *two-dimensional* (2D) recommendation starts with the specification of the initial set of ratings that is either explicitly provided by the users or implicitly inferred by the system. Once these ratings are gathered, a recommender system try to estimate the rating function R for the (user, item) pairs that have not been rated yet by the users. Here $Rating$ is a totally ordered set of non-negative numbers, and $User$ and $Item$ are the domains of users and items of the systems respectively. Once the function R is estimated for the whole $User \times Item$ space, a recommendation algorithm can suggest the highest-rated item for each user. This systems are called 2D since the recommendation process is based only in the $User$ and $Item$ dimensions. Next, the Equation 2.1 formally describes the rating functions.

$$R : User \times Item \rightarrow Rating \tag{2.1}$$

From its beginning, research in recommender systems area, has so far mostly concentrated on developing and optimizing recommender algorithms and on evaluating which methods work best under different conditions [32].

As a result, nowadays recommendations can be generated in various ways, using different algorithms and data inputs. Most techniques can roughly be divided into content-based, collaboration filtering, and hybrid approaches. Next, each technique is briefly described:

Content-Based Filtering

CB is one of the most widely used and researched recommendation technique [3]. One central component of CB is the user model, which contains information about the user, and the interest that the user has for the items in the catalog, such information was collected during the data collection process [33, 34]. Another important component for CB recommendation is the information about the items. Items are represented by a content model containing the items features (tags) and stored as a vector that contains the features and their weights for that item. To predict the utility of the items (generate recommendation), the engine compares the items that were already positively rated by the user with the items he did not rate and look for similarity in the items features. The items that have more similar features are selected and recommended to the users [8, 35].

Collaborative Filtering

This recommendation approach relays on the idea that if users shared the same interest in the past if the viewed or brought the same books, for instance they will also have similar taste in the future [36]. The main idea in CF is to exploit information about the past behavior or the opinions (ratings) of an existing user community for predicting which items the current user will most probably like [37]. After collecting the information of the system users, one of the most important step of this approach is performed; which is the construction of a neighborhood of like-minded users, for which the users in the system must be compared to one another, and create clusters of users based on their user profile [30, 35, 36]. Then the prediction of items for a specific user is based on items that he never rated, but that other users in the same neighborhood rated positively [34, 38].

Hybrid Recommendation

All approaches have their limitations. CB filtering is no applicable when the item have no proper descriptions. CF strongly depends on the availability of meaningful user rating on a large scale to generate recommendations [11]. Therefore hybrid recommenders systems emerge, which aims to eliminate specific drawback of both methods by combining them. Hybrid methods use CF and CB together with different strategies, some hybrid systems switch between CF and CB under different circumstances [30]. The combination of approaches can proceed in different ways [35, 37]: separate implementation of algorithms and joining the results, utilize some rules of CB in CF approach, utilize some rules of CF in CB approach, and create a unified recommender system, that brings together both approaches [39].

2.1.2 Context-Aware Recommender Systems

Traditional recommender systems base their results only on the user and his or her interest inferred from the log of previous transaction of the system, which in many cases is sufficient. There are, however situations in which including additional (contextual) information can be beneficial for producing more accurate and meaningful recommendations [11]. This contextual information may refer to detailed information about the user like his current mood, or physiological information [40]. Also this extra information may come from the environment where the user or the system are in a given moment, for example, the weather condition [41], or the users companion [16]. These type of recommendation systems that uses user and contextual information are called context-aware recommender systems.

Context-Aware Recommender Systems (CARS) aims to further improve recommendation accuracy and user satisfaction by taking into account contextual information in their recommendation algorithm [42]. CARS are based on the idea that similar user in similar context will like similar items, and that the user preferences for certain item change according to the contextual situation [14]. In the beginning, the research in CARS was leaned toward the development of new, and optimizing the existing algorithms to support contextual information and to generate better recommendation by improving the accuracy [11]. In recent years, researchers have become more aware of the fact that effectiveness of recommender systems goes beyond recommendation accuracy. Thus, research in the context and human factors has gained increased interest [12] as a potential opportunity to increase the user satisfaction with the recommendation results.

In traditional recommender systems with user set $User$ and item set $Item$, the set of possibilities in $U \times I$ is mapped to a rating value given by the user. In CARS an additional set of possibilities ($Context$) is needed to represent the contextual information. For example, the set $Context$ might be $Sunny, Cloudy, Rainy$, when presenting the *weather* context. In such a case, there is no longer possible to map the ratings in $User \times Item$, because the same user might have different preferences for an item depending to the contextual condition, which means that for the same item, the user may gave an rating of 4 when in sunny, but a rating of 2 when the weather is rainy. Therefore, the context must be included in the mapping of the rating in order to provide a more refined and accurate recommendation. Formally, the recommendation function for CARS or nD recommendation is:

$$R : User \times Item \times Context \rightarrow Rating \tag{2.2}$$

Even when the area of recommendation systems, the existed a lot of algorithms based to process the information and produce a suggestions list, this algorithms were designed to work with two-dimensional information, and are not capable of supporting this new dimension ($Context$), at least not without proper modifications. CARS research produced a series of paradigms to incorporate context dimension into the recommendation functions, next we describe the most popular.

2.1.3 Paradigms for Incorporating Context

As traditional recommendation algorithms were not able to work with multi-dimensional data, some algorithms were modified and other new algorithms were created to be able to generate contextual recommendations. These algorithms can be classified into [43] *contextual filtering* algorithms (pre and post-filtering) which intention is filter the information to reduce dimensionality and feed the filtered data through traditional recommendation algorithms, and *contextual modeling* algorithms

which model the contextual information directly into the recommendation algorithm.

Contextual Pre-filtering Recommendations

Also known as contextualization of recommendation input. In this recommendation paradigm (presented in Fig. 2.1) the context information drives the selection of relevant data (i.e. Ratings) from the background data.

Then the selected dataset (contextualized data) is used to generate prediction using a traditional (2D) recommendation system. Basically first the contextual data is preprocessed and the result serves as input for the recommendation engine, a hypothetical example could be a song recommender system that takes into account the weather, if the current weather is rainy, then from all the dataset, select only the transactions that were performed in a rainy day, then use the resultant data with a traditional recommendation algorithm to generate song predictions [43].

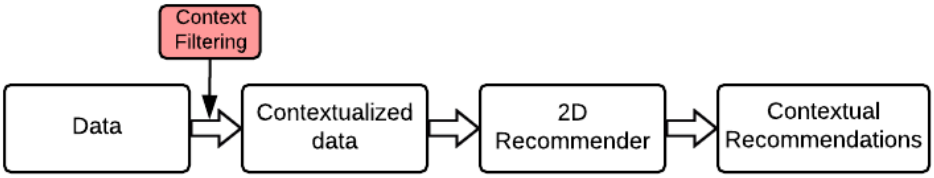

Figure 2.1: Pre-filtering context processing paradigm

Contextual Post-filtering Recommendations

Also known as contextualization of the recommendation output. In this approach, the contextual information is first ignore, the 2D matrix data ($User \times Item$) data is feed into traditional recommendation algorithms to generate a list of predictions like any other traditional recommender systems. Then the contextual information is used to filter the ranked list obtained from the recommendation algorithm to select the predicted items that better fits the current context of the user, based on the context stored for the item. This process is graphically depicted in Figure 2.2.

In general terms, Adomavicious et al. [43], describe post-filtering like an approach that analyzes the contextual preference data for a given user in a given context to find specific item usage patterns and then user these patters to adjust the item list resulting from a 2D recommender. For example, in a movie recommendation system, if a person is looking for a movie recommendation of a weekend, and on weekends, she only watches horror movies, the system can filter all the non-horror movies from the recommendation list.

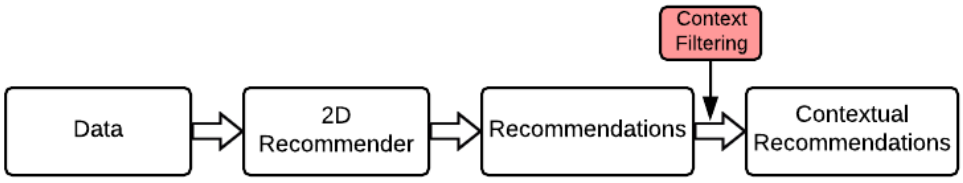

Figure 2.2: Post-filtering context processing paradigm

Contextual Modeling Recommendation

Also known as contextualization of the recommendation function. In this recommendation approach the context processing and recommendation generation is performed in a single step, the contextual information is used directly in the modeling technique as part of the rating estimation.

While pre-filtering and post-filtering approaches benefited from the existing 2D recommendation algorithms, the contextual modeling approached gives rise to truly multidimensional recommendation functions that are able to incorporate the contextual information in addition to user and item information [29].

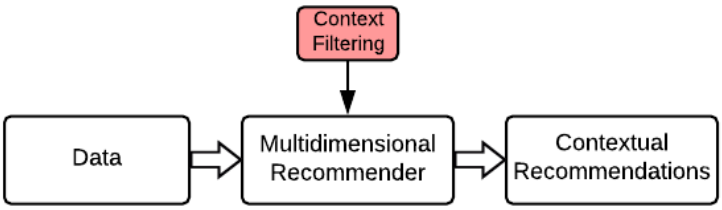

Figure 2.3: Contextual modeling paradigm

2.2 Context and Context-Awareness

Humans are quite successful at interacting with each other and reacting appropriately by considering many factors: the richness of the language, the common understanding of how the world works, and by using implicit situational information, or *context*. Unfortunately, this level of interaction does not transfer well to humans interacting with computers. In traditional interactive computing, software systems only have the information that the user has provided; therefore, such systems are not enabled to take advantage of the human situational information. By providing software system access to context, the richness of human-computer interaction is certainly increased [44].

In order to use context effectively, we must understand what context is and how to use it. This section presents some definitions of context to lay down the notion of what context is. Then we

briefly describe how context has been used in computing system to improve user interaction with especial interest on how context has been modeled in such software systems.

2.2.1 Context Definition

Context is a multifaceted concept that has been studied across different research disciplines, including computer science (primarily in artificial intelligence and ubiquitous computing), cognitive science, linguistics, philosophy, psychology, and organizational sciences [43].

Since context has been studied in multiple disciplines, each discipline tends to take its own idiosyncratic view that is, and how to define it, somewhat different from other disciplines. In computer science, Schilit et al. [45] described context as a union of three aspects: the location of user, the collection of nearby people and objects, as well as the change to those objects over time, but such definition is rather broad.

A more specific definition of context, and more commonly accepted in computer sciences comes from Dey and Abowd [44], they define context as follows:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

In literature related to context-aware and context-aware recommender systems, context as initially defined as the location of the user, the identity of people near the user, the objects around, and the changes in these elements [46]. Other factors have been added to this definition subsequently. For instance, some authors include the date, the season, and the temperature. Others added the physical and conceptual statuses of interest fir a user. Dey et al. [47] include the user's emotional status and broaden the definition to any information which can characterize and is relevant to the interaction between the user and the system [43].

In this work, we follow Dey and Bowd [44] definition of context, buy we do not consider the user information to be part of it, as we structure such information at the same level of the contextual information. Therefore, we consider context to be *any information that can be used to characterize the situation of a person or object, that is considered relevant to recommendation applications.*

2.2.2 Context-Aware Systems

The term of context-aware computing was first used by Schilit and Theimer [45] as a way to refer to "software that adapts according to is location of use, the collection of nearby people and objects, as

well as changes to those objects over time". Context-aware has become synonymous with other terms like: adaptive, reactive, responsive, situated, context-sensitive and environment-directed [44].

A system is considered to be context-aware or to have context-awareness if it can express aspect of the environment and subsequently uses such information to adapt its functionality to the context where the system is being used [48]. Typically, the specification of the context of interest can be given by the five "W" [49]: What?, Who?, Where?, Why? and How?.

Although the first appearing context-aware systems can be seen as relatively easy to develop (e.g. display the weather according to its current location), the fully exploitation of contextual information gives application much more potential, but involves a deeper understanding of context. Current literature of context-aware systems depicts that nowadays research is aimed towards creating computational systems that are able to understand not only simple contexts, but also other such as the environment, information about the user, and the social information of the user. The deeper context consciousness desired for the application, the greater the understanding it required on complex relation between context, human activity and human behavior [50, 51].

There exist two main research areas in context-aware computing, how the context is acquired, and how the context is used. The acquisition of contextual information it"s outside the scope of this thesis work, we part from the premise that the data is already gathered.

With respect to context usage, Perera et al. [22] describe the following three steps in the context usage life cycle, and Alegre et al. [26] identified a series of open challenges to overcome in each of the steps.

Modeling

After having the information gathered, it need to be translated into values that can be represented, used by a computer system and understand by humans. For example the user location (expressed in latitude and longitude values "32.85, -116.32"), must be translated into a street name, city, etc. This is archived by translating the real world concepts into modeling constructs. These models require: (1) To present any kind of context information, reflecting the entities of real world and the relations between such entities; (2) Uniquely identify the contextual information, context and entities; (3) to be simple, reusable, expandable and able to use the information at runtime; and (4) Validate pieces of data and encode its uncertainty.

Reasoning

Different kinds of conclusions can be inferred from the data stored on the model. Such that, new knowledge and understanding is obtained, based on the modeled context, which is typically done

in a 3 steps: (1) Context pre-processing, where data is cleaned to get rid of invalid, inaccurate and non desirable values; (2) Sensor data fusion, which consist of merging data from different sensors to produce more accurate information; and (3) Context inference, from low level information to high-level one.

Dissemination

Finally, both low-level and high-level context needs to be distributed to the consumer. The context information must have high availability, ideally to be provided in real-time.

From the three challenges of context usage, this thesis focus only in the modeling of the gathered data, with the intention of organizing, storing and making it available for recommender systems to take advantage of it in the recommendation generation process. In the next-subsection the area of context modeling is discussed, and some of the most relevant concepts and related works are presented.

2.2.3 Context Modeling

Context modeling technique, allows to describe and structure the contextual information [52]. In this work we focus on modeling the contextual information such a way that computer can use the information to enhance the interaction with users. We define a context model as a structure that can be used to represent the information of the context inside a computational system.

From an *infrastructural* perspective, context provides computing devices with information about its environment as provided by other system components. In order to provide such information, context needs to be classified in different 'types' or 'dimensions' of context, e.g. physical, computational, etc.

Dourish [53] introduces a taxonomy of context, according to which context can be classified into *representational* or *interactional* view. In representational approach, context is defined with a predefined set of observable attributes, a structure (or schema, using database terminology) which does not change significantly over time. In other words, the representational view assumes that the contextual attributes are identifiable and known a priori and, hence, can be captured and used within the context-aware applications. In contrast, the interaction view assumes that the user behavior is induced by an underlying context, but that context itself is not necessarily observable. As the interactional view of context is an approach borrowed from psychology [43] that considers context non-observable, it cannot be used to explicitly store the context information into a computer system. Therefore, in this work, as most of the works on context-aware recommender systems are based on

the representational view of context, as this approach allows to identify context attributes and to model it.

2.3 User Model

As user interact more and more with digital systems, they produce an increasing amount of information. This information flow represents a valuable source of information for personalization systems [54]. But to be able to take advantage of such information, the information must be stored first, when the information about the user is explicitly stored instead of being hard-coded in the system login, is known as a user model [55].

Most authors refers to user models as structures that organize and store the information about the user as a persona. For example, a common definition used in the area of personalization system is the one given by Kobsa [56]:

> *A user model is a collection of information and assumptions about an individual user (as well as a user group), which is needed in adaptation processes.*

In this work with stick with a more general definition of context, like the one proposed by Schreck [57]:

> *A user model is defined as the knowledge about the user, explicitly or implicitly encoded, which is used by any system to improve user interaction.*

A user model combines user preferences with the stated goals or interests and the behaviors performed by that user, and uses this information to deduce the perceived current goals and interests of the user [19]. When a instance of the model is created to store the information, is known as *User Profile* [58].

All this user information structured in the user model, and stored in user profiles is an important component of CARS, as in order for CARS to generate relevant recommendations, it needs to model the information of the user, the history of interactions between the user and the system, and the context where such interaction take place [18].

2.3.1 User Modeling

User modeling is the whole process of constructing the user model, and creating, updating or deleting user profiles. It contains the functions which are to incrementally build up a user model, to

store, update and delete entries in instantiated user profiles, and to maintain the consistency of the model [59].

According to [58], the user modeling process can be divided into two main steps:

1. *Collecting the data about users*. User data may be gathered by the system either in a client side, or in the server side, using various ways that goes from directly asking the user for it, or inferring it from user actions. Certainly is easier to show filling form to user and ask him to provide the required information, however, the user may withhold information. Another option is to infer the information from the usage of the system, however usage information is not fully reliable. A third and option is to use sensors to measure and log different aspects of user, and even when this option is less intrusive that filling forms, and more reliable that usage information, there are some information about the user that will be difficult to sens, like user's mood.

2. *Build or update existing models*. To build the model various different techniques are used. Manual building of the model may ensure a good outcome, but require a lot of human labor, and the participant will require high understanding of the modeled domain. Machine learning techniques like linear models, Markov models, and neural networks are also used to build user model. Also, data mining techniques such as association rule mining and maximal frequent mining are also used to build user models. In practice, systems us many different approaches and techniques to build up user model. Techniques and approaches should be chosen according to specific cases and needs.

Context-Aware User Modeling

The recent advance in computing technology that offers "anytime, anywhere, anyone" computing, has enabled software system to acquire more information about the user and his surroundings, which introduced the challenge of context-aware user modeling. A user model can be considered context-aware if it can express aspects of the user's contextual situation and helps software systems to adapt their functionality to the context of use [48].

In [17] context-aware user models are called *three-dimensional user model* being $User$, $Context$ and $Item$ each of the three dimensional space. This type of user model are specific for context-aware recommender systems. In addition, [17] discussed the notion that user preferences in any type of user model are generally valid only within specific contextual conditions. That is, a user's preferences store in the user model may change as function of various contextual conditions. Hence, to facilitate the provision of context-aware recommendation, traditional (2D) user models needed to

evolve in context-aware user models in order to support all the information needed by contextual recommendation algorithms.

User Model Requirements

In the early stages of user modeling technique, Kobsa [56] detailed the following list of requirements that any user model and user modeling server should consider.

- **Generality** User models must be usable in many applications and content domains as possible, and within these domains for as many user modeling tasks as possible. That means that user models should be independent of the application, and if are dependent of a specific domain, the user model should support different task within that domain, which allow the user model to be used and shared across different application that benefit from the same information domain.
- **Expressiveness** User models are expected to be able to express as many types of assumptions about the user as possible at the same time. This will allow model to express a broad range of information of the user.
- **Inferential Capabilities** User models mus support all sorts if reasoning that are traditionally distinguished in artificial intelligence and formal logic. This way, applications that uses the model should be able to perform reasoning process over the information stored in the model.

Some years later, Flink [60] increased the number of requirements for user model adding the following 3:

- **Import for external user information** The models should allow developers and if possible, the own users no only to add new information, but to load bulks of information from external sources.
- **Privacy Support** The models should be able to support company privacy policies, industry privacy norms and conventions, by using an access control to the contained information. Event when this rule is more applicable to the modeling server (the entity that houses and dispose the model information), it should be considered since the model design.
- **Extensibility** The model should support data acquisition and personalization methods. Commonly, this is comply by providing and Application Programming Interface (API) that allow for bi-directional exchange of user information between the model and the systems that benefits from such model.

2.4 User and Context Modeling Representation Techniques

Previous two sections describe the important for software systems, especially for context-aware recommender systems to user a model of the user and context information. In this section, the most common techniques used in literature to support such model insider computer systems are described and analyzed.

There exist several techniques that a model designer can use to represent the user and context information inside a computational system. Works like [22, 61, 62] present extensive surveys on the difference of each technique. Next, we describe and discuss the most commonly used representation techniques at high-level, and present the main advantages and disadvantages of each one.

Key-Value models

These models use pairs of a key and value to enumerate attributes (key) and their values to describe the contextual and user information. These models are the simplest data structure and are easy to manage, especially when they have a small amount of data. However, key-value modeling is not scalable and not suitable to represent complex data models [62]. This technique is best suited to represent and store temporary information, therefore is increasingly less used in recent contextual and user models.

Markup scheme models

These models use a hierarchical data structure formed by markup tags with at-tributes and content. To represent the user and context aspect, markup models use a set of symbols and annotations inserted in a text document that controls the structure, formatting and relations among annotations [63]. As markup languages do not provide advanced expressive capabilities, reasoning over the data they represent is hard. Further, retrieval, interoperability, and re-usability of the data over different models can be difficult, specifically if such models use different markup schemes [22].

Ontology-based models

Ontologies represent a description of the concepts and relationships. Ontologies incorporate semantics into XML-based representation or Resource Description Framework [64]. Ontology-based context models are fairly common, because of the formal expressiveness and the ability of ontologies to support reasoning. However, is hard to construct complete ontologies and avoid the

ambiguity in the ontology [61]. Also, the information retrieval can be computationally intensive and time-consuming when the amount of data is increased [22].

Graphical models

These models are mainly based on the Unified Modeling Language (UML) [65] and Entity-Relationship Model (ERM) [66]. UML is a standardized general-purpose modeling language used in software architecture description, which can represent the user and context aspects, as well as its relations. Graphical models are capable of expressing the modeled aspects by graphical diagrams, and regarding expressive richness, graphical modeling is better than markup and key-value modeling as it allows relationships to be captured into the model [22].

Object-Oriented context models

These technique models the data by using object-oriented techniques, this offers the full power of object orientation like encapsulation, reusability, and in-heritance. In most cases, this model technique encapsulates the processing of context at the object level, and allows instances to access context and user in-formation by inheritance mechanism [61]. The main flaw of this technique is the lack of direct model validation, and when complex models are created, it may not be supported by limited resources hand-held devices [62]. Nevertheless, as most of the high-level programming languages support object-oriented concepts, these models can be integrated into context-aware systems easily. This makes object-oriented modeling to be used as code base, run-time, manipulation and storage mechanism for user and context models.

As there are various model representation techniques, each with their advantages and disadvantages (summarized in Table 2.1), a challenged to overcome in the development of our proposal was to choose the one that better suit the need to represent a data model for a context-aware recommendation systems framework. We opted to create a combination of graphical and object-oriented modeling, which allowed us to create a reach and expressive data model that contains the intrinsic relationships of the user, context and items information, and can be easily understood by developers, designers and researchers as it is expressed using the semi-formal UML class diagrams.

Table 2.1: Context representation techniques comparison

Technique	Pros	Cons
Key-Value	Simple	Not scalable
	Easy to manage when data is small	Difficult to manage complex models
	Flexible	Do not support relationships
		Hard to retrieve information
Markup scheme	Flexible	Structure is specific for the application
	Structured	Complex to represent multi-level information
	Well validated formats (RDF, XML)	Hard to retrieve information
Ontology based	Strong validation mechanisms	Construction is complex
	Support semantic reasoning	Information retrieval is complex and computationally expensive
	Support expressive representation of information	Hard to avoid ambiguity
Graphical	Strong support for relationships	Not standardize structures, but governed by design principles
	Validation is possible through constrains	Querying can be complex
	Easy to interpret by humans	Interoperability between different structures can be complex
	There are tools to translate (UML) models into code	
Object-Oriented	Strong support for relationships	Not standardize structures, but governed by design principles
	Can be well integrated into programming languages	Lack of direct validation
	Processing tools are available	Too complex models can may be computationally expensive
	No need to learn a new language or representation	

2.5 State of the Art

From an *infrastructural* perspective, context provides computing devices with information about their environment as provided by other system components. In order to provide such information, context needs to be classified in different 'types' or 'dimensions' of context, e.g. physical, computational, etc.

Dourish [53] introduces a taxonomy of context, according to which context can be classified into *representational* or *interactional* view. The representational view assumes that the contextual attributes are identifiable and known a priori and, hence, can be captured and used within the context-aware applications. In contrast, the interaction view assumes that the user behavior is induced by an underlying context, but that context itself is not necessarily observable. As the interactional view of context is an approach borrowed from psychology [43] that considers context non-observable, it cannot be used to explicitly store the context information into a computer system. Therefore, in this work, as most of the works on context-aware recommender systems are based on the representational view of context, as this approach allows to identify context attributes and to model it.

The areas of user modeling and context representation are well-established research topics by themselves, next we review some of the most related publications on modeling user or context information in software systems.

Heckman [40] introduced the General User Model Ontology (GUMO) for the uniform interpretation of distributed user models. GUMO is intended to represent the information of the user for adaptive systems, the proposal presents a long list of user aspects to be considered in such system organized into 12 *Basic User Dimensions*. As GUMO is created at an ontological level, it does not present further organization of the user characteristics or any architecture or implementation design that could help CARS developers in their effort of designing a context-aware user model.

Kaklanis et al. [67] proposed another user model in their efforts of the Virtual User Modeling and Standardization 'VUMS'. Their proposal models user preferences for graphical interfaces, as well as some cognitive and physical human abilities. As the proposal is aimed at modeling people with disabilities and elderly people, the user aspects are limited to such interest. Therefore, this model would be of little help when designing a context-aware user model for CARS.

Jawaheer et al. [19] classify the different types of user feedback as a source of information for user modeling in recommender systems. However, they do not describe what user or context information must be included in such a model. They conclude the work with a series of future research challenges, including the need for a unified user model for recommender systems.

Respect to context modeling, Zimmermann et al. [68] defined five fundamental categories for context information: Activity, Time, Relations, Individuality, and Location. They describe such categories as the design space of context models for context-aware application to build upon. Later on, Verbert [29] through an extensive survey on context-aware systems, increased the number of categories for context information to 8, namely, Computing, Location, Time, Physical Conditions, Activity, Resource, Social Relations, and User. Unlike Zimmermann, Verbert presented a series of subcategories for each context category, e.g., Computing is categorized into *software*, *hardware*, and *network*. However, a low-level detail of what information about context should designers and developers consider when creating their own context-aware user model has not been described by either proposal.

As the need for tools that help in the design and development of CARS is known in the literature, some proposals have emerged proposing tools for that matter.

Hybreed [11] is a CARS recommendation framework, created to help developers build recommender systems by providing a large set of implemented algorithms for traditional as well as context-aware recommender systems. Hybreed is based on a very generic notion of context, which allow developers to exploit a broad range of information as context aspects. In terms of modeling the information of user and context is were this proposal stands short, as it does not described what

information could be used, besides they uses a restrictive key-value pair approach to store all the information related to user, context and the relations between them. In this regard, authors acknowledge the need of supporting a more complex data modeling approach to support sophisticated user models.

Mettouris & Papadopoulos [21] present a tool designed to facilitate the development of reusable user models for CARS. The researchers publish the proposal as a web application, where developers can describe the context aspect that will be included in their model design. Even when their works [21], help developers as a tool to enlist context aspects and the possible values of each aspect, the work of identifying the aspects that will be included in the model is left to designers and developers. This tool does not allow to specify the data type of each aspect or relations between aspects, neither the proposal uses a formal or semi-formal notation (such as UML or XML) to help to take their list of aspects closer to a model design. Mettouris proposals [21] differ from GUMCARS proposal as follows: their tool is intended as a place to register, test and share a list of context aspects that later can be used to create a model, and our proposal is closer to design and implementation phases as it presents the conceptual design and the UML model of a context-aware user model.

CHAPTER 3

A GENERAL USER MODEL FOR CONTEXT-AWARE RECOMMENDER SYSTEMS

This section describes the creation of a generic user model for context-aware recommender systems that represent the core of this paper.

GUMCARS address the problem of lack of structures for CARS data management (described in section 1.1) by providing a model that structures the information needed by CARS. The goal of GUMCARS is to represent the information of the user, context, and items, which can be used by CARS systems either as the input for recommendation algorithms or by the system itself to improve the interaction with the user.

Based on Dourish [53] representational view of context, GUMCARS is formed by a finite set of user, context, and item aspects. The goal of GUMCARS is to structure the aspects that can be used by CARS systems either as the input for recommendation algorithms or by the system itself to improve the interaction with the user.

GUMCARS can be used by researchers as a basis for future model developments, as it contains a holistic view of the information used by the research community to generate context-based recommendations, by designers of CARS as the model can serve as a guide when selecting the

aspects to create their own model, and by CARS developers as the model is expressed from a conceptual perspective, and also from a software architectural point of view, and is ready to support the information needed by most common recommendation domains.

The creation of GUMCARS consisted on 4 main steps: (1) to create a taxonomy that organizes CARS information; (2) to gather the information aspects that will constitute the model; (3) to propose an architecture to organize such aspects; and (4) to validate the created proposal as as depicted in Figure 3.1. The rest of this sections detail each of four steps, followed by the conclusions of this chapter that discuss how this proposal attends some of the thesis objectives and goal.

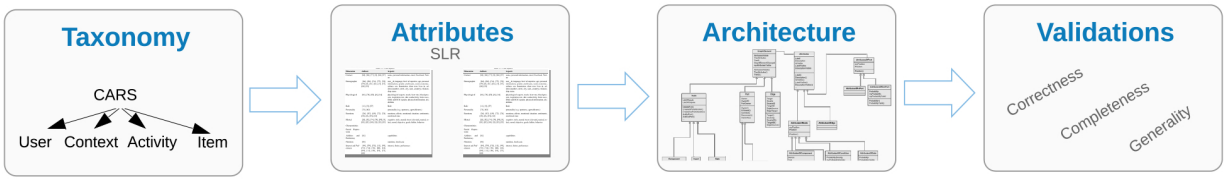

Figure 3.1: Steps followed for GUMCARS creation.

3.1 CARS Information Taxonomy

In a relatively short time of CARS existence, a lot of different proposals have been created [3], either to test new algorithms or to measure how including certain information can help the recommendation process to generate better results. In most cases, each proposal arbitrary selects which information to take into account.

In order to create a general user model for CARS, and before the identification of what attributes will be included in our general model, we needed to define a classification of concepts involved, that can be used by the model as the base to structure the information it contains.

In this section, we describe a taxonomy proposal of the information that CARS use to generate context-based predictions. This taxonomy represents one of the contributions of our work, as it organizes the high-level information of the user, context and items that CARS need to work, and to the best of our knowledge is the first taxonomy for CARS information.

The taxonomy comprises four categories of information: (1) User information, (2) Context information, (3) Activity information, and (4) Item information. Next, each category and its corresponding sub-categories are described.

3.1.1 User information

This top level category represent all the information of the user that describes him/her as a person, as such information is heterogeneous, and based on [40], we further organize the user category into eight subcategories: *Physiology*, *Contact information*, *Personality*, *Emotion*, *Role*, *Interest and Preference*, *Demographic*, *Mental*.

Compared to Heckman's [40] list of 12 dimensions of user information (contact information, demographics, ability and proficiency, personality, characteristics, emotional state, physiological state, mental state, motion, role, nutrition, and facial expression); we do not consider a category for user's characteristics as we consider (supported by Heckman's [40] itself), that there is no clear separation between personality traits and characteristics; we propose to support both information in the *Personality* subcategory. Heckamn uses ability and proficiency dimension to support information about abilities and disabilities of the user, such information is supported in the *Physiological* subcategory of our taxonomy. Heckman uses the motion user dimension to represent whether the user is *walking*, *sitting*, *goingUpStairs*, etc. We consider such information to be part of the activity of the user, and is considered in the *Activity* information category (described in 3.1.3). The last dimensions we do not include in our taxonomy are *Nutrition* and *Facial expressions*, as we considered them too specific and can increase the computational complexity of the model and most important, will make the model more cumbersome and harder to understand.

We also include a category called *Interest and Preference*, as CARS literature expresses that end users will consider useful a recommendation only if it fits in his/her interests and preferences [3].

Next, the proposed organization of user information category is depicted in Figure 3.2, and then each subcategory is briefly described.

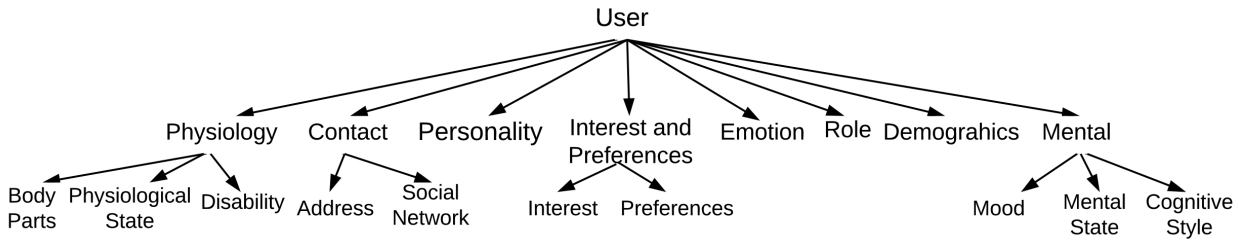

Figure 3.2: Proposed taxonomy of user information used by CARS.

Physiology

Physiological aspects represent the information about user's body and its functionality. This information is further subdivided into:

- *Body Parts*: This represents information about parts of the human body. The body parts subcategory is subdivided into *Head*, *Hand*, *Foot*. And could the list of body parts of humans as described in [69].
- *Physiological state*: Generic class to store information about physiological states, whose attributes are *name*, *level* and *isNormal*, these attributes can be used to represent states like respiration, blood pressure, heart rate, etc.
- *Disability*: Superclass for specific disabilities classes. As designing a class for all the existing disabilities it's outside the scope of this work, the model currently included only *ColorBlindness* and *Musculoskeletal* disability classes as a probe of concept, specific disabilities can be added as needed by specific projects.

Contact information

For CARS, it is important to know whom is the user interacting with the system, in order to provide tailored recommendations [3]. Contact subcategory refers to user aspects that identify a person (such as *name*, *last name*, and *email*), and the user's address. This information is available in most commercial systems since the user has to create an account and provide his data. Contact information is further subdivided into:

- *Address*: Used to represent the data related to the user's address, which could include information such as *house number*, *street name*, and *country*.
- *Social Network*: Represents the contact information of the user in social network sites, which can be used by recommender systems to infer user preferences from its social activities as done by [70] and [71].

Personality

Describe permanent or very slow changing patterns that are associated with a person [72]. CARS can use the personality information in to decide what items will fit better to the user, for example, [73] use the user personality as an important factor in their travel CARS. Certainly, personality subcategory can be subdivided with the list of personality traits, but as there are several personality models in psychology like Big Five [74] and Cattell's traits of personality [54], we opt to include the basic elements in the taxonomy to support all the personality models, and not to stick with any particular model.

Emotion

Emotions are subjective human experiences [20]. Even when emotion information may look similar to personality, they are different terms of duration of its contained information. Emotions tend to be closely associated with a specific event (context), and have a short duration of minutes up to an hour [40], while personality reflect long-term user characteristics.

Role

This user subcategory represents information about the role the user is playing at certain moment. According to [75], the user can take several roles and frequently change between them, for example, a user can visit a town as tourist or businessperson and a CARS should be able to recommend different places to visit depending on such role.

Demographic

This subcategory represent user's demographical information such as *age* or *family status*. This information can be used by CARS to improve the recommendations, for example [76] uses demographical information, such as *gender* and *socio-economic* information in a context-aware music recommendation system. Demographical information along with interest and preference can be used by CARS to improve the interaction of the user with the system, using such information to adapt the user interface accordingly as presented in [54].

Mental

Used to describe the user's state of mind, this subcategory is subdivided into *mood*, *mental state* and *cognitive style* subcategories that can be used by CARS to provide more tailored recommendations to users. For example [70] uses the user *mental stress*, and In [77] uses the user mood in their CARS.

Interest and Preference

This user information category allows to explicitly store user preferences for, or interest in some type of items. In a CARS presented in [78], the interest of the user is used to recommend places and events to visit accordingly, a user with music interest will be recommended to attend concerts, while a user with shopping interest will be recommended to visit near shopping malls. Similarly the CARS of [78] also implicitly collects the preferences of the users, which help the system to decide what specific type of place to recommend, in the music-lover user example, the system could use the

preferences of the user to decide whether recommend country, rock or classical events. This user information category is further subdivided into:

- Interest: Represents the interest of the user in certain topics or items.
- Preference: Represents the preference of the user for certain items in relation with certain context.

3.1.2 Contextual Information

This category represents information about the environment that surrounds the user, as well as other information about the items or the recommendation system itself that can be used to characterize the situation of the elements considered in the CARS. We organize the context information into the following six subcategories: *Computing*, *Location*, *Time*, *Physical condition*, *Resource* and *Social relation*.

Compared to the organization of contextual information proposed in [29], that describe a list of 5 types of context like *Computing*, *Location*, *Time*, *Physical Condition*, *Activity*, *Resource*, *User*, and *Social Relations*; we do not consider a subcategory for the activity information, as we consider such information in a category at a higher level. We do not consider the user information to be part of the context category, as we have a more detailed categorization of the user information, and is supported in a top-level category of the proposed taxonomy. The similarities and differences between the matching types of context from [29] and the proposed taxonomy are discussed along with the description of each subcategory.

Our taxonomy support other proposals of context information organization, like the presented in [79] that argue that as part of the contextual information is important to include the interactions between the environment (supported in *Computing* and *Physical Conditions* subcategories), the user (supported in *User* category), their tasks *Activity*), and other users (*Social Relation*).

Next, the proposed taxonomy for context information category is depicted in Figure 3.2, and then each subcategory is briefly described.

Computing

This subcategory represent the information that describe computational elements. Following [29], this subcategory is further divided into:

- Software: This sub-subcategory describe the characteristics of software systems that may be available to the user under a certain context, or software systems that the user is interacting

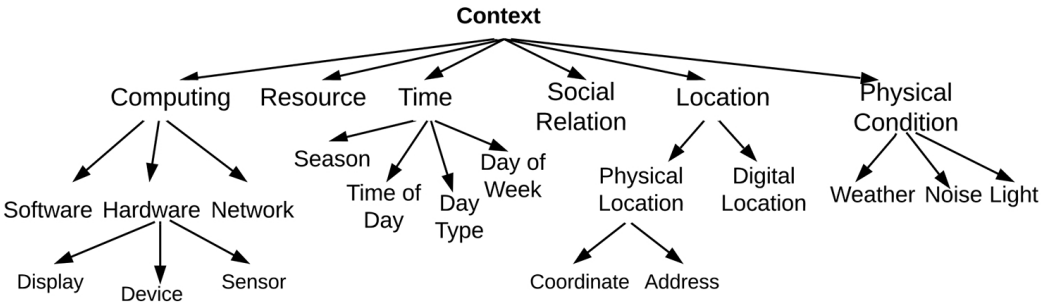

Figure 3.3: Proposed taxonomy of contextual information used by CARS.

with at certain moment. Let us consider a CARS that is recommending persons to a friend on different social networks, for such system, the information about what applications the user has installed on his smart-phone may be very useful.

- Hardware: Comprises information about the hardware as a physical item, such as *Vendor* and *Model*. We follow the Composite Capabilities/Preference Profile (CC/PP) [80] to further organize the hardware sub-subcategory into *Sensors*, *Display*, and *Device* that represent all the computing devices (like tablet, laptop and smart-phone) and include information of the minimum components of the computer device like *RAM*, *Processor* and *Storage*.
- Network: Information related to properties of the networks that are being used by computing devices. As for uses of *Network* information, CARS can take into account if the user's mobile device is currently on WiFi connectivity or via the cellular network in order to recommend High Definition or Standard Definition videos.

Resource

Resource information model relevant characteristic that describes elements and the services that items or places provide, for example, if a restaurant has wheelchair ramps, or if a hotel has transportation to and from the airport. Resource sub-subcategory of context can also be used to describe digital resources that are relevant to users, for example, a learning material can be used by a CARS to recommend learning topics to students.

Time

Time subcategory represent information about the time of the day, time of the week or time of the year (season), this information allow CARS to record the moment that some actions took place. The

use of *time* information in CARS is very common, as its values are easy to gather and have a deep impact on the user decisions [81].

Social Relation

This subcategory of contextual information refers to social associations or connection between the user of the system and other persons that may or may not be part of the CARS. The user social relations can help CARS to tailor items specific to current user companion, or to infer user preferences based on the knowledge the system has about other users related to him/her [82].

Location

In the proposed taxonomy the location subcategory refers to information that relates a user or an item with a physical or a digital location. CARS use location information to recommender user items like restaurants or movie theaters near him. Location is so often used in CARS, that there is whole research topic called location-aware recommender systems [83].

- Physical Location: Comprises information used to identify a specific point in the world, using the coordinates (latitude and longitude) or an address.
- Digital Location: Refer to information of where a digital resource (like a web page, video or song) can be located on Internet. A common example of digital a locator is a URL (Uniform Resource Locator) that specify the location a computer in a network.

Physical Condition

Describe the environmental conditions where uses or items are situated at a certain point in time. Physical condition describes physical conditions external to user or items like the level of crowdness and the level of traffic. This subcategory is further subdivided in:

- Weather: Represents weather information that CARS can use to better tailor suggestions, for example, the weather is important when recommending places to visit as recommending an outside place with a rainy weather may not be well received by users [41].
- Noise: Refers to information about the level of noise in a certain place, in a given time.
- Light: Used to represent the light level as well as the light source of the physical environment.

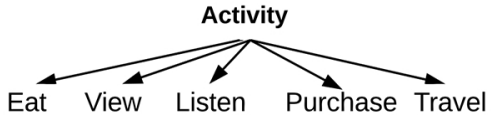

Figure 3.4: Proposed taxonomy of activity information used by CARS.

3.1.3 Activity Information

This information category represents the activity performed by the users of the system and relates such activity to the user and the context.

The activity of the user can help CARS perform more accurate recommendations, for example, Spotify, a music service and recommender system identity when the user is running or biking and play inspirational music to keep him going.

Unlike other proposals like [29,40] that consider the activity information as part of the contextual information, we create a category for such information at the same level of *User* or *Context* information.

Next, Figure 3.4 graphically describe the taxonomy of activity information category, and then each subcategory is briefly described.

3.1.4 Item Information

This category represent the information about the items of the CARS catalog like *Video*, *Book* and *Audio*. The importance of items information for CARS is paramount as items along with the transaction log are the base information for the CARS algorithms [84, 85].

For the item information category, not much details are presented in the taxonomy, as the information of each item varies according to the it type, for example, the information required for the *Movies* is different to that of *Hotels* and *Places*. Nevertheless, the taxonomy process this top-level category to organize such information despite the specific requirements of each project.

Having the proposed taxonomy for the CARS information, the next step towards the creation of a generic user model for CARS is the identification of the information aspects that will form the model.

3.2 Gathering of CARS information aspects

GUMCARS intends to structure a finite set of user, context, and item aspects that can be used by CARS as input information for the recommendation algorithm. Having presented a taxonomy for CARS information, the next step towards the creation of GUMCARS structure was the identification of such specific information aspects, for which, a literature review of CARS was performed and presented in [86].

Rather than using a traditional literature review, we used a Systematic Literature Review (SLR), following the methodology proposed by Kitchenham [87] since it is a rigorous and well-defined method in the field of software engineering. The SLR is a formalized and repeatable process to document relevant knowledge on a specific subject for assessing all available literature related to specific research question(s) [88]. The steps presented in the Kitchenham methodology are documented below.

3.2.1 Planning the Review

As we determined before, there was no systematic review in the field of user model for CARS; however, the increasing number of papers on CARS is appearing in many disciplines (such as ubiquitous and mobile computing, e-commerce, marketing, etc.) [89] can be taken as evidence that the importance of contextual information has been recognized by researchers and practitioners [90]. We consider that identifying what user's and context's information had been used in CARS to do context-based recommendations can be beneficial for future development and CARS research. Hence, we identified the need for a SLR that enlist the user and context information past studies has used to generate recommendations.

The main steps described in the Kitchenham methodology followed during the SLR execution are documented below.

Research Questions

The goal of the SLR here described is to identify the user and context information that has been used inside CARS to generate recommendations, the research questions addressed by the SLR study are:

SLR_RQ1 Which user aspects has been used in CARS?

SLR_RQ2 Which context information has been used in CARS?

SLR_RQ3 What items are the CARS recommending?

Bibliographic source

We chose the following bibliographic databases as source of information for our reviews: ACM, IEEExplore (IEEE), Science Direct (SD) and Springer Link (SL). The search were limited to journal and conference proceedings papers that were published within January 2012 through July 2016.

3.2.2 Conducting the Review

Through this step, a search is performed in the previous selected bibliographic databases in order to gather all the publications that can be related to the subject of interest.

To obtained the best possible results, for this search were used tree synonyms for context-awareness (context-aware, context aware and contextual), two for recommendation (recommendation and recommender system) and tree terms referring to user information (user model, user profile and user aspect) as depicted in the Fig. 3.5. The search for publications was carried out manually, using the 12 possible combinations of the terms, and making an union operation of the 12 the result sets.

Topic 1		Topic 2		Topic 3	Results
context-aware	AND	recommender system	AND	user model	248
OR				OR	
context aware		OR		user profile	
OR		recommendation		OR	
contextual				user aspects	

Figure 3.5: Search strategies and Results

Selection of Primary Studies.

During this step, the queries are used to search for publications in the selected bibliographic sources, then the result set follow a filtering process to discard publications that fitted the queries, but that are not really relevant to the purposes of the SLR.

We use the following strategy to identify the relevant publications out of the paper list retrieved from the search.

1. *Title-based exclusion*: First, we review the title of the publication to eliminate the publications that are clearly out of the scope of the review. After this stage, we end up with 248 papers out of 196.

2. *Screening-based exclusion*: Using abstract, keywords, sections titles, figures, tables, and conclusion (if present) we eliminate the publication out of this review interest. We shrink the list to 70 papers.

3. *Full text-based exclusion*: Reading carefully the full paper, analyzing what it proposes and using the inclusion and exclusion criteria described below, we eliminate the papers that are out of the interest of this review. The results are 36 related papers.

During the three filtering steps, we also considered a few additional exclusion criteria:

- Only included papers written in English or Spanish
- We exclude books and posters.
- Personal expert opinions about what information can be used to generate recommendation but do not present experimental results.
- Publication related to CARS that do not describe what information about the user or context was used inside the CARS.

3.2.3 Obtained Results

This section presents the obtained results, and responds the SLR research questions. First Table 3.1 presents a condensed list of the publication considered relevant (passed all the filters), the last column (*Recommending*) refers to what type of item(s) the publication's CARS are recommending, such information respond to **SLR_RQ3**. Also, this table shows that the most common recommended items are *Movies* and *Music*, although CARS are used to recommend an broad set of items.

Then, the obtained aspects for user and context information are presented, but first an example of how the aspects where mapped to the taxonomy prior presentation.

Table 3.1: Resulting publications from the data extraction and filtering

Ref	Year	Database	Recommending
[38]	2015	SL	learning materials
[91]	2015	SL	learning materials
[92]	2014	SD	photos
[93]	2013	ACM	music
[94]	2015	SL	music
[95]	2014	IEEE	movie
[96]	2014	SL	movie
[29]	2012	IEEE	learning materials
[97]	2012	SD	music
[77]	2015	SL	music, movies
[98]	2015	ACM	music, movies
[76]	2015	ACM	music
[99]	2012	IEEE	documents
[11]	2014	SL	shoes
[100]	2014	SL	
[101]	2014	SD	
[102]	2014	ACM	
[103]	2012	SL	task, places
[8]	2015	SD	music
[104]	2014	SL	
[70]	2014	SL	music
[14]	2014	SD	music
[14]	2014	SD	music
[105]	2013	ACM	plates
[83]	2015	IEEE	places
[106]	2016	IEEE	
[107]	2015	IEEE	indoor elements
[16]	2015	IEEE	music
[108]	2016	ACM	movie
[75]	2016	ACM	news
[91]	2016	SL	movie
[41]	2016	SL	news
[73]	2016	SL	hotels
[27]	2016	SL	movies
[78]	2016	SL	places
[109]	2016	SL	places

Mapping results into CARS taxonomy

The SLR yielded a list of aspects and values for such aspects, to present such results we first mapped each element into the information categorization described in the taxonomy. To perform the mapping, we use the description of each category, subcategory, and sub-subcategory as described in the taxonomy and the descriptions given by CARS authors when mentioning the used aspects to select where to assign each attribute.

Next, Table 3.2 show some examples of aspects found in the reviewed literature, and how they were mapped into the taxonomy elements.

Table 3.2: Example of mapping aspects into the taxonomy.

Ref	Aspect	Correspondent location in taxonomy		
		Category	**Subcategory**	**Sub-subcategory**
[97]	*name*	User	Contact	Fist, Last name
	mood	User	Mental	Mood
	temperature	Context	Physical Condition	Weather
[38]	*song*	Item	Song	
	battery level	Context	Computing	Device
[41]	*season*	Context	Time	Season
	companion	Context	Social Relation	
	emotion	User	Emotion	
[11]	*shoe size*	User	Physiology	Body Part
	movie	Item	Movie	
	weather	Context	Physical Condition	Weather

Uses's Aspects Considered by CARS

The user aspects found in through the SLR were mapped into the proposed taxonomy using the provided description of each level of the taxonomy, as well as the description given by the authors when mentioning the each aspect.

Next, Table 3.3 present the user information found in the reviewed publication, this table answers the **SLR_RQ1** of this systematic literature review (*Which user aspects has been used in CARS?*). Column 1 present the 12 categories used, column 2 shows the references to the publications that considered such type of information useful for context-aware recommendation generation, and finally column 3 shows the list of aspects found in the publications that were mapped into the corresponding category.

Table 3.3: User aspects

Dimension	Authors	Aspects
Contact	[103], [29], [96], [11], [103], [75]	name, personal information, email, Facebook, Twitter
Demographic	[103], [29], [93], [96], [97], [77], [11], [100], [103], [8], [106], [107] [73]	user_id, language, level of expertise, age, personal information, gender, profession, socio-economic, culture, sex, hometown, show size, lives in, address(number, street, city, state, country), relationship status
Physiological	[103], [93], [76], [11], [103]	physiological aspects, needs, heart rate, blood pressure, respiration rate, skin conductivity, brain wave, EMA and ECG signals, physical information, disabilities
Role	[14], [8], [75]	Role
Personality	[97], [78]	personality (e.g. openness, agreeableness)
Emotions	[29], [70], [38], [91], [97], [76], [11], [103], [41]	emotions, affects, emotional situation, sentiments, emotional state
Mental	[29], [70], [93], [97], [77], [11], [100], [101], [103], [8], [16], [73]	cognitive style, mental stress (elevated, neutral, relax), mood, objective, goals, habits, behavior
Interest and Preferences	[103], [98], [94], [14], [38], [91], [93], [76], [99], [102], [13], [14], [105], [109], [75], [78]	interest, desire, preferences

Context's Aspects Used in CARS

Similarly, to the process follow to categorize the user information, the context aspects found through the SLR were mapped into the *Context* information category of the taxonomy. The results is shown in Table 3.4, which can be used to answer the **SLR_RQ2** (*Which context information has been used in CARS?*) that drive the SLR. Column 1 shows presents the context categories, column 2 shows the references to the publications, and column 3 show the list of context aspects mapped to the corresponding category.

Table 3.4: Context aspects

Dimension	Authors	Aspects
Computing	[29], [14], [91], [92], [93], [95], [101], [103], [8], [106], [41], [41]	network (GPS, Wifi, Bluetooth, RFID, RF, 3G, 4G), hardware, software, device, incoming calls, system service, notification, battery charge, PC, mobile phone, tablet, network bandwidth, display resolution, network bandwidth, display resolution, storage capacity, RAM memory, OS, device type, sensors (GPS, accelerometer, gyroscope, illumination sensor)
Location	[103], [29], [14], [93], [95], [96], [97], [97], [76], [99], [11], [100], [101], [8], [14], [83], [106], [107], [108], [109], [41], [41] [78]	location (in-door, out-door, latitude, longitude), place (home, shops, coffee shops, bus station, train station, office, school, sport center, store, movie theater, restaurant), latitude, longitude, GPS, country
Time	[103], [29], [98], [94], [104], [38], [91], [92], [95], [96], [97], [77], [76], [99], [11], [100], [102], [103], [8], [14], [105], [83], [106], [107], [16], [41], [41], [27], [73]	time (morning, lunchtime, afternoon, evening, night), time of day (morning, noon, evening, night), day of week (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday) time of week, time of year, season, hour, minute, period, date
Physical Conditions	[103], [29], [14], [38], [91], [92], [93], [96], [97], [77], [11], [100], [8], [14], [83], [106], [16], [41], [73]	environment, physical conditions, temperature, weather, traffic, noise level, light level, humidity, crowd
Activity	[103], [38], [91], [93], [95], [96], [97], [99], [101], [103], [8], [107], [41], [78]	activity, event, walking, jogging, lying, running, standing, driving, sitting, ascending stairs, descending stairs, ascending in elevator, descending in elevator, charging cell, meeting, shopping, writing, cooking, swimming, reading, computing, exercise, travel, working, transit, walking, guided tours, movie show times
Resources	[29], [94], [73], [78], [108]	resource, transportation, parking lot
Social Relations	[29], [14], [91], [93], [95], [97], [77], [76], [99], [11], [103], [8], [14], [105], [106], [107], [16], [109], [75], [27], [73]	social relation, companion, relationship with, friends, family, parents

3.3 The GUMCARS Architecture

This section describes the architectural view of the GUMCARS, compared with the conceptual view (taxonomy), the architectural view goes a deeper in detailing what specific aspects is considered for each information category.

As modeling field states [110], a model should represent the information for which it has been designed, in a suitable abstraction form to aid an efficient and effective software construction. Therefore, GUMCARS architecture is designed following the object-oriented (OO) paradigm [111], and the semi-formal Unified Modeling Language (UML) [65].

GUMCARS architecture uses the same four top-level categories proposed in the taxonomy, which are called (packages) in the UML notation, the sub-categories and sub-subcategories are *classes*, and the aspects are *class attributes*.

3.3.1 GUMCARS User Aspects

This section describes the user attributes considered in the GUMCARS model proposal and organized following the proposed taxonomy of CARS information, specifically the user information category of the taxonomy. Though the attributes, the model intends to represent the information that describes the user as a persona, which could be relevant to any context-aware recommender system either to generate better predictions or to adapt its user interface to enhance the interaction between the user and the system.

GUMCARS model contains a total of 31 classes and enumerators, and 127 class attributes to represent the information of the user, more details of the class counting for this package is presented in Table 3.5.

Table 3.5: Number of classes and attributes of GUMCARS for User information

Subcategory	**Classes**	**Attributes**
Physiology	12	33
Contact	3	23
Personality	1	4
Interest and Preferences	3	10
Emotion	1	6
Role	1	3
Demographics	5	28
Mental	5	20
Total	**31**	**127**

Next, Figure 3.6 presents a UML class diagram of how GUMCARS organize the user information, and then each user information category is described. In addition, representative examples of what user aspects were found through the SLR and how they were organized in the proposed model are given. For a detailed documentation of each class and attribute please refer to[1], were the GUMCARS model is used as the core component of a user modeling framework for CARS.

Physiological Information

Physiological aspects represent the information about user's body and its functionality. Physiological aspects like *respiration*, *blood pressure* or *heartbeat* can be used by CARS as symptomatic information for higher level user dimensions like mood and emotions [40, 76]. The Physiological package contains the following classes:

- *Physiology*: Main class to represent information of the user body that relates to the rest of the classes in the package.
- *Physiological state*: Generic class to store information about physiological states, whose attributes are *name*, *level* and *isNormal*, these attributes can be used to represent states like respiration, blood pressure, heart rate, etc.
- *BodyPart*: Superclass for classes that represent parts of the human body such as *Foot*, *Arm*, *Head*, etc.
- *Disability*: Superclass for specific disabilities classes. As designing a class for all the existing disabilities its outside the scope of this work, the model currently included only *ColorBlindness* and *Musculoskeletal* disability classes as a probe of concept, specific disabilities can be added extending this superclass.

In the model diagram, the *body part* sub-subcategory presented in the taxonomy is specialized into four types of body parts. Certainly, a CARS recommending therapeutic exercises may benefit from a detailed description of the body parts (like the NASA report in [69]), and from a detailed modeling of each part capabilities (like in [67]). Nevertheless, GUMCARS aims at creating a generic model that can be used in a wide range of CARS domain, and only basic information (such as size) is considered for body parts like *Foot*, *Arm*, *Head* and *Hand*. For example, *Shoe Size* is considered by [11], while *Ring Size* and *Bracelet Size* attributes can be used by a CARS recommending fashion accessories like the ones proposed in [112, 113].

[1] http://bit.ly/GUMCARS_user

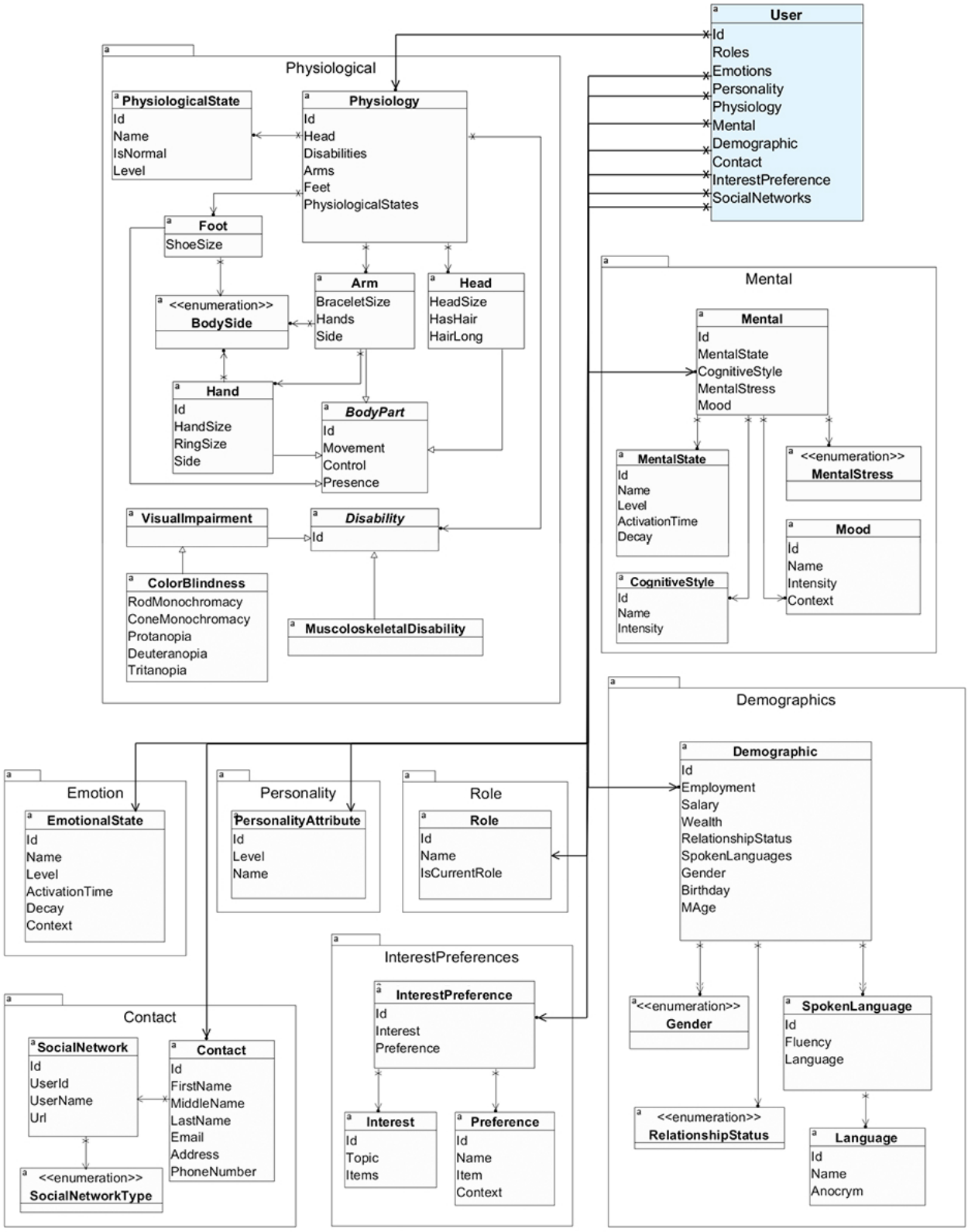

Figure 3.6: User aspects considered in GUMCARS.

Contact Information

Contact information refers to user aspects that identifies a person.

For CARS is important to know who is the user interacting with the system, in order to provide tailored recommendations [3]. User's address could be useful for CARS, for example, the *postal code* and *house number* can be used by a system that mails some products to the user, *Country* is used by Amazon[2] and Netflix[3] to filter the available items depending on the user current country.

The *Contact* package has the following classes:

- *Contact*: Main class of the package with attributes about the user contact information (*name*, *email*, *phone number*, etc.).
- *SocialNetwork*: Class to contain information of the social networks of the user. For the social information a separated class was created in GUMCARS, which allows to relate a list of social networks to the same user.
- *Address*: Used to represent the data related to the user's address, with attributes such as *house number*, *street name*, and *country*. The address class is packaged under the *Context* information, specifically in the *Location* package described later, but a reference of such class is created in the *Contact* class.

Personality

Describe permanent or very slow changing patterns that are associated with an individual [40]. Personality information is used for example by [97] as a way to improve the music preference predictions in their CARS. Even when exist several models of personality in the psychological literature, GUMCARS do not stick to any specific model, rather, only basic information is supported through attributes like *Level* and *Name* that can be used to support different personality models, or can specialized to fit a selected personality model.

Emotion

Emotions are subjective human experiences [20]. The Emotional state of the user is used by CARS authors like [11, 70, 73, 76]. This information is supported in by the *Emotional State*, as most authors only describe using the emotional state (*name*), except [76] that also considers the arousal and valence. They describe arousal as the level of activation of the state, and valence as whether the emotion is negative (like anger) or positive like (satisfaction). Such information is supported in

[2] http://amazon.com
[3] http://Netflix.com

GUMCARS by the *level* attribute of the *emotional state* class, where the arousal represent the value of the attribute, and the valence represents whether the value is positive or negative.

Role

Role information describes the different functions assumed or parts played by the user in particular situations. Role information is used less often that many other elements, in [75] the usage of role information is explicitly mentioned to be used as recommendation algorithm input. Also, authors in [114] and [14] used information about the role the user in playing when visiting a place, but not explicitly label such information as *Role*. GUMCARS supports the *name* of the role the user is playing or played, and if such role *is the current role*.

Demographic

Represent user's demographical information such as *gender*, *language*, and *family status*. Demographics is one of the most used information about the user by reviewed literature, for example [106] describe the importance of user *age* and *gender* in the recommendation result, in [94] the *employment* and *marital status* (called *relationship status* in GUMCARS), and in [70] the *language* along with other aspects of the user in their CARS. Demographic package contains the following classes:

- *Demographic*: Main class to represent demographic information of the user, such as *employment*, *salary*, *relationship status*, etc.
- *SpokenLanguage*: Class to represent the list of languages that the users speak, and the fluency on each one.

Mental

Mental information describes the state of mind of the user, including information about user's *mood*, *mental state* and *cognitive style* that can be used to provide more tailored recommendations. For example [70] uses an ECG signal meter to detect the heart rate variability in order to infer the user *mental stress* that later is used to recommend suitable multimedia content.

Following [76] in their structure used to support mental state information into a CARS, GUMCARS uses *name*, *level*, *activation time* and *decay* attributes to store the mental state information of the users. In [77] the user mood states like *active*, *happy* and *sad* is used as important information to decide what music to recommend to the user. Also, [100] uses moods like *normal*, *tired* and *shocked* along with other contextual information like *Weather* and *Location* in their CARS implementation. GUMCARS can support these implementations, using the *name* attribute of the *Mood* class, the

different moods can be represented, and using the *context* attribute, the contextual information (as considered by [100]) can be stored.

Mental package contains the following classes, thought to represent the basic information, which can be further derived into more detailed structures to suit project specific needs.

- *MentalState*: Class to represent the different mental states of the user.
- *CognitiveStyle*: Class to represent the way the user processes information, for this iteration, this class only represents the name and intensity of the *CognitiveStyle*.
- *Mood*: Class to represent the different *moods* of the user, their *intensity*, and *valence*.

Is important to notice the difference between *mood* and *emotion*, following [115] distinction: *moods* are said to be concerned with larger, longer lasting, existential issues about the person's life and how it is going, while *emotions* are apt to be brief.

Interest and Preferences

Information that explicitly represent the user interest and preferences. Interest information is used for example by [14] to recommend movies and trips, by [38] to recommend places and by [91] in a new CARS. Also, preference information is used by [83] and [93] in their CARS implementation, in both cases, the preferences of the user are linked to an item of the system, which is reflected in GUMCARS preference class. This are the classes contained in the package:

- *InterestPreferences*: Main class of the package that contains a list of *Interest* and a list of *Preferences*.
- *Interest*: Simple class to represent the user interest in a certain *topic* or in specific *items*.
- *Preference*: Simple class thought to be modified by developers to support the preferences structure they need. Currently, this class the structure to represent the preferences of the user for an item in a context.

3.3.2 GUMCARS Context Aspects

This section describes the context aspect modeled in GUMCARS and organized following the proposed taxonomy of CARS information, specifically the context information category of the taxonomy. Though this set of classes and attributes, GUMCARS intends to support all the contextual information needed by CARS to perform their recommendations.

GUMCARS model contains a total of 31 classes and enumerators, and 172 class attributes to represent the contextual information, more details of the class counting for this package is presented in Table 3.6.

Table 3.6: Number of classes and attributes of GUMCARS for context information

Subcategory	Classes	Attributes
Computing	12	67
Resource	1	2
Time	5	33
Social Relation	2	11
Location	6	40
Physical Condition	5	19
Total	**31**	**172**

Next, Figure 3.7 presents a UML class diagram of how GUMCARS organize the context information, and then each information category is described. Also, representative examples of what context aspects were found through the SLR, and how they were organized in the proposed model are given. For a detailed documentation of each class and attribute please refer to[4], were the GUMCARS model is used as the core component of a user modeling framework for CARS.

Computing

This data refers to computing information useful for CARS, following [29]. Information of computing devices like smart phones in used by [41], specially the information of sensors like GPS and accelerometers they possess. Also [93, 101] consider that storing the battery level of mobile computing devices is important for CARS to better decide what to recommend to user. To support this information, the *device* class is further specialized into *mobile device* class that can represent devices like tables, smart phones, laptops, etc. Also *sensor* class is created as an specialization of *hardware*. The *sensor* class also is useful to support other sensors information like blood pressure and heart rate sensors described in [76].

Authors like [93] and [108] used WiFi network information for recommendation purposes in their CARS; similarly, [8] and [96] describe the possibility of including cellular network information in their respective CARS. Such information about computer networks is supported by the *Network* class of the computing package.

Contextual computing characteristics can be classified into three areas:

- *Network*: Information related to properties of the networks that are being used by computing devices. As for uses of *Network* information, CARS can take into account if the user's mobile device is currently on WiFi connectivity or via the cellular network in order to recommend

[4] http://bit.ly/GUMCARS_user

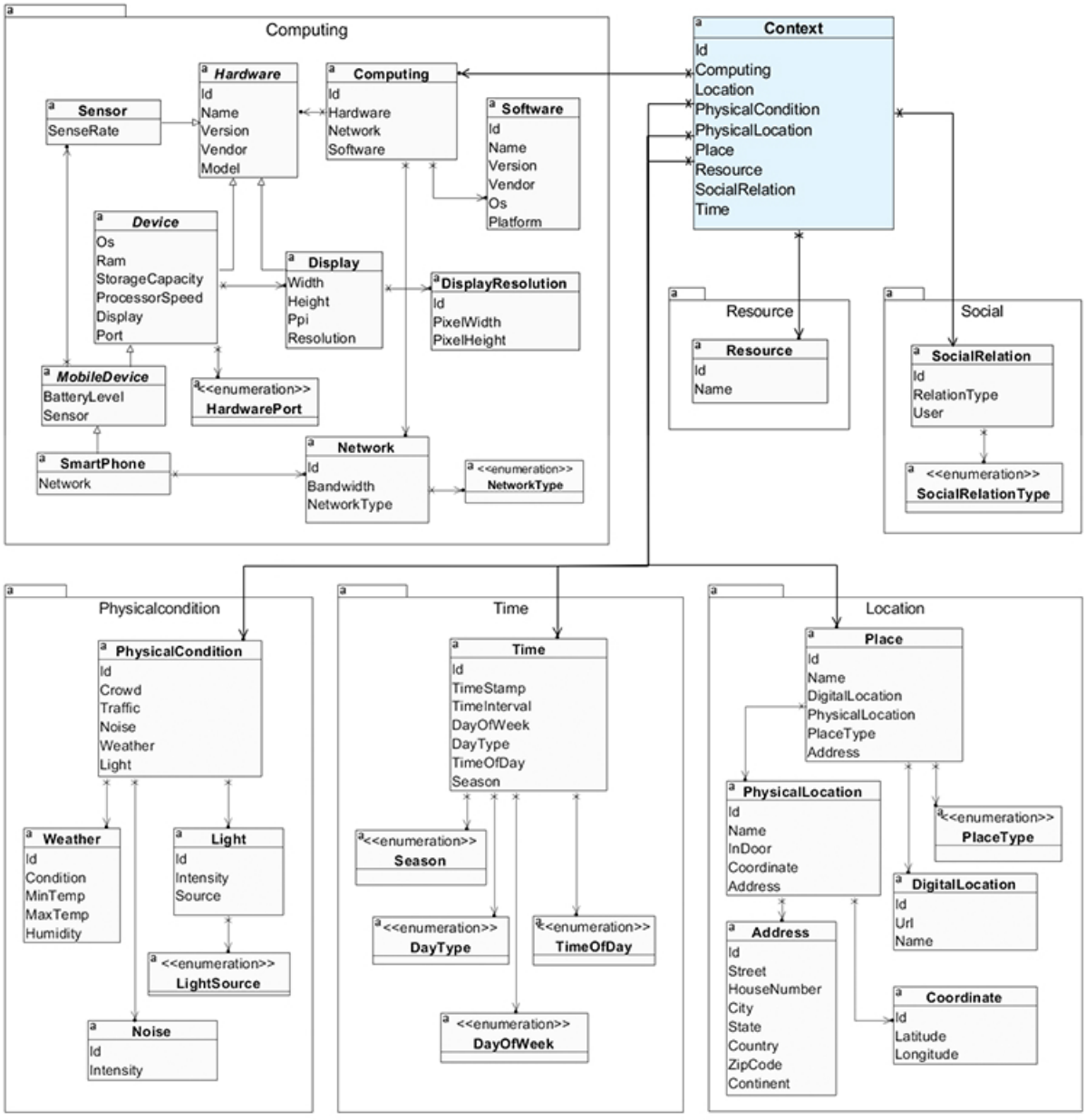

Figure 3.7: Context aspects considered in GUMCARS.

High Definition or Standard Definition videos. In music, for example, [93] uses the available network connectivity of the user device in their music CARS.

- *Hardware*: Comprises device capabilities, storage, display, etc. The proposal of [41] is an example of the use of this information into CARS, they use the device type (smartphone or tablet) where the user is interacting with the system as input information into a News Context-Aware Recommender System.
- *Software*: Describes the characteristics of software systems that may be available to the user under a certain context, or software systems that the user is interacting with at certain moment. Let us consider a CARS that is recommending persons to a friend on different social networks, for such system, the information about what applications the user have installed on his smartphone may be very useful, such information is supported in the GUMCARS through the *Software* area of Computing.

Resource:

Information about characteristics of the physical or virtual environment. Various type of resources are used as input information for recommendation algorithms, as reviewed literature suggest. Even when authors do not specifically label such information a *resource* information, they used different label to describe characteristics of places, for example [94] called *facility characteristics* and are used to describe a hotel services (gym, pool, wifi), [73] describe services provided by travel agencies like transportation to and from airport. In GUMCARS the information of physical and digital resources is supported by the *Resource* class at a basic level, but can be further specialized to fit specific project needs.

Time

Represents information about the *date*, *time* or less specific information about *season* or year. According to the findings through the SLR, *Time* is the most common information used in CARS. For example [41] uses the *time of the day* to tailor news to users as he identified a patter in his dataset where users read some type of news in the morning and a different type late in the day. Wu et al. [106] uses *weekday* and *weekend* categorization to recommend movies through their CARS, as they were testing if the movie preferences of their users varies depending of the day type.

The time category of the contextual information contains elements like:

- *DayOfWek*: Used to represent if which day of the week the interaction is taking place.
- *DayType*: Used to represent if the day is a *Weekday*, a *Weekend* day or a *Holiday*.

- *Time*: That can represent the time of the day.
- *Season*: Represents what of the four seasons of the year such interaction is taking place.

Social relation:

Refers to social associations or connections between the user of the system and other persons. The information about the user's companion is often used by CARS, for example [16] considers if the users was with a *friend*, *family* or *partner* when consuming an item recommended by their CARS. Authors in [106] argue that when the user is with his/her *children* is more prone to consume cartoons, while being with the *partner* is more likely to consume a romantic movie, such information can be easily exploited by a CARS recommending movies. GUMCARS *Social Relation* packages allows to relate a list of other users and specify the relation that each user has with the user in question.

Location

Location category refers to information that relates an item, user, or other context information with a geographical position. The location was often found to be used in CARS as decisive information in the recommendation generation process, for example, [109] and [78] present location-based CARS that use the latitude and longitude of the user to decide what to recommend to him, such information falls straight into the *Coordinate* class of GUMCARS. Authors in [8] use information about at time of the day (morning, launch time, evening, etc.) a show time will take place; GUMCARS support such information in the *Time of Day* class. In [104] the *Time of the Day*, *Day of the Week* and *Season* information is described as being easy to collect by CARS, later on such information is used by [16] in their CARS.

Location package contains the following classes:

- *Place*: Is an area with a *name*, it can be *DigitalLocation* or *PhysicalLocation*.
- *DigitalLocation*: Comprises information of where the *Place* can be located on Internet
- *PhysicalLocation*: Comprises information used to identify a *Place*, *Event* or *User* in the physical world.
  - *Address*: Used to represent the address information of the user with properties such from *house number* to *country*.
  - *Coordinate*: Refers to the *latitude* and *longitude* of the geographic coordinate system.

Physical conditions:

Describes the environmental conditions where the system or user is situated at a certain point on time. Physical condition information is used to identify the environment where the user or the item is situated so CARS can reach to such environment and make better predictions. One of the most used information about physical condition is weather, for example [96] uses the weather *condition*, [97] uses the level of *humidity*, and [91, 100] both use the *Temperature* in their CARS. Apart from weather information, [96] uses the level of *illumination* and level of *noise*, and [73] uses the *crowd* as information for the recommendation algorithm of their CARS.

This package contains the following classes:

- *Weather*. Class to represent weather information is the most common *physical condition* characteristic considered in the reviewed literature. Weather can be used for example in a travel recommender [73] to recommend *places* to visit according to weather preferences of the user.
- *Noise*: Refers to information about the level of noise in a certain *Place* in a given *Time*.
- *Light*: Used to represent the light level as well as the *LightSource*.

3.3.3 User Activity Information

We consider the *Activity* information outside of the *Context* information, as in the activities we relate information about the *User* that is performing it, and the *Context* where the activity is performed.

The information about the activities is essential for the recommendation process, such information is called transaction logs in [43, 116]. Recommender systems use the information of what item the user consumed and in what context such activity took place, as base information for the prediction generation [3]. A good example of how all this information is used is presented in [96], their CARS consider the activity of the user (running, driving, standing, etc.), along with some contextual information (like location, time of day, day of week) and information about what song the user is currently listen to decide what to recommend next.

With the *Activity* package, GUMCARS support the past and current activity of the user, the item consumed in such transaction, the contextual information, and the rating given by the user. GUMCARS contains an specialization of activity information called *Rated Activity* which in addition to user and context information, also include the rating value expressed by the user that reflects how much he/she liked that item. To clarify the basic difference between them, a *RatedActivity* will be used when the User interact with the *Items* that the CARS is recommending, for example an Activity can be ”John is running in the park on a cloudy morning”, and a RatedActivity can be ”Alice is

listening the *eye of the tiger* song while running in the park on a cloudy morning, and rated it with a 4 stars".

The *Activity* package contains the following classes:

- *Activity*: Main class of the packages, used to store simple activities information, or as a superclass for more specialized activities.
- *RatedActivity*: Is an abstract specialized *Activity* class, with information about the rating, and must be inherited to add information about an specific type of *Item*.
- *View*: An specialization of *RatedActivity*, represent the video viewing activity.
- *Listen*: An specialization of *RatedActivity*, refers to information of listening to a song or audio file.
- *Eat*: An specialization of *RatedActivity*, for storing the eating activity along with relations to a *Dish* item and optional *Restaurant* information.
- *Travel*: An specialization of *RatedActivity*, refers to the travels that a User performs.
- *Purchase*: An specialization of *RatedActivity*, for storing all the purchase record transactions and reference to the *Items* the user bought

Next, Figure 3.8 presents the UML class diagram, and the attributes of each of the classes that GUMCARS uses to represent the activity information.

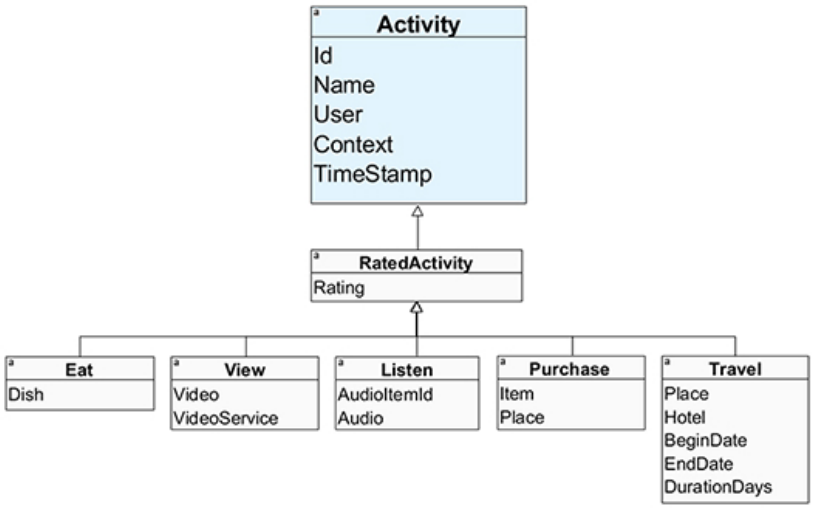

Figure 3.8: Activity - UML class diagram

3.3.4 Item Information

In addition to User and Context information, GUMCARS also includes base aspects related to the *Items* that a CARSs can recommend. These aspects are included in an *Item* package, which contains classes to represent the elements the reviewed publications are recommending.

The *Item* superclass is the one that all the elements to be recommended must derive from; this superclass allows the model to relate a *User* and some *Context* information with any *Item* by creating a relation to the superclass instead of a relation to each specific subclass.

The package contains the following *Item* subclasses:

- *Audio*: Considers information about the Title, Duration, Quality and Album of the audio file.
- *Book*: Includes properties like Author, Topic, Year of publication, and number of Pages.
- *Dish*: Includes information such as Calories, Price, a collection of *Photos*, and optional relations properties to a *Restaurant* where the User consumed such dish.
- *Hotel*: Contains a relation to a *Place* where the Hotel is located.
- *NewsArticle*: Considers the Topic of the article, optional relations to a *Place* that an article may refer to, and a collection of *Photos*.
- *Photo*: Contains an optional relation to a *Place*.
- *Restaurant*: Has references to a *Place* and a collections of *Dish*es from the menu.
- *Video*: Considers information about the Duration, Quality, release Year, Directors and Actors of the video/movie.

The *Item* package contains base classes and basic information on each class, and by no means, these classes are intended to be complete, rather these classes represent the bases and can be adapted to support the specific information of any project. Figure 3.9 shows a UML class diagram of the Item package, and the classes it contains.

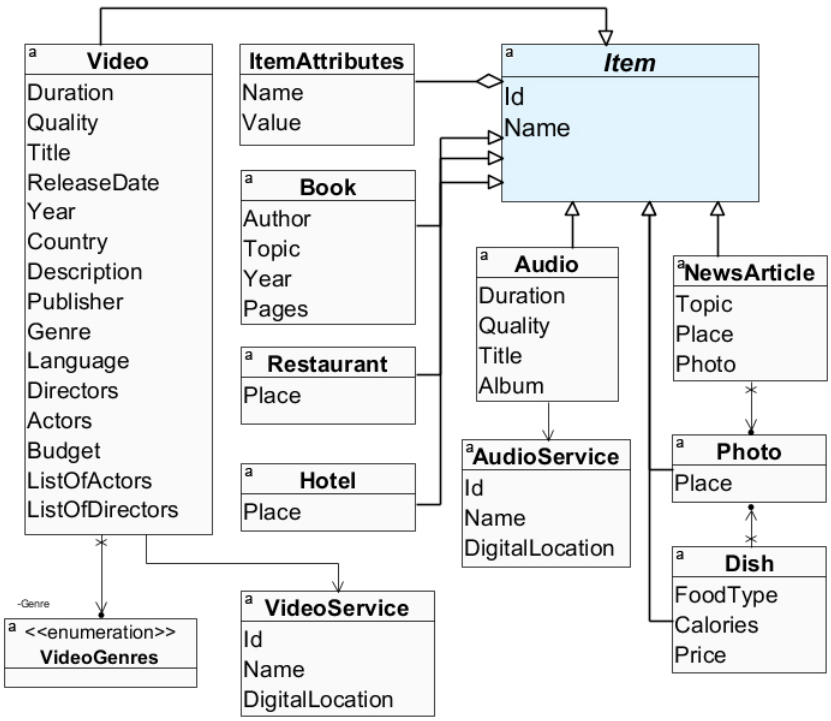

Figure 3.9: Item aspects considered in GUMCARS.

3.4 Validation of GUMCARS Completeness and Generality

This section describes the first validation of GUMCARS, where the model completeness and generality are validated.

According to [117], the quality of a conceptual model can be categorized in:

- Syntactic: Syntactic quality is a relation between the model and the language. The goal is *syntactic correctness*, which can be simplified as whether or not the model contains only statements that are valid in the language;
- Semantic: Semantic quality is a relation between the model and the domain. This quality category measures how well the model represents the important elements of the domain.
- Pragmatic: Pragmatic quality is a relation between the model and the audience's interpretation. This category measures how the intended audience interprets the models.

As all the attributes, classes and packages contained in the model were gathered from CARS literature, we consider that the language contained in model belong to CARS domain; therefore, we do not consider necessary to evaluate the *Syntactic* quality of the model. The *Pragmatic* quality

of the model refers on how the target users of the model will interpret it, even when we do not specifically evaluate such quality attribute, in Section 4.5 we evaluate developers perception of usability of the model and its surrounding framework.

In this experiment, we focus on evaluation the *Semantic* quality evaluation of GUMCARS. According to [118], the *Semantic* quality of the model (M) with respect to the target domain (D), can be defined by two quality attributes: *Validity* and *Completeness*.

- Semantic Validity ensures that the elements included in the model are relevant to the modeled domain.
- Semantic Completeness refers to how many of the relevant elements of the domain are included into the model. In [119], *Completeness* is also defined as *whether the model supports all the information required by target systems*.

As all the elements that constitute the model where gathered from CARS domain, we overlook *semantic validity*, and focused in assessing the *semantic completeness* of GUMCARS, which will reflect is the model is ready to support the information that CARS need to perform their recommendations.

This experiment also allowed us to validate the *Generality* of the model, that according to Moody [120] describes how wide is the application scope of the model. In this work, we define *Generality* as the ability of the model to support data from different domains.

Next, the material used to perform this experiment are described, followed by the method followed during the experiment, the results obtained, and a discussion of results.

3.4.1 Materials

For this experiment, we used Visual Paradigm 13 to open and explore the UML representation of GUMCARS. To open and execute C# code, the Visual Studio Ultimate 2017 programming IDE was used. We also used 8 datasets gathered from the literature. We selected only datasets that were created for recommendation purposes, and that contain data about the 3 main elements (*Users*, *Items* and *Context*).

The selected datasets vary in the number of instances (rows), the classes (aspects), and they correspond to different domains (*one* from *food* domain, *three* from *Hotel reservations and travel booking*, *two* from *movies* and *two* from *music* domain). Next, a brief explanation of each data set is presented.

Table 3.7: My caption

Dataset	Type	Attributes				Instances
		User	Context	Item	Total	
Japan	Food	3	14	17	34	800
TripAdvisor	Hotel / Travel	3	2	4	9	14,176
Expedia	Hotel / Travel	4	15	154	173	62,100
STS	Hotel / Travel	8	15	4	27	2,535
DePaulMovie	Movie	1	4	1	6	50,043
LDOS CoMoDa	Movie	9	8	14	31	2,296
ConcertTweets	Music	1	7	4	12	249,470
InCarMusic	Music	2	8	8	18	4,012

- *Japan*: A restaurant dataset created by [121]. It has aspects describing the restaurant, time, environment, and user's social information when rated each restaurant.
- *Trip Advisor*: A hotel booking dataset scripted from online reviews on tripadvisor.com by [122]. The dataset contains information about the hotel, contextual information, and location of the user.
- *Expedia*: A travel booking dataset from expedia.com that contains 150 aspects about the travel destination, aspects of the user and the context where the user search or booked the travel. This dataset was gathered from kaggle.com and posted by Expedia.
- *STS*: A travel dataset by [123]. This dataset is rich in user and context information, as has some aspects of the travel. This dataset includes information about the user's personality following the Big Five personality traits.
- *DePaul Movie*: A movie rating dataset by [124]. The data was collected from students, which were asked to rate movies in different time, location and with different companions.
- *LDOS CoMoDa*: A movie rating dataset by [125]. This dataset contains emotional information about the user while rating/watching the movie, as well as movie and context information.
- *Concert Tweets*: A music dataset by [126]. This dataset contains information about concert events, the user, and the context of the user when attending the events.
- *InCarMusic*: A music dataset by [127]. This dataset contains information about the song, the mood and driving style of the user while listening to the song, as well as other contextual information.

Next, Table 3.7 shows information about each of the 8 selected datasets.

In addition, to perform this evaluation, the implementation of GUMCARS is used, such implementation is part of wider project where here proposed model is being used as part of a CARS

modeling framework (described in Section 4). The implementation was created by exporting the UML representation of the model to a C# code library using NDepend automatic code generation tool. We consider the code to be just another representation of the model, as there are no changes between the architectural view of the model and the implemented code. In this experiment, as well as the described in 3.5, we opted to use the implemented model as a way to automatize the evaluations and avoid biases.

3.4.2 Method

First, we mapped all the possible aspects found in each dataset to their corresponding attribute in GUMCARS classes. This allowed us to identify which dataset aspect are, and what aspects are not supported by GUMCARS. An aspect is considered to be supported if GUMCARS contains an equal or similar attribute in their classes. For example, *Social* aspect of LDOS CoMoDa dataset refers to *the companion of the user to watch the movie*, and was mapped to *Context.SocialRelation.RelationType* enumerator in the model, that represents the same.

Then, a simple script was developed for each dataset, where the dataset file was read and its contained features are mapped into GUMCARS implemented representation, following the mapping created in the first step. During this step, no changes were made to GUMCARS code.

The *Completeness* of GUMCARS was calculated as the percentage of the total numbers of aspects contained in the dataset (D), that were successfully mapped into the model (M), i.e. $Completeness = M/D$. Each dataset aspect scores a one (1) into the D variable if a corresponding attribute exists in GUMCARS, and all of its values were fully supported by the model. In cases where a corresponding attribute exists in GUMCARS, but some of the values present in the dataset were not supported by the model aspect, such attributes scored a 0.5 into D and considered *Partially Supported*.

3.4.3 Results

A total of 31 user, 73 context, 206 items aspects, and 323,332 instances were contained in the 8 datasets, and were mapped (separately) into GUMCARS. Along the datasets, some aspects are repeated, being the most common *userId*, *itemId*, *rating* and *weather*. As a result, in this experiment, 22 unique *User* aspects were used, 64 *Context* aspects, and 194 *Item* aspects of 6 different type of items, namely *food*, *hotels*, *travels*, *movies*, *concerts* and *songs*, were used.

For *Japan* dataset, 6 of the 32 attributes were not supported, and 1 attribute (RelationType) was considered partially supported, as 2 (*Boss* and *Subordinate*) of the possible 6 values could not be mapped to GUMCARS's *RelationType* enumerator. Resulting in a *Completeness* value of

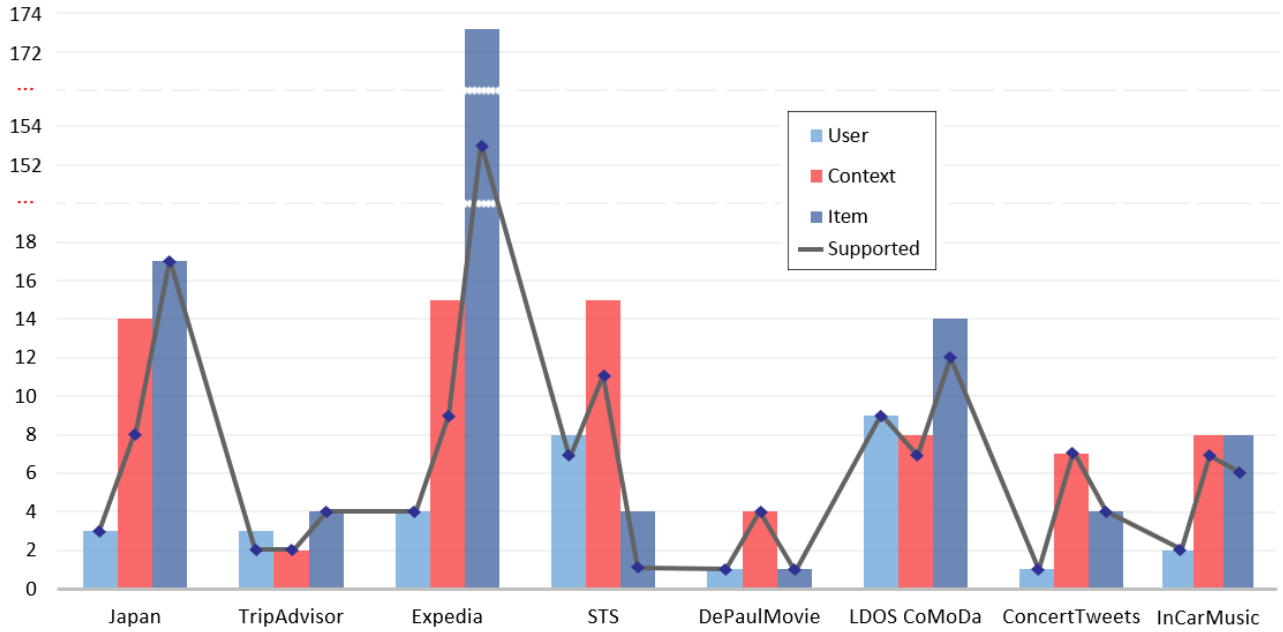

Figure 3.10: Relation between total (*bars*) aspects considered by each dataset, and number of aspects supported by GUMCARS (*line*)

0.81, meaning that the 81% of the attributes were supported. We got *Completeness* of 89% for *TripAdvisor*, a 96% for *Expedia*, and a 70% for *STS*. For *DePaul Movie* we obtained a *Completeness* of 100%, and a 90% for *LDOS CoMoDa*. In *ConcertTweets* dataset, we were able to map all the contained attributes to GUMCARS, thus a *Completeness* value of 100%. Finally, for *InCar Music* dataset we obtained a completeness of 83%, as 3 of its 18 attributes were not supported.

In general, GUMCARS supported the 93.55% of the user aspects, 75.34% of contextual aspects, and 96.12% of item aspects contained in the eight datasets used in this experiment. The obtained *Completeness* for each dataset as well as the not supported attributes are presented in Table 3.8. Also, Figure 3.10 graphically presents the total and supported attributes by *User*, *Context* and *Item* for each tested dataset.

Table 3.8: Results of mapping and loading dataset into GUMCARS model

Dataset	Total Attributes	Not Supported		Partially		Completeness
Japan	34	6	Budget Num. of Male (Partners) Num. of Female (Partners) Lowest Age (Partners) Highest Age (Partners) Partner Status	1	Relation type	0.81
Trip-Advisor	9	1	User time zone	0		0.89
Expedia	173	7	Orig_destination_distance Is_package (is a travel package) Srch_adults (num of adults) Srch_children (num of children) Srch_rm (num of rooms) Cnt (number of similar events) Posa_continent (point of sale)	0		0.96
STS	27	8	distance knowledgeOfSurroundings budget travel goal means of transport POI's category1 POI's category2 POI's category3	0		0.70
DePaul Movie	6	0		0		1.00
LDOS CoMoDa	31	3	Physical (health condition) Decision (to watch the movie) Interaction (n-th interaction)	0		0.90
Concert-Tweets	12	0		0		1.00
InCar Music	18	3	DrivingStyle RoadType Sleepiness	0		0.83

3.4.4 Discussion

The average value of *Completeness* for the 310 aspects tested was a 0.88 of 1, with a standard deviation of 0.09, which strongly suggest that GUMCARS is capable of supporting most of the aspects considered by the datasets.

In cases like *DePaul Movie* (6 aspects), *ConcertTweets* (12 aspects) and *Expedia* (173 aspects), we got a high level of *Completeness* supporting 100% of the aspects from the datasets.

The lowest *Completeness* value for a dataset corresponds to *STS*, which scored a 0.70, as 8 out of the 27 aspects considered in the dataset were not supported by GUMCARS. This dataset contains some very specific traveling aspects like *Travel Goal* and *Traveled Distance* that would have been difficult to foresee.

We consider that GUMCARS performed well in supporting the datasets, especially in user and item information with a 94% and 96% of *Completeness* respectively. The obtained 75% of *Completeness* for the context information, even when is a good value, is lower than the others. We attribute this to the fact that recommendation system field just started to considered contextual information [1], and there is not an accepted standard of what context information works best for CARS. Also, the amount of information of the environment (context) that surrounds users is enormous and trying to create a contextual model that encompasses everything is practically impossible. Nevertheless, we strongly believe that GUMCARS has the needed base to allow model designers and researchers to increase the model to their specific contextual needs.

In general, this experiment allowed us to evaluate to some degree the *Completeness* of the model by mapping into it different datasets with different characteristic obtaining encouraging results. The fact that the dataset contains information from different domains, allow us to probe GUMCARS *Generality*, as it was able to support most of the aspect of the tested domains. As for the not supported elements, they were included after finishing the evaluations, and the increased version of GUMCARS will be published as open-source.

3.5 Validation of GUMCARS Structural Correctness

Assessing the quality of a complex model is generally a difficult task [11], since GUMCARS is, to the best of our knowledge, the first user model designed specially to build context-aware recommender systems, a direct comparison with other proposals is difficult or even misleading.

As the goal of GUMCARS is to be used in CARS implementations to support their data, we considered important to assess some quality attributes of the *architectural* and *implemented* representation of the model. Therefore, in this experiment, we focus on evaluating the structural

Correctness of GUMCARS, which is an important quality attribute of conceptual models [118].

The *Correctness* of a model is defined as whether the model conforms to the rules of data modeling techniques [119]. Because GUMCARS was created following the object-oriented design technique, in this experiment we used software quality metrics to assess the level of *Correctness* of the proposed model, which (according to [128]) allow software designers to evaluate quality characteristics of OO models, objectively.

3.5.1 Materials

For this evaluation, the *Depth of Inheritance Tree*, *Number of Children* and *Coupling Between Objects* metrics from the (CK) metric suite [129] were selected, as these metrics can be applied either to class diagram or to code [128]. We also considered important to assess the Maintainability of GUMCARS, as it expresses the easiness with which the model could be adapted to suit specific needs [130], therefore we also include the *Maintainability Index* metric in this evaluation. Next, the used metrics are briefly described.

- **Depth of Inheritance Tree (DIT)**: The depth of a class within the inheritance is defined as the maximum length from the class node to the root of the class hierarchy tree and is measured by the number of ancestor classes [129].

  A low value of DIT implies less complexity in the model organization, but also reuse little information from its ancestors, and reflect a shallow conceptualization of the domain. A high number of DIT implies higher complexity and will make the model hard to understand, use or modify. There is no currently accepted standard for DIT, in this experiment we follow [131, 132] that suggest $1 \leq DIT \leq 6$.
- **Number Of Children (NOC)** Number of immediate sub-classes subordinated to a class in the class hierarchy [129]. The value of NOC represents the number of classes that inherit from a specific class.

  A high value of NOC implies a greater likelihood of improper abstraction of the parent class. The range of NOC value found in nowadays model varies from 0 (being the most common) up to 100 [133]. An optimal value for NOC is between 0 and 11, values from 12 to 28 are regular and greater than 29 as bad NOC [134].
- **Coupling Between Objects (CBO)** Coupling Between Objects, also known as Class Coupling measure the relationships between entities [135]. CBO is a measure of how many classes a single class uses.

A high CBO value implies a class too dependent on other classes which make harder to modify and more prone to failure if one of the depended class fails, therefore a low value of CBO is preferred [129]. The CBO value threshold proposed by Microsoft [136] is a 0-9 range for optimal (green) CBO value, a 10-80 acceptable (yellow) value, and CBO greater than 80 is considered critical (red) value.

- **Maintainability Index (MI)** ISO/IEC 9126 [137] defines maintainability as *"the capability of the software product to be modified. Modification may include corrections, improvements or adaptation of the software to changes in the environment"*. To measure the *maintainability* of the GUMCARS proposal we used the *Maintainability Index* (MI) metric.

  MI metric [138] is one of the most cited models for measure maintainability [139]. MI measures maintainability by taking the size, complexity, and self-descriptiveness of the classes into account. A low value of MI means a hard to maintain model; therefore, a high MI value will make the model easily to adapt by increasing or groom the classes. We used the Microsoft [140] version of the MI, which is a 0 to 100 bounded version of the original metric that ranged from 171 to an unbounded negative number, which can be hard to interpret.

3.5.2 Method

The selected metrics could be applied either to UML class diagram performing the calculation by hand or to classes in code using tools to perform the calculations.

To avoid any bias in the metric applications, and to get the most objective results, we opted to apply the metrics to the code representation of GUMCARS (described in 3.4), using automated tools to perform the calculations.

The GUMCARS's values for *DIT* and *NOC* metric were gathered using NDepend, for the *CBO* and *MI* metrics, Visual Studio IDE was used. Both tools gave a numeric value of each class for the evaluating metric. The results of each metric are loaded into SPSS for the data analysis.

3.5.3 Results

According to [129], a lower DIT value is preferred, with a threshold between 1 and 6 for an optimal value. This experiment revealed a maximum DIT value of 4, a mean value 1.41 with a Standard Deviation (SD) of 0.69 as can be seen in Table 3.9, placing GUMCARS inside the optimal threshold for this metric. The deepest leaf found in GUMCARS inheritance tree is *Hardware* >*Device* >*MobileDevice* >*SmartPhone* in the context package.

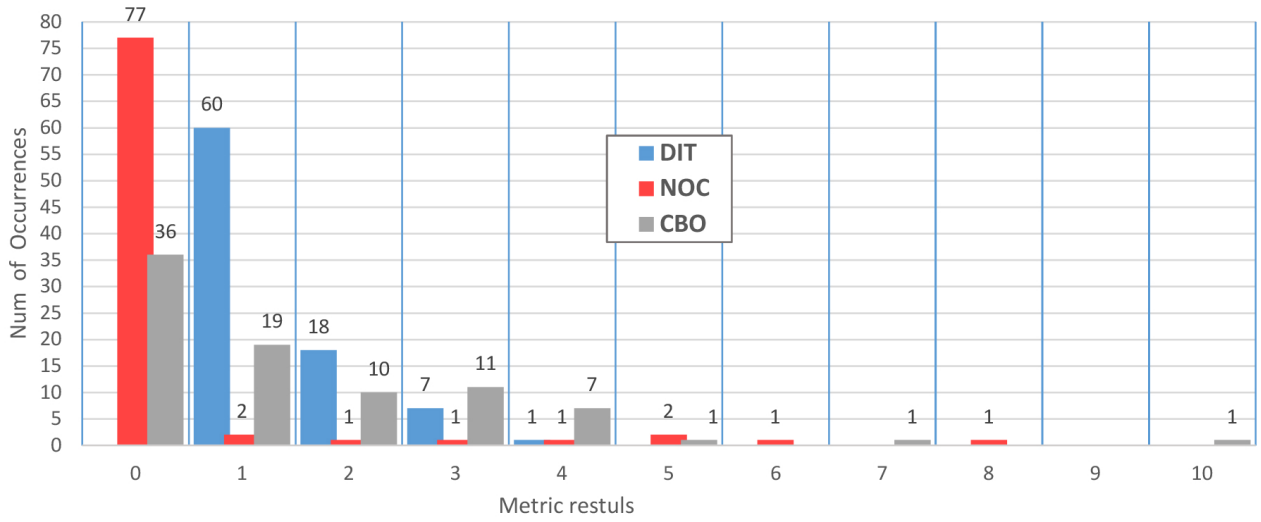

Figure 3.11: Frequency distribution of DIT, NOC, and CBO metrics for GUMCARS.

The NOC mean value of GUMCARS is 0.41 with an SD of 1.40 and a maximum number of 8 children for a single class, that correspond to *Item* class, as all the types of items considered in GUMCARS are dependent of *Item* class.

In this experiment, we got a CBO value of 1.42, with SD of 1.79 and a maximum value of 10. The maximum CBO value corresponds to *User* class, as it is dependent of 10 other classes, even when the 10 is outside the optimal value (as shown in Figure 3.11), it just occurred once, and the rest of the 85 classes fall between 1 and 7.

The last metric applied in this experiment is MI, the mean value obtained is 94.8, with SD of 2.89 and the lowest value of 90 with only 1 occurrence, and correspond to *Item* class.

Next, Figure 3.11 shows the frequency distribution for the DIT, NOC and CBO metrics, and Figure 3.12 shows the distribution for MI values. Table 3.9 summarize the results obtained from the application the quality metrics to GUMCARS. Then, 3.10 shows some representative examples of the raw data obtained from the metrics, showing some best and worst cases.

Table 3.9: Results of applying quality metrics to GUMCARS.

Metric	Min	Max	Mean	SD	Threshold	Better when	Tool
DIT	1	4	1.41	0.69	$1 \leq DIT \leq 6$	Lower	ND
NOC	0	8	0.41	1.40	$0 \leq NOC \leq 29$	Lower	ND
CBO	1	10	1.42	1.79	$0 \leq CBO \leq 80$	Lower	VS
MI	90	100	94.83	2.89	$0 \leq MI \leq 100$	Greater	VS

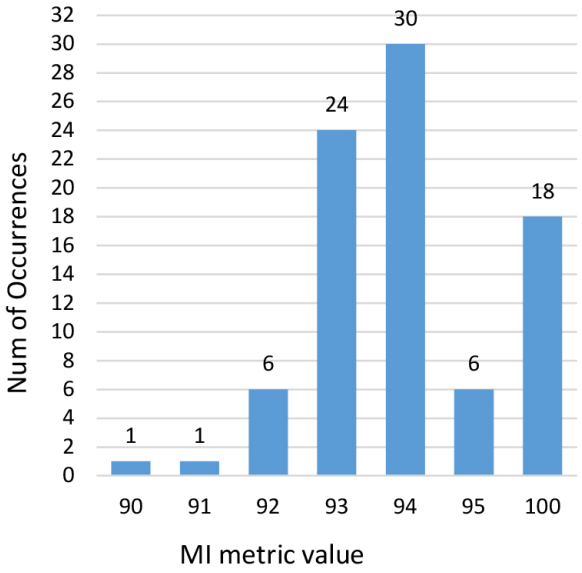

Figure 3.12: Frequency distribution of MI values for GUMCARS.

Table 3.10: Representative examples of raw data obtained from the application of the metrics

Class	Location in Model	DIT	NOC	CBO	MI
User	*(top level)*	1	0	10	90
Context	*(top level)*	1	0	7	92
Item	*(top level)*	2	8	0	91
Activity	*(top level)*	1	6	2	93
RelationType	Context.Social	1	0	0	100
SmarthPhone	Context.Computing	4	0	4	95
MentralStress	User.Mental	1	0	0	100

3.5.4 Discussion

The low value obtained for DIT (1.41) reflects that the model organization is not complex, which will help GUMCARS to be understood and used by its target users (CARS designers, researchers, and developers). This DIT value supports *Correctness* with respect to the organization of the model classes.

For the NOC metric, we obtained a 0.41 with a maximum of 8, this place GUMCARS in the optimal values part of the threshold. The result of NOC strongly suggests that the proposed model uses a correct level of abstraction, as there is not a single class with too many dependent children.

Coupling Between Objects (CBO) metric yield a mean 1.42 value which reflects that GUMCARS has an optimal level of coupling between classes. From the 86 total classes of the model, only one lays outside the optimal value, namely the *User* class, which got a CBO of 10, placing it in the *good* part of the threshold. Certainly, the *User* CBO could be improved sub-dividing the class, but we need to consider the increase of complexity by adding another level on the inheritance tree, thus increasing the overall DIT value. Overall, the result of this metric support the general *Correctness* of the proposal, showing that the model represents the intended domain using a correct abstraction level.

This experiment yielded a mean value of 94.83 for the MI metric, meaning that the proposal's correct organization of classes and attributes resulted in an optimal level of MI. Figure 3.12 shows that most of the classes have a MI of 93 and 94. The lowest value of MI found in the whole model was 90, which correspond to *Item* class, as all the items that the model consider are dependent on this class. This means that any change made to the *Item* class will be propagated through other classes, and that designers need to consider this before making any change to *Item* class. Nevertheless, a MI value of 90 falls very close to an optimal value, and we consider that it is a good value considering that *Item* class is some of the cornerstones of the GUMCARS.

Overall, in this experiment we evaluated the *computational* representation of the proposal, obtaining very good results for all the quality metrics, which strongly suggest that GUMCARS contains a high level of structural *Correctness*, and can be used by CARS designers and developers to support the data needed for their systems.

3.6 Conclusions and Summary

This chapter described the creation of GUMCARS, a General User Model for Context-Aware Recommender Systems. GUMCARS is a proposal to address the problem of lack of data structures that support the management of information required by recommendation algorithms. To address this problem, GUMCARS provides designers and developers a comprehensive set of classes and attributes that together include the information of user, context, and items information that literature has used as input information for recommendation algorithms. GUMCARS is presented from a conceptual view by a proposed taxonomy of CARS information, and from architectural view by describing the model using the object-oriented paradigm and UML class diagrams.

Two validation where performed to GUMCARS. First, the semantic completeness and generality of the model was assessed, the results obtained suggest that the model is capable of supporting most of the information contained in CARS datasets, and that the model is capable of supporting

information from different recommendation domains. The second validation was an assessment of the structural correctness of the proposal using software quality metrics, and the obtained results suggest that the model elements are structured correctly

The main limitation of GUMCARS proposal it that, as the model is based on existing CARS literature, it may no include new trends of information fed into recommendation algorithms, but as we gave special importance in making the model generic, even when some very specific information aspect of user or context is no considered in the model, the model can be adapted to fit such specific needs. Another possible limitation of GUMCARS if the complexity of the class landscape, as some designer and developers may argue that the model contain too many classes to remember. In next chapter, an extensive documentation of the model classes will be present to alleviate this limitation.

Summary

Within this chapter, the thesis objectives **Obj1** (*Identify the user, context, and items information aspect that has been used in context-aware recommender systems literature*) and **Obj2** (*Design a user model for context aware recommender systems that structure the user, context, and items information*) were achieved, and their respective goals were met.

Section 1 Presents a proposed taxonomy of CARS information, which directly address **Goal 1** (*Design a taxonomy if CARS information*).

Section 2 Describes the systematic literature review performed which attends **Obj1** and **Goal 2** (*Perform a literature review based on Kitchenman methodology ...*).

Section 3 Describes the creation of the GUMCARS user model which attends **Obj2** and **Goal 3** (*Create a data structure for CARS information*).

Section 4 Validates GUMCARS completeness and generality, and its results addressed **Goal 4** (... *support information from different recommendation domains*).

Section 5 Validates GUMCARS structural correctness, which attends **Goal 5** (... *comply with object-oriented quality rules*).

CHAPTER 4

USER MODELING FRAMEWORK FOR RECOMMENDER SYSTEMS

In previous section, we presented GUMCARS, a proposal of a user model designed to support the user, context and item's information used by CARS, and address at some extent the problem of lack of structures for CARS information by proposing a reference model that can be used by designers, developers, and researchers to manage the CARS data.

With the User Modeling framework for context-aware Recommender Systems (UM4RS) the problem of lack of tools that support developers in the implementation phase of CARS, described in the introductory chapter of this thesis work, is addressed. UM4RS takes the model conceptualization into a working tool intended to ease the data management needed by CARS.

UM4RS (pronounced *umars*) is a framework intended to support designers and developers of CARS facilitating the creation of context-aware recommender systems, as UM4RS contains the following features:

1. UM4RS was created following the Object Oriented (OO) Paradigm [141] which allow developers to create a direct map between real-world elements and model elements, and with a few lines of code, developers can set up an object landscape for managing user, context, and item information.

2. UM4RS' core is an implementation of the GUMCARS model that provides the framework a

structure for the User, Context and Items attributes needed to support the CARS information.

3. UM4RS implements a mechanism to automatically data persistence for all the model base classes and can be easily extended to any custom classes added later on.

4. UM4RS contains data exportation mechanisms that allow developers to export the model information a dataset file in different forms, which ensures the interoperability of the framework with third-party algorithms.

5. UM4RS functionality is provided through an easy to use Application Programming Interface (API) that allow developer to create objects, access its attributes and store them in a database solution.

6. UM4RS provide designer and developers a web-based framework documentation that eases the process of learning to use the framework.

The result is a user modeling framework for context-aware recommender system that aims to achieve balance between ready to use functionality that increases developers efficiency, maintainability so the framework can be extended to suit specific project needs and usability so developers can easily get along with the framework. UM4RS provides developers an extensive set of functionality that used through the frameworks API as shown in Figure 4.1, and depicts the overall architecture of UM4RS 4.1.

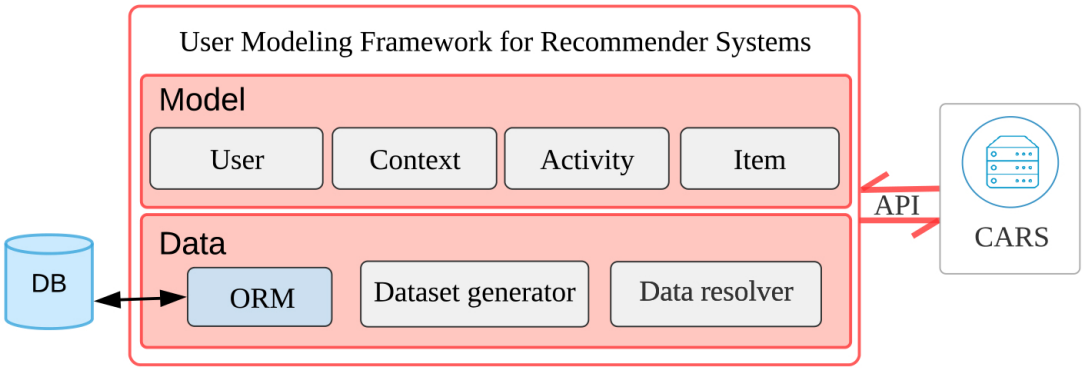

Figure 4.1: Overall view of UM4RS architecture.

The following two sections of this chapter describe each of modules, followed by Section 4.4 where a structural validation of the framework is performed. Then, Section 4.5 presents the evaluation of UM4RS usability. Finally, Section 4.6 presents a use case where UM4RS is used inside a recommendation framework.

4.1 User Model Module

As one of UM4RS' main modules is a data model, UM44RS uses an implementation of GUMCARS model that allows the framework to have a defined structure into which organize the data required by CARS. GUMCARS provides a static architecture of packages, classes, an attributes created under OO paradigm. As GUMCARS is presented in Chapter 3 as a conceptual data model, in order for UM4RS to use its classes, and implementation of the models needed to be performed.

The implementation of GUMCARS and UM4RS were performed using C# language and Microsoft .Net Framework 4.5. Even when technical details about both implementations are given through this chapter, neither the model, nor the framework is tight to any specific technology, and implementation of them can be performed in any other language, platform, or database.

As GUMCARS proposal provides with UML diagram of the classes, the implementation of the model was performed automatically using Visual Paradigm[1] code generation tool. This converted the UML representation of the model into a code library that respected the relations between classes, the package organization, and the composition of classes. We consider this GUMCARS code library, as an exact representation of the conceptual model, and called this library an *implemented view* of the model.

Next, Listing 4.1 shows the resulting code for the *User* class as an example of the code obtained for the implementation of GUMCARS.

Listing 4.1: Implementation of GUMCARS User class

```
namespace GUMCARS.User
{
    public class User
    {
        public int Id { get; set; }
        public Demographic Demographic { get; set; }
        public Contact.Contact Contact { get; set; }
        public Contact.Account Account { get; set; }
        public ICollection<Role.Role> Roles { get; set; }
        public ICollection<EmotionalState> Emotions { get; set; }
        public ICollection<PersonalityAttribute> Personality { get;set; }
        public Physiology Physiology { get; set; }
        public Mental.Mental Mental { get; set; }
```

[1] https://www.visual-paradigm.com/

```
        public InterestPreference InterestPreference { get; set; }
        public ICollection<SocialNetwork> SocialNetworks { get; set; }
    }
}
```

4.2 Data Module

The data module of UM4RS is in charge of managing the database creation and connections, store the information of the GUMCARS objects' attributes into database tables and convert database records into objects of GUMCARS classes. In addition, this module formats and serve the model data so it can be used by the recommendation algorithms. This module consist of an Object-Relational Mapping *ORM*, *Dataset generator* and *Data resolver* components.

4.2.1 Object-Relational Mapping

For the Object-Relational Mapping UM4RS uses ADO.Net Entity Framework [142] and a *Code First* approach [143], which means that developers implement the classes and relations, then the ORM creates and maintains the Database schema according to mapped classes. Persistable classes are *annotated* with a `[Table("name")]` attribute which specifies the name that the generated table will have, if not present, the class name will be used as table name.

In order to provide current and future classes with CRUD functionality, UM4RS uses the technique of *Generics* classes and methods, along the following classes to provide basic database management.

- **IBaseEntity**: Base interface for any class with database mapping capabilities, this ensures that all the class that will be mapped to database tables have an *Id* property that is used as a *Unique Key* in the database table.
- **ModelEntity**: Abstract superclass for any class with database mapping capabilities, this class contains the mentioned CRUD methods, and inherits CRUD operations to any child, e.g. *Find*, *FindAll*, *Save*, *SaveAsync*, *Delete*, *DeleteAsync*, *Cout*, *CoutAsync*, etc. Figure 4.2 shows the list of CRUD operations and some examples of usage of such operations.

  This class allows UM4RS users to create any new class, make it extend *ModelEntity* and implements *IBaseEntity*, and such class will have all the CRUD operations, no further modification needed apart from adding such class to *ModelORM* class so the system can create a database table to store its data.

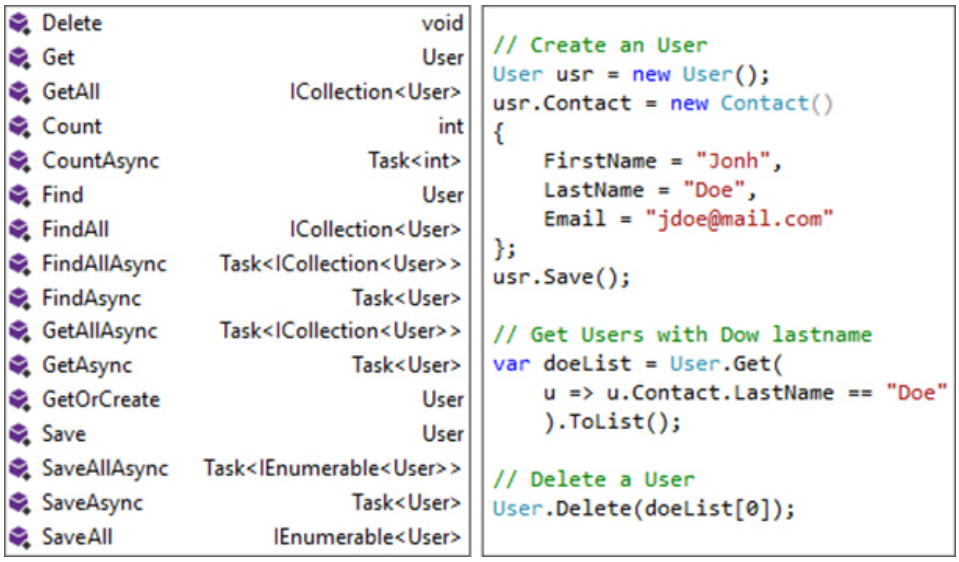

Figure 4.2: Basic operations for each class that inherits from *ModelEntity* superclass.

- **ModelORM**: Class responsible for the ORM management, this class registers the classes that will be persisted to database. Currently, UM4RS uses Microsoft SQL Server as the database management system, but if developers want to use another database manager (as MySql for example), they just need to pass a corresponding *connection string* to this class and add a reference to such database manager API.

  If new classes are added to UM4RS model, such class need to be registered in *ModelORM* class so it can be mapped to a corresponding database table.

4.2.2 Dataset Generator

The second main functionality of the Data module is to provide the stored data so it can be used by recommendation algorithms, for this, a *Dataset generator* functionality was built into this module.

As most of the contextual information is in shape of categorical data [124] (as shown in Table 4.1), but most of the algorithms work with binary data matrix (example shown in Table 4.2), this module allows developers to export the information contained in the model either in categorical (comma separated values) or in a binarized format, so developers can be fed the exported dataset directly into recommendation algorithms without the need to change anything.

To export the information from database to a dataset file, the UM4RS framework uses class and attributes *annotation* to specify what information should be passed into the dataset, based on the following attributes:

Table 4.1: Categorical dataset example

UserId	ItemId	Rating	Weather
21	33	5	Sunny
21	33	4	Rainy
22	45	4	Sunny
22	45	2	Rainy

Table 4.2: Binary format dataset example

User	Item	Rating	Weather:Sunny	Weather:Rainy
21	33	5	1	0
21	33	4	0	1
22	45	4	1	0
22	45	2	0	1

- *ExportableClass*: Attribute that indicates the *DatasetGenerator* to include all the class attributes in the exportation process unless specifically stated in any attribute.
- *Exportable*: Used to annotate class attributes, indicates that such attribute data will be included in the generated dataset. Additional information can be passed to this annotation, for example, the *Name* of the dataset column and the *GlobalOrder* in which the column will appear in the dataset.
- *NonExportable*: Can be used over classes or attributes, and indicates that such elements will not be included in the dataset.

Next, Listing 4.2 shows the *Context* class using the class and attributes annotations to specify that the class is exportable, that the *id* attribute should appear as second column in the resulting dataset with a header name *contextId*, and that *PhysicalLocation* attribute should not be exported.

Listing 4.2: Implementation of GUMCARS User class

```
namespace UM4RS.Context {
  [Table("Context")]
  [ExportableClass]
  public class Context : ModelEntity<Context>, IBaseEntity
  {
     [Exportable(GlobalOrder=2, Name="contextId")]
     public int Id { get; set; }
```

```
        [NonExportable]
        public PhysicalLocation PhysicalLocation { get; set; }
        public Time Time { get; set; }
        public Computing Computing { get; set; }
        public PhysicalCondition PhysicalCondition { get; set; }
        public ICollection<Resource> Resources { get; set; }
        public ICollection<SocialRelation> SocialRelation {get; set;}
        public Place Place { get; set; }
    }
}
```

4.2.3 Data resolver

This module is based on Hybreed [11] resolvers, the idea is to specify actions to perform when some model data changes or some time interval have occurred, such actions are used to derive or retrieve additional information for the model. For example, a weather resolver can be defined to obtain the weather conditions of the user when his latitude or longitude changes.

Here, developers could use basic rules to specify resolvers, or use complex AI and data mining techniques in order to infer additional information (from social media for example), without disturbing the user with questions. This module is theoretical, not yet implemented in the current iteration of the framework.

4.3 UM4RS Documentation

The functionality of UM4RS can be accessed by developers through the provided API, which allows them to create objects of the contained classes and execute their methods.

APIs, like other sophisticated technical tools, must be properly learned and correctly operated to be effective. To achieve this goal, APIs typically come with some kind of documentation [144] that developers can use to familiarize with the API functionality. API documentation contains information presented as text, code, images, etc. presented in a document or most commonly in web page format [145]. The objective of a documentation is to clarify the assumptions and constraints of the API, to serve as guideline for developers when interacting with the API, and to help developers to pitfalls [146].

// TODO: Quitaer aqui UM4RS provide developers with an online documentation[2]. The documen-

[2]http://UM4RS.com

tation gives developers an introduction to the goals and motivation of UM4RS. Following [147, 148], the documentation also contains a getting started section that shows developers how to configure the framework, as well a sample code section that present developers some examples of how to use the framework to store and retrieve information to and from the database, and how to export that information into a dataset file. Figure 4.3 shows the welcome page of UM4RS documentation.

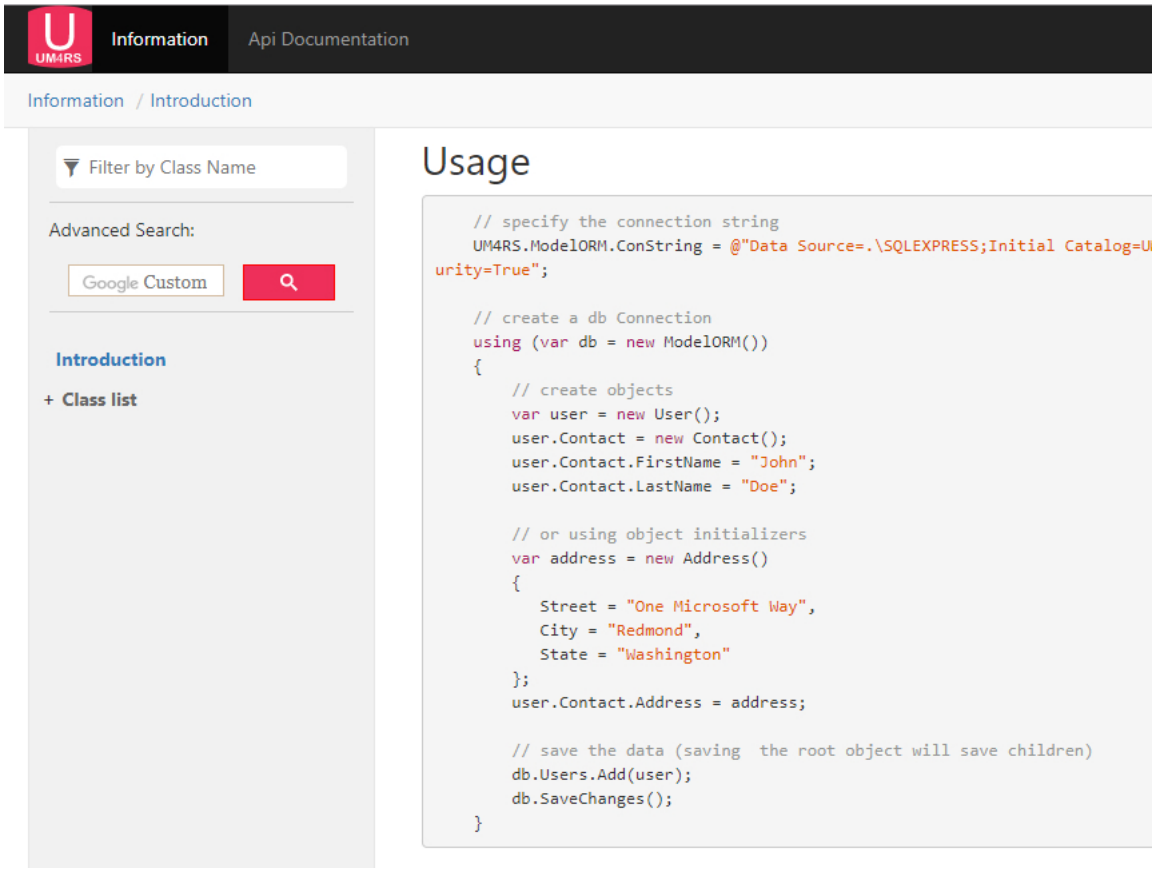

Figure 4.3: Welcome page of UM4RS API documentation.

Following the documentation creation guidelines of [149, 150], UM4RS documentation also have detailed descriptions about each *Class*, *Interface*, and *Method* included in the API, as well as, the *Parameters* of each method. This detailed documentation also can serve as GUMCARS documentation, as most of the classes contained in UM4RS correspond to the GUMCARS model. This detailed documentation is not intended to be fully read by UM4RS users, rather is there so developers can refer to it when going deep in the usage of the API, and need to gather specific details about some class.

4.4 Validation of UM4RS Structural Correctness

As UM4RS is a modeling framework whose end users are developers, we consider important to validate assess the architectural correctness quality attribute. This validation is similar to the one performed over GUMCARS in Section 3.5 of this document. As part of the UM4RS framework is GUMCARS model, we consider interesting to perform this validation, as obtained results will reflect if the structural correctness of GUMCARS kept similar, or if the model quality degrades when including it in a wider project. Also, it is interesting to see if design decisions taken by the model affect the modeling framework.

Next, the materials used for this evaluation is presented, followed by the obtained results. Then this section discuss the obtained results, and compare them with the results for obtained when applied the same metrics to GUMCARS model.

4.4.1 Materials

For this validations, as for the GUMCARS validations, the Chidamber & Clark (CK) metric suite for object oriented software is selected, these metrics reflect various quality aspects of an object-oriented software, and can be applied automatically to the code, which avoids biases in the measurements. These metrics are *Depth of Inheritance Tree*, *Number of Children*, *Coupling Between Objects*, which were also used to validate GUMCARS architecture. Like in GUMCARS, *Maintainability Index* [138] is also used in UM4RS to assess the capability of the framework to support changes and adaptation in its functionality.

As UM4RS is a tool with implemented functionality, rather than an implementation of a conceptual model, some extra metrics were applied to UM4RS that could not be applied to GUMCARS, as these metrics evaluate functionality characteristics of software. Such metric are *Weighted Methods per Class*, *Response For a Class*, and *Lack of Cohesion Method* also proposed by in the CK metric suit. Another metric applied to UM4RS is *Cyclomatic Complexity* [151] as it was stated important to apply to any object-oriented system by [152, 153].

Next, the applied metrics are described.

- **Depth of Inheritance Tree (DIT)**: The depth of a class within the inheritance is defined as the maximum length from the class node to the root of the class hierarchy tree and is measured by the number of ancestor classes [129].

  A high number of DIT implies higher complexity and will make the model hard to understand, use or modify; therefore, a low value is preferred. This work follows [131, 132] that suggest a

threshold of $1 \leq DIT \leq 6$.

- **Number Of Children (NOC)** Number of immediate sub-classes subordinated to a class in the class hierarchy [129]. A high value of NOC implies a greater likelihood of improper abstraction of the parent class. An optimal value for NOC is between 0 and 11, values from 12 to 28 are regular and greater than 29 as bad NOC [134].
- **Coupling Between Objects (CBO)** CBO is a measure of how many classes a single class uses [135]. A high CBO value implies a class too dependent on other classes which make harder to modify and more prone to failure, therefore a low value of CBO is preferred [129]. The CBO value threshold proposed by Microsoft [136] is a 0-9 range for optimal (green) CBO value, a 10-80 acceptable (yellow) value, and CBO greater than 80 is considered critical (red) value.
- **Maintainability Index (MI)** MI measure the capability of the software to support modifications [139] by taking the size, complexity, and self-descriptiveness of the classes into account. We used the Microsoft [140] version of the MI, which is a 0 to 100 bounded version of MI original metric.
- **Weighted Methods per Class (WMC)** Weighted Method per Class is the CK metric to measure the complexity of a class, and it's strongly related to McCabe CC, since WMC is the sum of the CC of all the methods in a class [129]. Therefore the almost the same rules of CC values apply to WMC, a low value of WMC is preferred. A value of WMC lower than 12 is optimal (green), WMC between 12 and 34 is regular (yellow) and higher than 34 is not desirable (red) [134].
- **Response for a Class (RFC)** Is the number of sets of all methods that can be invoked in response to a message to an object of the class, or by some method in the class [129]. This metric evaluates the complexity of class through the number of methods and communication with other classes. The larger the number of methods that can be invoked form a class through an object, the greater the complexity of a class, and a bigger number of test and debugging needed [154], therefore a low value for $0 \leq RFC \leq 10$ is preferred.
- **Lack of Cohesion Method (LCOM)** In Object Oriented systems, cohesion is defined as "*the degree to which the methods and attributes of a class belong together*" [155]. CK [129] defined a metric to measure the Cohesion of Object Oriented software called Lack of Cohesion Metric (LCOM), this metric is based on the notion that if all attributes of a class are used by the methods of the same class, the elements of the class belong together, therefore the class is considered highly cohesive. For the LCOM metric, the range of possible values are $0 \leq LCOM \leq 1$, a high LCOM value generally pinpoints a poorly cohesive class, therefore a low LCOM value is preferred.

- **Cyclomatic Complexity (CC)** The McCabe Cyclomatic Complexity [151] is used to evaluate the complexity of an algorithm in a method [130] by counting the number of independent paths through a program unit, that is, the number of decision statements plus one. The CC metric provides a single ordinal number that represents the complexity of the method or the member to which was applied [156]. A high value of CC implies a complex piece of software, therefore a low value of CC is preferred. According to [157] value between 0 and 10 represents an optimal (green) vale, 11-20 is acceptable (yellow), a cc of 21-50 represents high risk (orange), and a cc greater than 50 is considered unstable system (red).

4.4.2 Results

To apply the described metrics, we used NDepend (ND) [132] code metric, and Visual Studio (VS) 2015 ultimate, which allow us to automatically collect the values for each metric, and avoiding possible biases of performing the counting and calculation manually.

The total number of classes contained in UM4RS is $n = 90$. Next, Table 4.3 show the results of UM4RS for each of the applied metrics, and then each result is briefly described.

Table 4.3: Results of applying software quality metrics to UM4RS framework.

Metric	Min	Max	Mean	SD	Threshold	Better when	Tool
DIT	0	5	2.17	0.90	$1 \leq DIT \leq 6$	Lower	ND
NOC	0	62	1.07	6.64	$0 \leq NOC \leq 29$	Lower	ND
CBO	0	30	4.47	4.68	$0 \leq CBO \leq 80$	Lower	VS
MI	75	100	94.59	3.52	$0 \leq MI \leq 100$	Greater	VS
WMC	0	19	0.30	2.03	$0 \leq WMC \leq 34$	Lower	ND
LCOM	0	0	0	0	$0 \leq LCOM \leq 1$	Lower	ND
RFC	0	9	1.33	1.28	$0 \leq LCOM \leq 10$	Lower	ND
CC	0	35	8.02	6.98	$0 \leq CC \leq 50$	Lower	VS

The obtained mean value for DIT metric is a 2.17, which places UM4RS in an optimal value in the DIT threshold proposed by [132], even the maximum value of 5 obtained for a single class falls in the optimal range. The deepest leaf found in UM4RS inheritance tree corresponds to *SmartPhone* class contained in the *Context* package. The 91% of the classes have a DIT lower than 4. The distribution of DIT values is shown in Figure 4.4.

The NOC mean value of 1.07 place UM4RS well into the optimal threshold for this metric, even when the max value of 62 obtained for *ModelEntity* class its outside the 29 value considered optimal [134], the 98.88% of the classes falls in the optimal part of the threshold, as shown in Figure 4.5.

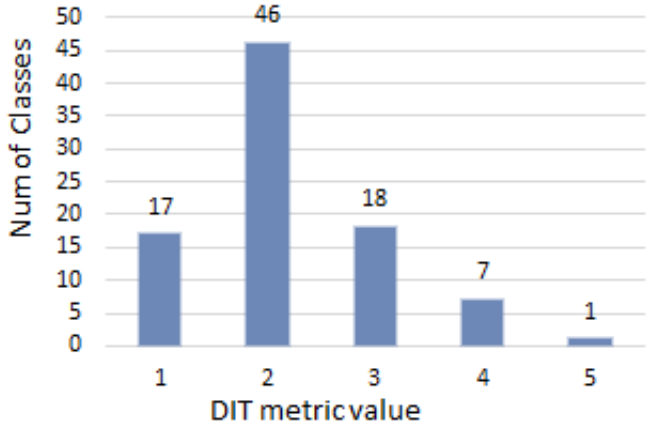

Figure 4.4: Frequency distribution of DIT values for UM4RS.

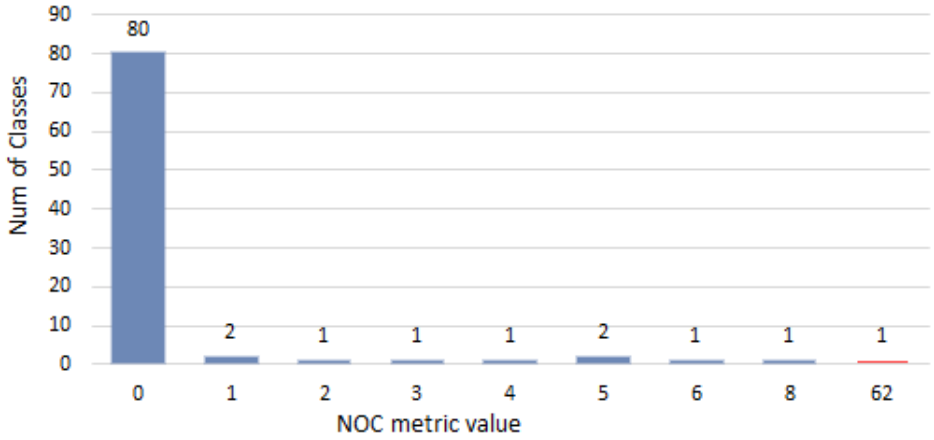

Figure 4.5: Frequency distribution of NOC values for UM4RS.

The mean CBO value of 4.47 with an SD of 4.68 falls into the optimal part of the threshold, an even the maximum value obtained of 30 is an acceptable (yellow) value according to [136], such value corresponds to *ModelEntity* class. The CBO distribution is shown in Figure 4.6, where can be seen that 91% of the classes contains an optimal value, and the rest 9% corresponds to acceptable values in the threshold.

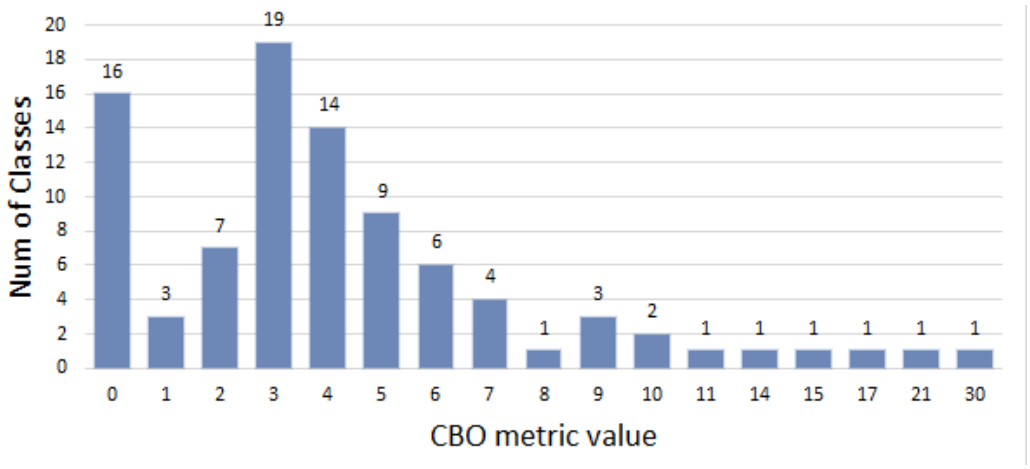

Figure 4.6: Frequency distribution of CBO values for UM4RS.

For MI the higher the value the better [140], the obtained distribution for MI is shown in Figure 4.7, where can be seen that the lower value of UM4RS is 71 with only one occurrence, and the mean value is a 94.59 with an SD of 3.52.

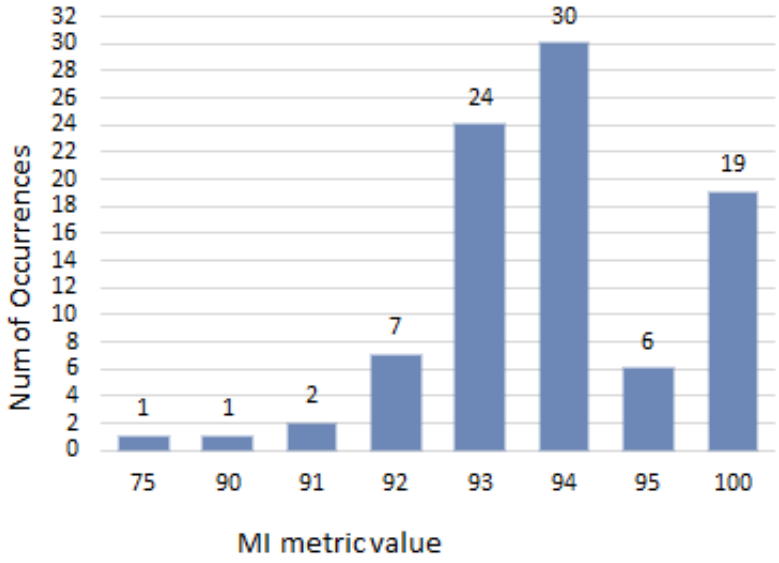

Figure 4.7: Frequency distribution of MI values for UM4RS (n=90).

As expected, the values obtained for WMC, LCOM, and RFC fall in the optimal part of the corresponding threshold of each metric. The WMC contains an outlier value of 19 that corresponds to *ModelEntity* class. LCOM shows zeros for all the classes suggesting that all the elements of each class correspond to such class and no attribute or relation of misplaced. RFC mean value of 1.33 fall into the optimal part of the threshold, with an outlier value of 9 that still falls into the optimal

threshold, and corresponds to *Video* class of the *Item* package.

Finally, CC metric showed an 8.02 mean value with an SD of 6.98. This value is well in the optimal part of the threshold. This metric showed a maximum value of 35 with one occurrence, while the 95% of the classes obtained a value between 0 and 20, as shown in Figure 4.8.

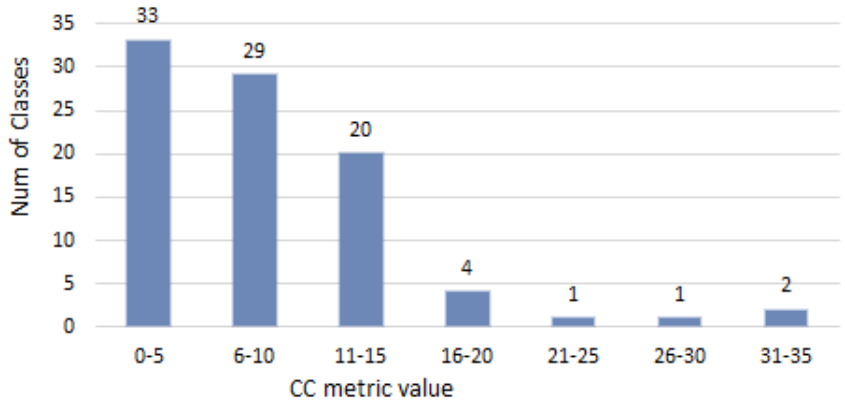

Figure 4.8: Frequency distribution of MI values for UM4RS (n=90).

4.4.3 Discussion

As the framework added the *ModelEntity* as the superclass for all the model classes, the DIT value increased by one in all the classes, nonetheless, no class overpass the threshold of optimal values for this metric. This decision of inherit all the classes from a single point (*ModelEntity*) also affect the CBO and NOC values, going from a mean 0.41 to a mean 1.07 for the NOC metric, which does not represent any problem when evaluating mean values.

As *ModelEntity* is the parent of all other classes, it got a NOC value 62 that is well outside the recommended threshold. However, having this superclass with all the methods for storing, searching, and retrieving information to and from the database, avoid repeating such methods in all the other 86 classes, which dramatically affects the WMC, LCOM, RFC, and CC in a positive way. The values for this metrics will be much higher if *ModelEntity* class do not exist, and each class implemented its own methods for database storing its information. Having this in consideration, we believe that the downside of having one class with too many children is counterbalanced by the improvements it brings to the last four metrics applied.

The results obtained from this evaluation, are very similar to the obtained from the same evaluation performed to GUMCARS, showing that the model structure was not altered during the development of the framework, and that GUMCARS structure support extensibility. Overall, this

evaluation showed optimal mean values for all the applied metrics, which strongly suggest that UM4RS contains a high level of structural *Correctness* and that the framework has a good level of maintainability which reflects its capability to be adapted by developers to suit specific project need with minimum possibilities of affecting the overall structure of the framework.

4.5 Evaluation of UM4RS Usability & Impact on Productivity

According to [158], one of the most common measurements in software engineering used to measure the impact of a new process or tool in the development process is the programmers productivity. The term productivity is defined by [159] as the ratio of outputs produced to the resources consumed. Even when this is a broad definition of the term, the ratio between the input and the output still applies.

As the main objective of this thesis work is to improve the productivity of CARS developers, it is essential to measure the impact of UM4RS when used as part of the CARS development process. In [160], the productivity of software developers is defined as the ratio of *number of functions points to the effort in man required*. Base on this definition is safe to say that the lower the effort, the higher the productivity in function points. In this work, we indirectly measure productivity by measuring the effort required to implement a defined set of function points.

To be able to distinguish if the UM4RS impact in developers productivity is due to the model functionality, or is because developers could not understand how to use it, we consider important to also assess the usability of the framework, as usability has an important impact of developer' ability to complete the tasks [161].

Usability is the degree to which a system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use [162]. Mapping the usability definition to UM4RS, the software product corresponds to UM4RS API, the users are developers, and the goal is to manage the CARS data.

This section describes an experiment performed to gather empirical evidence of UM4RS usability level as perceived by developers, to identify the impact of the framework on developers productivity when using the framework to perform some CARS data management task, and to compare the lines of code required to implement the same task with and without using the framework.

4.5.1 Participants and Environment

The participants of this study included 24 developers from Tijuana, México. Participants age ranged from 20 to 31 years old. All participants had at least basic C# programming skills and were familiar

using Visual Studio and MS SQL Server. None of them had previous interactions with the proposal. We had 13 participants from 2 different software developer companies (hereafter referred to as industry participants), the rest (11) were students at the Autonomous University of Baja California in Tijuana México from Informatics and Computer engineering degrees, hereafter referred to as students.

The experiment was performed in 4 sessions, in different locations. Two sessions were conducted in the software companies' offices, where participant used laptops running Windows 10, Sql Server 2012 R2 and Visual Studio 2015. Two more sessions were performed in the university, where participants used customary PC running Windows 7, Visual Studio 2012 and SQL Server Express 2012 R2. All participants were instructed to use Google Chrome to browse the UM4RS documentation.

4.5.2 Method and Materials

For the collection of data during this experiment, two main research instruments where used:

Productivity measurement

The productivity model proposed in [160], and formally presented in Equation 4.1, is used to measure developers productivity when implementing a set of development task with and without using UM4RS. Then, the difference between the two represents the increase (or decrease) in developers productivity brought by UM4RS.

$$Productivity = \frac{\text{Number of Function Points}}{\text{Effort in Man}} \tag{4.1}$$

- Function point is a commonly used unit of measurement in software engineering. Function points are defined as the amount of functionality of a system as described by the specification [163]. Based on this definition, in this experiment, we defined a series of development task related to the management of CARS information, and consider each task as a function point.
- Effort estimation in software implementation has many approaches of measurement, but most of them require the subject to have prior experience with the technique [164], or imply mathematical calculations over logged information from various executions [165] to be effective.

  In this experiment, we used *Expectation Measurement* [166] technique, which in reliable and easy to use method of effort estimation [11]. This technique consists of describing a task to participants and ask them to estimate the effort required to implement such task base on

personal experience, ability, and knowledge. Then, participants are asked to implement the described task, and after completion, they are asked to rate the required effort to implement the task. Then the actual effort is calculated as the difference between the estimated and the actual effort required. A 5-point Likert scale was used to collect both, the estimated and the required effort for each task.

Usability measurement

The System Usability Scale (SUS) metric proposed by [167] was used as a way to effectively asses the participants' perception of UM4RS usability. SUS metric consists of ten elements for which participants express their level of agreement using a 5-point Likert scale. Examples of the statements are "I think that I would like to use this system frequently" and "I found the various functions in this system were well integrated". The participant ratings are processed using the SUS provided algorithm, and a result value between 0 and 100 is obtained, that represent the usability of the evaluated system.

Tasks

Participants performed the following tasks using Visual Studio and a reference to UM4RS. Within the test, participants were free to use any programming convention to solve a set of different task, which are typical tasks to manage the information of a video CARS. The tasks are:

1. Design the classes to support the following information:

   For User: *id*, *age*, *gender*, *first name*, *last name*, *email*, *user name* and *password*.

   For Video: *id*, *title*, *list of actors*, *genre*, *release year* and *duration* in minutes.

   For View Action: the *User* that performed the action, the *Video* the user saw, *rating* value in a 1 to 5 scale, *weather condition* as string, *min. temperature* and *max. temperature* from the moment the user saw the video.

2. Create two objects for each class, and store them in database.

3. Get the list of users stored in the system and print their *first name*.

4. Modify the information of the user whose *id* is 1.

5. Delete the video from the database whose id is 2.

4.5.3 Experimental design

At the beginning of the experiment, developers were given an explanation of the tasks they will implement, the technology, and the tools that should be used. Then, a printed document detailing the tasks, and the methods they should follow to implement the task was provided to each participant. The experiment was conducted using the following procedure:

1. A list of 5 tasks was given to study participants.
2. An explanation of each task was given by the moderator, and any question regarding them was answered to ensure there were no doubts about any task.
3. Participants were asked to state the *expected* effort needed to solve the tasks as if they will perform the task without using UM4RS, i.e. designing, and implementing the classes needed to support the presented tasks. Participants used a five-point Likert scale to rate the amount of effort they considered was needed to perform each task.
4. All participants received basic (3 minutes) explanation about UM4RS API philosophy and goal, as well as a guided tour of the documentation.
5. Participants performed the tasks using UM4RS. We observed and noted whether each task was completed single-handedly, with assistance or not at all.
6. After finishing all tasks, each participant was asked to rate the *actual* effort required to implement each task, using the same 5-point Likert scale.
7. Finally, we asked participants to answer the SUS questionnaire to express the perceived usability of UM4RS. The questionnaire was available online using Google Forms platform, and no identification of participants was required to answer it, so developers can express openly in their responses.

To be able to compare measure the lines of code required by developers to implement the tasks, after participants finished the experiment, the wrote code was collected, except for participants of the second company, whose privacy policies do not allow us to have the code.

4.5.4 Results

One participant from the industry and 2 students needed assistance with the usage of UM4RS, 3 students needed assistance with programming, not directly related to UM4RS. Apart from that, all participants were able to solve the assigned tasks.

Industry participants required an average 49 min and 31 secs with a standard deviation (sd) of 13 min 29 secs to complete all the task. Student participants required an average 91 min 43 secs with an sd of 12 min 11 secs.

The average *expected effort* when developer do not use UM4RS was 3.43 with an sd of 0.87, and for the *required effort* when using UM4RS was an average 2.12 with an sd of 0.74. Figure 4.9 presents a graphical comparison of the expected and required effort values for each participant.

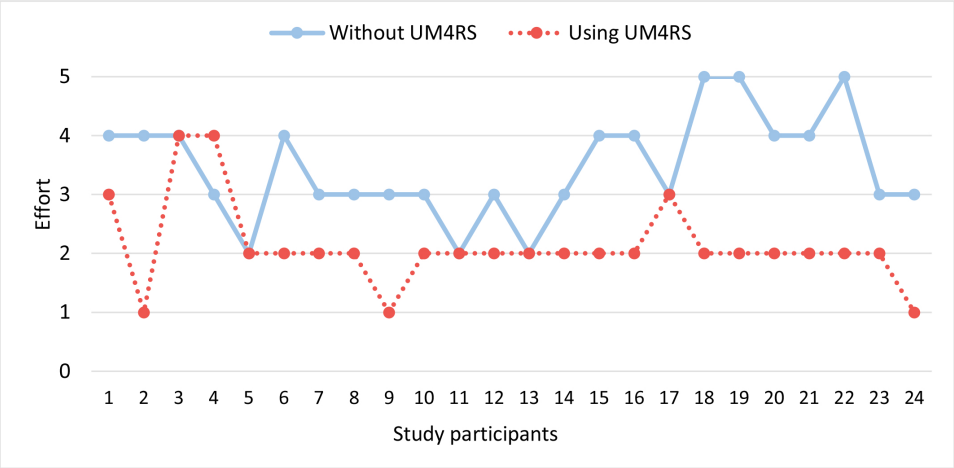

Figure 4.9: Comparison of effort required to implement the task without and with UM4RS.

Using the obtained 3.43 and 2.12 values for the expected and required effort respectively, and knowing that number of function points (task) used is 5, the productivity is calculated. Using the expected effort into the productivity formula (Equation 4.1), a productivity value of 1.45 is obtained. When using the value for the real effort, a productivity value of 2.35 is obtained as depicted in Figure 4.10. Then, the difference between those productivity values represents the increase in developers productivity, which is a 62% increase.

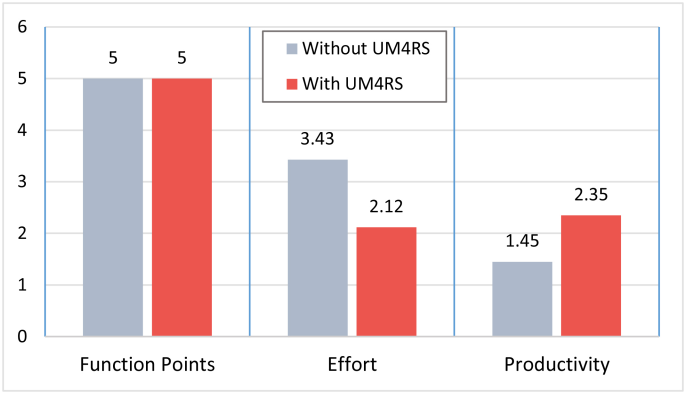

Figure 4.10: Comparison of developers productivity with and without using UM4RS (n=24).

As for the results of usability evaluation, the average SUS score for UM4RS was 78.64 with an sd of 13.9. Analyzing the SUS scores of industry participants and students separately, a slight difference appears, participants currently working as developers rated UM4RS API a 75.6 while students give an 82.27 score in the SUS scale. The average number of lines of code required to implement the 5 tasks was 72 lines when using UM4RS.

4.5.5 Discussion

All the participants of the experiment were able to complete assigned tasks using UM4RS in their first interaction with the framework, with just a short explanation of UM4RS' goal and using the online documentation.

The results obtained from the experiment suggest that using UM4RS, the assigned data management task were easier to implement by participant than they expected. In Figure 4.9 we can appreciate that in 75% of the cases, the required effort to implement the task was lower using UM4RS compared with the effort participants estimated they will need to implement them without using UM4RS. The 20.8% of participants needed the same effort with and without using UM4RS, and only 1 participant (4.1%) required more effort to implement the task when using UM4RS compared to not using any modeling framework.

The application of SUS questionnaire during this experiment allowed us to assess the usability level of UM4RS. Even when the sample size of 24 may appear too small, the research of [167] demonstrated that with SUS, a sample size of 12 participants is enough to give reliable measurements of the system usability. UM4RS got an average 78.64 in SUS scale with an sd of 13.90. According to [168, 169] the average SUS score for most commercial software systems is 68, any value lower than that is considered to have low usability, and values higher than that are considered to have good to excellent usability level. Figure 4.11 shows the interpretation graph for SUS values respect as proposed in [168], along with UM4RS SUS score.

As expected, by using UM4RS API, developers require fewer lines of code to implement the same data management task compared to not using the framework, as they will require to implement the classes to structure the information and the logic to persist the information of that classes to a database solution. In this particular case, comparing the average number of lines resulting from participants implementation of the 5 tasks, with an standard implementations of the same tasks without the framework, showed that by using UM4RS to implement this tasks, developers save 56% of the code. We firmly believe that using UM4RS in wider projects, the percentage of lines saved will be much greater, as this is a representative study of all the task required to implement a fully functional CARS.

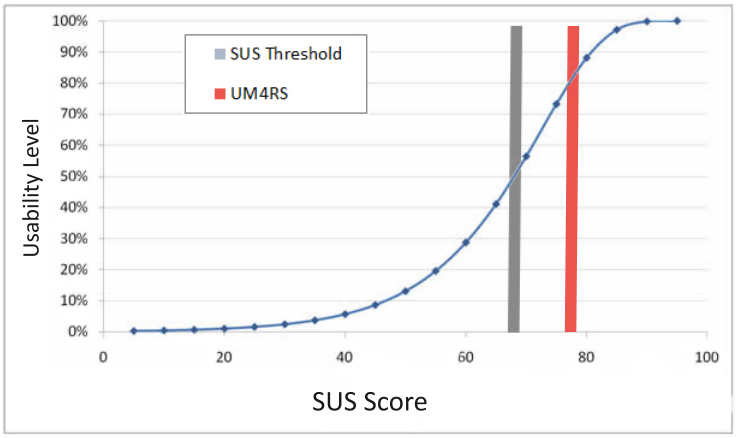

Figure 4.11: Interpretation graph for SUS scores.

Overall, the results obtained from this study suggest that UM4RS increases developers productivity in term of effort and lines of code required to implement the data management related task. In addition, the high level of usability of the framework suggests that UM4RS is easy to use by its target users, CARS developers.

4.6 Use case: Recommendation Framework

Even when the intention of this work is not to propose new or improve existing recommendation algorithms, this section describes a use case implemented to further test the applicability UM4RS.

For this use case, we develop a recommendation engine that help developers to create traditional or context-based recommender systems. The engine is called MoRe (*Modeling and Recommendation engine*), and was published in [170]. Even when similar tools exist (e.g. [11] and [124]) that allow developers to create recommender systems by providing implemented algorithms, none of them support developers with the management of the data. We see this as an area of opportunity and proposed to integrate UM4RS into a recommendation engine.

As a result, MoRe is the first comprehensive engine that not only provides developers with state of the art recommendation algorithms, but also includes a static structure that organizes the user, context, and items information. Also, MoRe includes the ability to automatically store, retrieve, and convert the store data into a dataset that is used as input for recommendation algorithms.

Next, Figure 4.12 presents how previous proposals (GUMCARS and UM4RS) are integrated into MoRe's architecture.

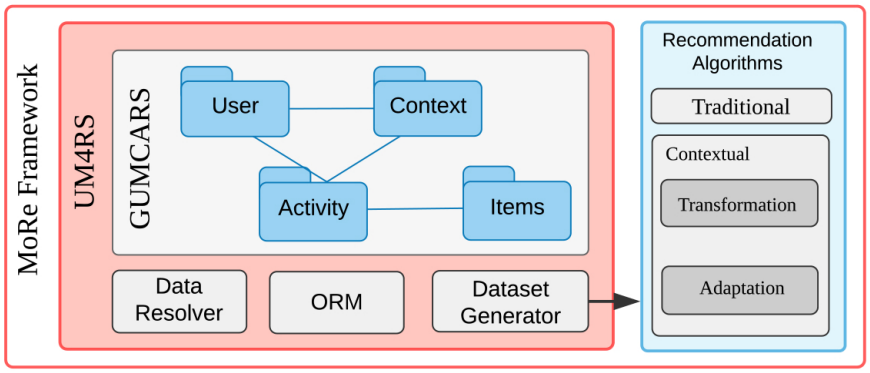

Figure 4.12: Interpretation graph for SUS scores.

4.6.1 Recommendation Algorithms

MoRe uses a large set of algorithms for both, traditional (2D) and contextual (nD) recommendation. These algorithms were originally implemented in Java by [171], for this use case, a custom wrapper was developed that allowed us to execute the algorithms from C# language.

MoRe target developers who do not want to spend too much time implementing existing recommendation techniques, or developers that have little experience implementing this type of complex systems. The recommendation algorithms can be used as a black box, which need to feed whit a dataset to yield a list of recommended item for the specified user. The framework also allows custom algorithms to be included, which makes MoRe ideal for researchers who want to test their new algorithm or recommendation approach against existing state of the art algorithms.

Even when MoRe is created to be a context-aware recommendation framework, it includes the following traditional recommendation techniques.

Traditional Recommendation Algorithms

Ranking Recommendation technique builds a recommendation model by analyzing the importance of each with respect all other items in the system [172]. The included algorithms for this technique are Sparce Linear Method (*SLIM*) and Bayesian Personalized Ranking (*BPR*).

Average Recommendation is a simple recommendation technique that perform prediction by the average rating given to an element [3]. Included algorithms are Global Average (*GlobalAvg*), User Average (*UserAvg*), and Item Average (*ItemAvg*).

Collaborative-Filtering Recommendation (CF) technique construct a model of similarities between users or items based on the idea that if two users gave a similar rating for a specific movies, the users or items are very similar [173]. Included algorithms are User K-Nearest-Neighbor (*UserKNN*),

Item K-Nearest-Neighbor (*ItemKNN*), and Singular Value Decomposition (*SVD++*).

Contextual Recommendation Algorithms

The multi-dimensional recommendation algorithms are the core of any recommendation framework, therefore MoRe provides several different algorithms from state of the art techniques, which gives developers the ability to choose the algorithm that yields better results for their specific domain, as some algorithms may perform better when recommending movies based on users companions, and others perform better when recommending songs or books.

Context Filtering Algorithms try to pre- or post-process the information and convert it from a multi-dimensional matrix to a 2-dimensional rating matrix that contains only the user reference (Id), item reference, and rating. Then, this two-dimensional matrix can be used in traditional algorithms to generate recommendations. There exist many contextual filtering algorithms; one of the most effective is the context-aware splitting [174]. MoRe includes the three main variants of this algorithm:

- *UserSplitting*: From a user point of view, the preference for certain items may vary depending on the context. User Splitting group items based on the context and the rating the user gave to them.
- *ItemSplitting*: Separates the items that were rated differently under different context as being different items.
- *UserItemSplitting*: Combines both previous separations, the result is that an item rated by a user in different context, is converted into 2 items, and the user is also divided into 2 users.

Contextual Modeling, unlike splitting approaches; take into account the influence of context aspects on the rating prediction model. The algorithms used for contextual modeling supported by MoRe are:

- *Tensor Factorization*: This algorithm is based on the Matrix Factorization (MF) dimensional re-duction technique which is used for 2D recommendations. TF [175] consist of extending the two-dimensional MF problem into a multi-dimensional version, where the rating tensor is converted into a lower-dimensional vector space. Such that the interactions between users, items, and contextual factors are represented by a latent factor vector.
- *Context-Aware Matrix Factorization* (CAMF) is a more scalable (than TF) contextual modeling approach based on MF. CAMF uses a contextual baseline prediction function to represent the interactions of contextual information with the items or users. Baltrunas et al. [114] proposed

different variants of CAMF that model the influence of contextual conditions at different granularities. CAMF-C assumes that a context factor affects the user and items in the same way. CAMF-CI models the in influence of a contextual aspect over items. And CAMF-CC assumes the context affects the ratings for all the items, ignoring the effect of context over users. MoRe contains the three variants (CAMF-C, CAMF-CI and CAMF-CC) variants of this algorithm.

4.6.2 Using MoRe to Perform Recommendations

The execution of the use case consisted of using the frameworks dataset generator to create the data in the database into a data matrix that can be fed to the algorithms, to generate recommendations simulating the process that will be followed in a real-world CARS implementation.

UM4RS will be used in real CARS implementation to continuously store data as it is gather from sensors or other sources. In this use case, to simulate how UM4RS will save gathered data, the LDOS CoMoDa [125] dataset (described in Section 3.4) was loaded using UM4RS API. Listing 4.3 shows part of the code used, showing how some columns of the dataset file are used to create *Users* in the framework.

Listing 4.3: Example code of loading dataset into UM4RS

```
// ...
string[] line = fileReader.ReadLine().Split(",");
User user = new User();
int userAge = int.Parse(line[3]);
user.Demographic = new Demographic()
{
  Age = userAge,
  Gender = line[4] == "1" ? Gender.Male : Gender.Female
};
// ...
user.Save();
// ...
```

Then, the dataset generation functionality of UM4RS is used to export the modeled data into a binarized dataset file which is the expected format of the algorithms. Finally, to get the list of recommended items, a recommender object is created from one of the algorithms contained in MoRe as show in the example Listing 4.4.

Listing 4.4: Example code to perform recommendations

```
 // ...
string workingDirectory = "<pathToMoRe>";
var recommender = new UISplitting()
{
    WorkPath = workingDirectory,
    Dataset = "ratings_binary.txt", // dataset file
    ShouldLoadModel = false,        // load a trained model
    TopN = 15,                      // tuning parameters
    KFold = 5,
    Neighbors = 51,
};
recommender.BuildModel();
var listOfRecommendations = recommender.GetRecommendationResults();
 // ...
```

To ensure that the data remains consistence after importing it where the data contained in a single file is separated and stored in different classes, and then is joined and formatted during the exportation process, the results of all the algorithms contained in MoRe is compared to the results presented in [174], where the same dataset is used across different recommendations algorithms, which allow us to directly compare the results from the matching algorithms.

Therefore, all the algorithms of MoRe were executed over the explored LDOS CoMoDa data, and the results were evaluated using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) ranking prediction metrics, except for SLIM and BPR algorithms, as they only support Top-N item recommendation and not rating prediction. Next, Figure 4.13 presents the obtained MAE and RMSE for the contextual modeling algorithms.

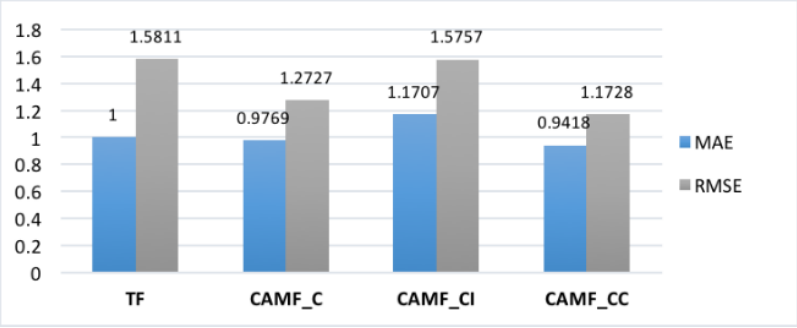

Figure 4.13: Interpretation graph for SUS scores.

Next, Table 4.4 presents the results for the combinations of filtering techniques with the baseline algorithms of MoRe.

Table 4.4: Evaluation results of Contextual Filtering algorithms on LDOS CoMoDa data.

Technique	**Algorithm**	**UserSplitting**		**ItemSplitting**		**UISplitting**	
		MAE	RMSE	MAE	RMSE	MAE	RMSE
Average	GlobalAvg	0.7700	0.9789	0.8509	1.0590	0.8509	1.0580
	UserAvg	0.7700	0.9789	0.7615	0.9757	0.7697	0.9788
	ItemAvg	0.4728	0.7461	0.4870	0.7700	0.4969	0.7701
Collaborative Filtering	SVD++	0.8573	1.0662	0.8561	1.0686	0.8552	1.0643
	UserKNN	0.8125	1.0447	0.8173	1.0501	0.8210	1.0502
	ItemKNN	0.7041	0.9272	0.7089	0.9331	0.7170	0.9388

Comparing the results of contextual modeling algorithms presented above, with the obtained in [174], the RMSE values obtained in MoRe are a bit higher, for example for CAMF_C we obtained 1.27 compared to 1.01, and for CAMF_CC 1.17 compared to 1.03 obtained in [174]. The difference obtained are because we used the entire LDOS CoMoDa data and XX used only two of the contextual dimensions contained in the dataset.

In the other hand, the results obtained for contextual filtering are similar to the presented in [174], for example, for *UserSplitting* technique combined with user-based algorithm (*UserAvg*) we obtained an RMSE 0.987 compared to a 0.978, and for *ItemSplitting* combined with user-based we obtained a RMSE 1.050 compared to 0.933.

Overall, this use case allowed us to test UM4RS and to gather more data about its behavior when using as part of a wider project, not only as a standalone tool, like used during the validations performed in this chapter. Also, the finding of this use case suggest that the consistency of the data is kept along the data flow of UM4RS, which means that even when the framework disjoin the data to store them in the corresponding classes, the link between the data is kept by the relationships between classes.

An aside result of this use case is MoRe, that by itself represents an interesting research path. MoRe provides developers a comprehensive recommendation framework that similarly to other frameworks, offers a set of state of the art algorithms, but with the plus of offering the data management functionality provided by UM4RS.

4.7 Conclusions and Summary

This chapter described the creation of UM4RS (pronounce umars), a user modeling framework for context-aware Recommender Systems. UM4RS is proposed to address the problem of lack of tools that support developers in the creation of CARS. The problem is address by providing developers a tool that saves developers the process of modeling and implementing a data structure.

UM4RS structure the information of users, context, and items using GUMCARS. In addition, the framework contains means to automatically persist the information contained in the model classes, and to format the stored data in different datasets formats that allows recommendation algorithms to make use of such data.

A validation of UM4RS structural correctness performed using software quality metrics suggest that the model elements are structured correctly. Also, an experiment was performed to measure the impact of that UM4RS has in developers productivity, obtained results strongly suggest that by using the framework to implement data management tasks, developers increase their productivity by decreasing the effort and lines of code required to the tasks, compared to not using the framework. This experiment also allowed us to measure the usability level of UM4RS, the obtained results showed that the framework has an above average usability level according to SUS scale, which means that the framework is easy to use by software developers.

Even when encouraging results were obtained in the applied validations, the framework has yet to be tested in different scenarios to gather more data about the behavior of the model when using in real-world applications. Nevertheless, we firmly believe support by the obtained results that the framework is applicable in the real world, and that will behave as expected in many different CARS scenarios.

Summary

This chapter allowed us to test the hypothesis and sub-hypothesis of thesis, as well as to achieve some objectives and goals as described next.

Section 1 and 2 Describe the design and implementation of the modeling framework, which directly attends **Goal 6** (*Create a user modeling framework ...*).

Section 3 Describes the creation of an API documentation to help developers learn to use the functionality provided by UM4RS, this directly attend **Goal 9** (*Create web-based documentation to allow developers explore and learn the functionality provided*).

Section 4 Validates UM4RS structural correctness by applying several object-oriented quality metrics, the obtained results support **Goal 7** (*Design the modeling framework to comply with object-oriented quality rules*).

Section 5 Presents the experiment performed to evaluate the impact of UM4RS on developers productivity. Obtained results show a decrease in the effort and lines of code required to implement data management tasks. Therefore, we consider that **Obj 3** (*Design and implement a user modeling framework to increase CARS developers productivity*) was achieved.

The same experiment also allowed us to measure the usability level of the framework, and the obtained results suggest that UM4RS has an above average usability level. Therefore, we consider that **Goal 8** (*Design the framework API to be easy to use by software developers*) was met.

Overall, this chapter allowed us to test the proposed user model inside a modeling framework, obtained results suggest that using such model help developers to increase their productivity; therefore we accept the hypothesis H_1**: A user modeling framework that structure user, context, and items information, increase developers productivity in terms of effort and lines of code required to implement a context-aware recommender systems**.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The work of this thesis focuses on the design of a user model structure and its later implementation into a user modeling framework that considers the information of user, context, and items that context-based recommendation algorithms require.

The main objective of this thesis is to support developers during the implementation phase of CARS, by providing a tool capable of structure, automatically store and retrieve data to and from a database, and that allows developers to easily format the data so it can be used in different recommendation algorithms.

In order to achieve such objective, first, we presented a GUMCARS, a general user model for context-aware recommender system that presents an extensive set of user, context, and items information aspects, which are used in CARS to generate recommendations. A quantitative validation of GUMCARS show that is complete enough to support the data from different real-world datasets loaded into it, and that is capable of cope with information from different recommendation domains like movies, travels, and food. These encouraging results shows that even when the information about humans and the environment that surrounds them is practically impossible to model in full, GUMCARS did grate selecting and modeling the information that CARS may use. Another quantitative evaluation allowed us to assess the structural correctness of GUMCARS using software quality metrics, and based on the results obtained; we conclude that the elements contained in the model are correctly structured, and interconnected. This is especially important as showed that

GUMCARS will maintain the inner relation between the data, and is capable of suffering adaptations without affecting the overall structure of the model.

As an extension of the proposed model, we also present UM4RS, a user modeling framework for context-aware recommender systems to further address the problem of lack of tools that facilitate the creation of CARS. UM4RS provide developers a comprehensive set of classes resulted from the implementation of GUMCARS, an ORM module that give classes the ability to store and retrieve its data into and from a database, and a dataset generator module that can be used to create a dataset file from the information contained in the model in different file formats used by recommendation algorithms.

A quantitative validation of UM4RS using software quality metrics showed that the classes, attributes, and methods of the framework are structured correctly. Also, this validation showed that UM4RS has a maintainability index of 94%, which reflects its capability to be adapted to suit specific project needs, like including some aspects information not currently supported by the framework's model or even to add the framework the feature of performing recommendation directly over the data stored.

As the goal of UM4RS is to help developers during the implementation of CARS, a qualitative user study was performed to evaluate the effect of UM4RS in software developers productivity. The study showed that by using UM4RS, developers required less effort than estimated to accomplish the assigned task, and fewer lines of code compared to an implementation from scratch. By decreasing the effort and lines of code required to implement data management functionality, UM4RS increases developers productivity. In addition to that, UM4RS showed that its programming interface has above average usability, and with the provided documentation developers solve assigned task with no training or previous interaction with the framework.

5.1 Future Work

While we have shown that GUMCARS and UM4RS fulfills the objective of this work, we envisage a number of future improvements. In the following list, we present of topics as possible research paths.

- The structure for Item's information of GUMCARS need to be further detailed to support new types of items, like learning resources and software applications. This will increase the application domain of the model.
- An interesting challenge is to test the ability of the framework to support multiple recommendation systems in a single instance of the data model. Such ability will convert the model into

a centralized data model for different applications, and will allow a challenging usage of data obtained in one recommendation domain as source of knowledge to improve recommendations in another domain.

- A great improvement for UM4RS will be inclusion of a future selection module. Currently, the model dataset generator is designed to export all the information or to let developers choose what information to consider in the exportation process. We believe that a mechanism to automatically select the most representative information from to model will improve the recommendation results, as CARS literature state that feeding all the information available to prediction algorithms not ensure better predictions.
- Another research path surged from the use case implementation of UM4RS. MoRe was the first recommendation engine that in addition to implemented algorithms provides developers the data management functionality of UM4RS. However, current version of the framework uses open-source recommendation algorithms, implemented in another language and rely on a custom wrapper for the interconnection. Implementing algorithms as part of the framework will increase the reliability, performance, integration with the systems, and may even improve the recommendation results.

5.2 Discussion of the Contributions

This thesis work presents the following theoretical and practical implications.

CARS information taxonomy A taxonomy that categorize the information user, context, and items is described. The taxonomy can be used as a reference model of how to organize the information needed by CARS algorithms.

Attribute collection An extensive list of information aspects about users, their surrounding context, and the items recommended by CARS is presented. The list can direct researchers on what information to use when testing new recommendation techniques, or use the list as an overview of the state of the art in CARS.

User model for CARS This thesis present a user model that structures the information that CARS algorithms uses as source of knowledge. The elements of the model are described in natural language as well as in UML class diagrams, which make it easy to perform implementation of the model in different programming languages.

User modeling framework A framework that implements the functionality of storage, retrieval, and formatting the data contained in the model into different datasets formats was created. The framework is implemented in C# language, using Entity Framework and SQL Server for the data persistence.

5.2.1 Main Contribution Papers

In this section, we present the list of publications resulting from this thesis. Each publication is part of the body of this document.

1. Inzunza, Sergio, & Juárez-Ramírez, Reyes. (2016). Building Context-Aware Recommendation Systems: A Software Engineering Point of View. In Software Engineering Research and Innovation (CONISOFT), 2016 4th International Conference in (pp. 175-184). IEEE.

2. Inzunza, Sergio, Juárez-Ramírez, Reyes, & Ramírez-Noriega, Alan (2016). User and context information in context-aware recommender systems: a systematic literature review. In New Advances in Information Systems and Technologies (pp. 649-658). Springer, Cham.

3. Inzunza, Sergio, Juárez-Ramírez, Reyes, & Jiménez, Samantha (2017). User Modeling Framework for Context-Aware Recommender Systems. In World Conference on Information Systems and Technologies (pp. 899-908). Springer, Cham.

4. Inzunza, Sergio, & Juárez-Ramírez, Reyes (2018). A Comprehensive Context-Aware Recommender System Framework. In Computer Science and Engineering-Theory and Applications (pp. 1-24). Springer, Cham.

5. Inzunza, Sergio, Juárez-Ramírez, Reyes, & Jiménez, Samantha (2018). API Documentation. In World Conference on Information Systems and Technologies (pp. 229-239).

BIBLIOGRAPHY

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, jun 2005.

[2] C. C. Aggarwal, "An Introduction to Recommender Systems," in *Recommender Systems: The Textbook*. Cham: Springer International Publishing, 2016, pp. 1–28.

[3] F. Ricci, L. Rokach, B. Shapira, P. B. Kantor, P. Lops, M. D. Gemmis, and G. Semeraro, *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Boston, MA: Springer US, 2011.

[4] S. Berkovsky, D. Heckmann, and T. Kuflik, "Addressing challenges of Ubiquitous User Modeling: Between mediation and semantic integration," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5830 LNCS, 2009, pp. 1–19.

[5] M. Morita and Y. Shinoda, "Information filtering based on user behavior analysis and best match text retrieval," *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 272–281, 1994.

[6] P. Resnick, H. R. Varian, and G. Editors, "ecommender Systems mmende tems," vol. 40, no. 3, pp. 56–58, 1997.

[7] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[8] L. O. Colombo-Mendoza, R. Valencia-García, A. Rodríguez-González, G. Alor-Hernández, and J. J. Samper-Zapater, "RecomMetz: A context-aware knowledge-based mobile recom-

mender system for movie showtimes," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1202–1222, feb 2015.

[9] K. Stefanidis, E. Ntoutsi, M. Petropoulos, and K. Hans-Peter, "A Framework for Modeling , Computing and Presenting Time-Aware Recommendations," in *Trans on Large-Scale Data and Knowledge-Centered Systems X.*, A. Hameurlain, J. Küng, R. Wagner, S. W. Liddle, K.-D. Schewe, and X. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 146–172.

[10] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin, "Incorporating contextual information in recommender systems using a multidimensional approach," pp. 103–145, 2005.

[11] T. Hussein, T. Linder, W. Gaulke, and J. Ziegler, "Hybreed: A software framework for developing context-aware hybrid recommender systems," *User Modeling and User-Adapted Interaction*, vol. 24, pp. 121–174, 2014.

[12] C. He, D. Parra, and K. Verbert, "Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities," *Expert Systems with Applications*, vol. 56, pp. 9–27, 2016.

[13] K. L. Skillen, L. Chen, C. D. Nugent, M. P. Donnelly, W. Burns, and I. Solheim, "Ontological user modelling and semantic rule-based reasoning for personalisation of Help-On-Demand services in pervasive environments," *Future Generation Computer Systems*, vol. 34, pp. 97–109, 2014.

[14] A. Hawalah and M. Fasli, "Utilizing contextual ontological user profiles for personalized recommendations," *Expert Systems with Applications*, vol. 41, no. 10, pp. 4777–4797, 2014.

[15] H. Lee, Y. Choi, and Y. Kim, "An adaptive user interface based on Spatiotemporal Structure Learning," *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, vol. 49, no. 6, pp. 923–927, jan 2011.

[16] B. Chen, P. Yu, C. Cao, F. Xu, and J. Lu, "ConRec: A Software Framework for Context-Aware Recommendation Based on Dynamic and Personalized Context," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2, 2015, pp. 816–821.

[17] S. Berkovsky, T. Kuflik, and F. Ricci, "Mediation of user models for enhanced personalization in recommender systems," *User Modelling and User-Adapted Interaction*, vol. 18, no. 3, pp. 245–286, 2008.

[18] H.-N. Kim, I. Ha, K.-S. Lee, G.-S. Jo, and A. El-Saddik, "Collaborative user modeling for enhanced content filtering in recommender systems," *Decision Support Systems*, vol. 51, no. 4, pp. 772–781, 2011.

[19] G. Jawaheer, P. Weller, and P. Kostkova, "Modeling User Preferences in Recommender Systems," *ACM Transactions on Interactive Intelligent Systems*, vol. 4, no. 2, pp. 1–26, 2014.

[20] P. Germanakos and M. Belk, *Human- Centred Web Adaptation and Personalization From Theory to Practice*. Springer, 2016.

[21] C. Mettouris and G. A. Papadopoulos, "Designing Context Models for CARS Incorporating Partially Observable Context," pp. 118–131, 2015.

[22] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[23] B. Djoudi, C. Bouanaka, and N. Zeghib, "A formal framework for context-aware systems specification and verification," *Journal of Systems and Software*, vol. 122, pp. –, 2015.

[24] J. Luo and H. Feng, "A Web-Based Framework for Lightweight Context-Aware Mobile Applications," *International Journal of Database Theory and Application*, vol. 9, no. 4, pp. 119–134, 2016.

[25] G. Shani and A. Gunawardana, "Evaluating recommendation systems," *Recommender systems handbook*, pp. 1–41, 2011.

[26] U. Alegre, J. C. Augusto, and T. Clark, "Engineering context-aware systems and applications: A survey," *Journal of Systems and Software*, vol. 117, pp. 55–83, 2016.

[27] C. Mettouris and G. A. Papadopoulos, "Using Appropriate Context Models for CARS Context Modelling," in *Knowledge, Information and Creativity Support Systems: Selected Papers from KICSS'2014 - 9th International Conference, held in Limassol, Cyprus, on November 6-8, 2014*, S. Kunifuji, A. G. Papadopoulos, M. J. A. Skulimowski, and K. Janusz, Eds. Cham: Springer International Publishing, 2016, pp. 65–79.

[28] D. Heckmann, *Ubiquitous User Modeling*, 2005, vol. 297.

[29] K. Verbert and N. Manouselis, "Context-aware recommender systems for learning: a survey and future challenges," *Learning . . .*, vol. 5, no. 4, pp. 318–335, 2012.

[30] C. Yaman and N. Cicekli, "A Content Recommendation Framework Using Ontological User Profile," 2013.

[31] T. Walter, F. S. Parreiras, and S. Staab, "An ontology-based framework for domain-specific modeling," *Software & Systems Modeling*, vol. 13, no. 1, pp. 83–108, apr 2012.

[32] M. D. Ekstrand, M. Ludwig, J. a. Konstan, and J. T. Riedl, "Rethinking the Recommender Research Ecosystem : Categories and Subject Descriptors," *Proceedings of the 5th ACM conference on Recommender systems - RecSys '11*, pp. 133–140, 2011.

[33] H. Gomaa, *Software modeling and design: UML, use cases, patterns, and software architectures*, 2011.

[34] J. Beel, B. Gipp, S. Langer, and C. Breitinger, "Research-paper recommender systems: a literature survey," *International Journal on Digital Libraries*, no. February 2014, 2015.

[35] D. Asanov, "Algorithms and Methods in Recommender Systems," *Other Conferences*, 2011.

[36] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[37] J. Dietmar, Z. Markus, A. Felfernig, and G. Friedrich, *Recommender Systems: An Introduction*, 2010.

[38] A. Abbas, L. Zhang, and S. U. Khan, "A survey on context-aware recommender systems based on computational intelligence techniques," *Computing*, 2015.

[39] J. Beel, S. Langer, and M. Genzmehr, "Research paper recommender system evaluation: a quantitative literature survey," *Proceedings of the . . .* , no. April, pp. 15–22, 2013.

[40] D. Heckmann and T. Schwartz, "Decentralized user modeling with UserML and GUMO," *. . . , Agent Based and . . .* , 2005.

[41] T. De Pessemier, K. Vanhecke, and L. Martens, "A Personalized and Context-Aware News Offer for Mobile Devices," in *Web Information Systems and Technologies: 11th International Conference, WEBIST 2015, Lisbon, Portugal, May 20-22, 2015*, V. Monfort and K.-H. Krempels, Eds. Cham: Springer International Publishing, 2016, pp. 147–168.

[42] Y. Shi, H. Lin, and Y. Li, "Context-Aware Recommender Systems Based on Item-Grain Context Clustering," in *AI 2017: Advances in Artificial Intelligence: 30th Australasian Joint Conference, Melbourne, VIC, Australia, August 19–20, 2017, Proceedings*, W. Peng, D. Alahakoon, and X. Li, Eds. Cham: Springer International Publishing, 2017, pp. 3–13.

[43] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," *Recommender systems handbook*, pp. 67–80, 2011.

[44] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," *Computing Systems*, vol. 40, pp. 304–307, 1999.

[45] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. IEEE, 1994, pp. 85–90.

[46] A. Umyarov and A. Tuzhilin, "Using external aggregate ratings for improving individual recommendations," *ACM Transactions on the Web*, vol. 5, no. 1, pp. 1–40, 2011.

[47] A. K. Dey and G. D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," vol. 16, pp. 97–166, 2001.

[48] G. Siolas, G. Caridakis, P. Mylonas, S. Kollias, and A. Stafylopatis, "Context-Aware User Modeling and Semantic Interoperability in Smart Home Environments," *2013 8th International Workshop on Semantic and Social Media Adaptation and Personalization*, pp. 27–32, dec 2013.

[49] D. R. Morse and A. K. Dey, "The what, who, where, when, why and how of context-awareness," *CHI EA '00 CHI '00 Extended Abstracts on Human Factors in Computing Systems*, pp. 371–372, 2000.

[50] K. Mitchell and K. Mitchell, "A Survey on Context Awareness," pp. 144–147, 2002.

[51] O. Yurur, C. H. Liu, Z. Sheng, V. C. Leung, W. Moreno, and K. K. Leung, "Context-awareness for mobile sensing: A survey and future directions," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 68–93, 2016.

[52] O. Saidani, C. Rolland, and S. Nurcan, "Towards a generic context model for BPM," *Proceedings of the Annual Hawaii International Conference on System Sciences*, vol. 2015-March, pp. 4120–4129, 2015.

[53] P. Dourish, "What we talk about when we talk about context," *Personal and Ubiquitous Computing*, vol. 8, no. 1, pp. 19–30, 2004.

[54] T. Kuflik, J. Kay, and B. Kummerfeld, "Challenges and Solutions of Ubiquitous User Modeling," in *Ubiquitous Display Environments*, ser. Cognitive Technologies, A. Krüger and T. Kuflik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 7–30.

[55] ——, "Ubiquitous Display Environments," pp. 7–30, 2012.

[56] A. KOBSA, J. KOENEMANN, and W. POHL, "Personalised hypermedia presentation techniques for improving online customer relationships," p. 111, 2001.

[57] J. Schreck, *Security and Privacy in User Modeling*, 2003, no. September.

[58] J. Zhang and A. Ghorbani, "GUMSAWS: A generic user modeling server for adaptive web systems," ... *Research, 2007. CNSR'07. Fifth Annual* ..., pp. 117–124, may 2007.

[59] L. Kuflik, "Advances in User Modeling," in *Advances in User Modeling: UMAP 2011 Workshops, Girona, Spain, July 11-15, 2011, Revised Selected Papers*. Springer, 2012.

[60] J. Fink, "User modeling servers: Requirements, design, and evaluation," PhD, Universität Duisburg-Essen, Standort Essen, 2004.

[61] D. Zhang, H. Huang, C. F. Lai, X. Liang, Q. Zou, and M. Guo, "Survey on context-awareness in ubiquitous media," *Multimedia Tools and Applications*, vol. 67, no. 1, pp. 179–211, 2013.

[62] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc* ..., vol. 2, no. 4, 2007.

[63] A. a. A. Sabagh and A. Al-Yasiri, "GECAF: a framework for developing context-aware pervasive systems," *Computer Science - Research and Development*, sep 2013.

[64] Dan Brickley and R.V. Guha, "Resource Description Framework (RDF) Schema Specification 1.0," 2000.

[65] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language for Object-Oriented Development," *Unix Review*, vol. 14, no. 13, p. 29, 1996. [Online]. Available: http://www.ccs.neu.edu/research/demeter/course/f96/readings/uml9.pdf

[66] Ormfoundation.org, "The ORM Foundation," 2017. [Online]. Available: www.ormfoundation.org

[67] N. Kaklanis, P. Biswas, Y. Mohamad, M. F. Gonzalez, M. Peissner, P. Langdon, D. Tzovaras, and C. Jung, "Towards standardisation of user models for simulation and adaptation purposes," *Universal Access in the Information Society*, aug 2014.

[68] A. Zimmermann, A. Lorenz, and R. Oppermann, "An Operational Definition of Context," in *6th International and Interdisciplinary Conference, CONTEXT 2007*, vol. 4635, 2007, pp. 558–571.

[69] E. Churchill, L. L. Laubach, J. T. McConville, and I. Tebbetts, "Anthropometric source book. Volume 1: Anthropometry for designers," 1978.

[70] M. F. Alhamid, M. Rawashdeh, H. Al Osman, M. S. Hossain, and A. El Saddik, "Towards context-sensitive collaborative media recommender system," *Multimedia Tools and Applications*, 2014.

[71] B. Shapira, L. Rokach, and S. Freilikhman, "Utilizing Facebook Single and Cross Domain Data for Recommendation Systems," *User Modeling and User-Adapted Interaction*, vol. 23, no. 2-3, pp. 211–247, sep 2012.

[72] D. Heckmann and E. Schwarzkopf, "The user model and context ontology gumo revisited for future web 2.0 extensions," 2007.

[73] M. Braunhofer and F. Ricci, "Contextual Information Elicitation in Travel Recommender Systems," in *Information and Communication Technologies in Tourism 2016: Proc of the Int Conf in Bilbao, Spain, 2016*, A. Inversini and R. Schegg, Eds. Cham: Springer International Publishing, 2016, pp. 579–592.

[74] O. P. John, E. Donahue, and R. Kentle, "The Big Five," *Factor Taxonomy: Dimensions of Personality in the Natural Language and in Questionnaires."In Handbook of Personality: Theory and Research, ed. Lawrence A. Pervin and Oliver P. John*, pp. 66–100, 1990.

[75] C.-k. Hsieh, L. Yang, C. Tech, C. Tech, and D. Estrin, "Immersive Recommendation : News and Event Recommendations Using Personal Digital Traces," *Www*, pp. 51–62, 2016.

[76] J. J. Deng, C. H. C. Leung, A. Milani, and L. I. Chen, "Emotional States Associated with Music : Classification , Prediction of Changes , and Consideration in Recommendation," vol. 5, no. 1, 2015.

[77] V. Codina, F. Ricci, and L. Ceccaroni, "Distributional semantic pre-filtering in context-aware recommender systems," *User Modeling and User-Adapted Interaction*, 2015.

[78] H. Yin and B. Cui, "Spatial Context-Aware Recommendation," in *Spatio-Temporal Recommendation in Social Media*. Singapore: Springer Singapore, 2016, pp. 41–63.

[79] R. Beale and P. Lonsdale, "Mobile Context Aware Systems: The Intelligence to Support Tasks and Effectively Utilise Resources," in *Mobile Human-Computer Interaction - MobileHCI 2004*, 2004, vol. 3160, pp. 240–251.

[80] W. W. W. Consortium and Others, "Composite Capabilities/Preference Profiles," *See http://www. w3. org/Mobile/CCPP*, 2004.

[81] a. a. Hamed, R. Roose, M. Branicki, and A. Rubin, "T-Recs: Time-aware Twitter-based Drug Recommender System," *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 1027–1031, aug 2012.

[82] A. Seth and J. Zhang, "A Social Network Based Approach to Personalized Recommendation of Participatory Media Content," in *Sociological Theory*, 2008, pp. 109–117.

[83] X. Fan, J. Li, and C. Wang, "Context-aware Ubiquitous Web Services Recommendation based on User Location Update," 2015.

[84] F. Ricci, "First International Workshop on Decision Making and Recommendation Acceptance Issues in Recommender Systems (DEMRA 2011) and Second International Workshop on User Models for Motivational Systems : the affective and the rational routes to persuasion (UM," no. Demra, 2011.

[85] C. C. Aggarwal, "Recommender Systems The Textbook," in *Recommender Systems The Textbook*, 2016, pp. 1–29.

[86] S. Inzunza, R. Juárez-Ramírez, and A. Ramírez-Noriega, "User and Context Information in Context-Aware Recommender Systems: A Systematic Literature Review," in *New Advances in Information Systems and Technologies*, Á. Rocha, M. A. Correia, H. Adeli, P. L. Reis, and M. Mendonça Teixeira, Eds. Cham: Springer International Publishing, 2016, vol. 444, pp. 649–658.

[87] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - A systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2009.

[88] H. P. Breivold, I. Crnkovic, and M. Larsson, "A systematic review of software architecture evolution research," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 16–40, 2012.

[89] Z. D. Champiri, S. R. Shahamiri, and S. S. B. Salim, "A systematic review of scholar context-aware recommender systems," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1743–1758, sep 2014.

[90] G. Adomavicius and D. Jannach, "Preface to the special issue on context-aware recommender systems," *User Modeling and User-Adapted Interaction*, vol. 24, no. 1-2, pp. 1–5, 2014.

[91] T. De Pessemier, C. C. Courtois, K. Vanhecke, K. Van Damme, L. Martens, and L. De Marez, "A user-centric evaluation of context-aware recommendations for a mobile news service," *Multimedia Tools and Applications*, vol. 75, no. 6, pp. 3323–3351, 2015.

[92] J. Han, H. R. Schmidtke, X. Xie, and W. Woo, "Adaptive content recommendation for mobile users: Ordering recommendations using a hierarchical context model with granularity," *Pervasive and Mobile Computing*, vol. 13, pp. 85–98, 2014.

[93] M. Schedl, "Ameliorating Music Recommendation: Integrating Music Content, Music Context, and User Context for Improved Music Retrieval and Recommendation," *Proceedings of International Conference on Advances in Mobile Computing & Multimedia*, pp. 3:3—-3:9, 2013.

[94] G. Chen and L. Chen, "Augmenting service recommender systems by incorporating contextual opinions from user reviews," *User Modeling and User-Adapted Interaction*, 2015.

[95] A. M. Otebolaku and M. T. Andrade, "Context-aware media recommendations," *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014*, vol. 1, pp. 191–196, 2014.

[96] ——, "Context-aware media recommendations for smart devices," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 1, pp. 13–36, 2014.

[97] M. Kaminskas and F. Ricci, "Contextual music information retrieval and recommendation: State of the art and challenges," *Computer Science Review*, vol. 6, no. 2-3, pp. 89–119, 2012.

[98] H. Yin, B. I. N. Cui, L. Chen, Z. Hu, and X. Zhou, "Dynamic User Modeling in Social Media Systems," vol. 33, no. 3, 2015.

[99] D. Bouneffouf, A. Bouzeghoub, and A. L. Gançarski, "Following the user's interests in mobile context-aware recommender systems: The hybrid-e-greedy algorithm," *Proceedings - 26th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2012*, pp. 657–662, 2012.

[100] J. Kim, D. Lee, and K. Y. Chung, "Item recommendation based on context-aware model for personalized u-healthcare service," *Multimedia Tools and Applications*, vol. 71, no. 2, pp. 855–872, 2014.

[101] W. P. Lee and K. H. Lee, "Making smartphone service recommendations by predicting users' intentions: A context-aware approach," *Information Sciences*, vol. 277, pp. 21–35, 2014.

[102] L. Li, L. Zheng, F. Yang, and T. Li, "Modeling and broadening temporal user interest in personalized news recommendation," *Expert Systems with Applications*, vol. 41, no. 7, pp. 3168–3177, 2014.

[103] K. L. Skillen, L. Chen, C. D. Nugent, M. P. Donnelly, W. Burns, and I. Solheim, "Ontological user profile modeling for context-aware application personalization," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7656 LNCS, 2012, pp. 261–268.

[104] P. G. Campos, F. Díez, and I. Cantador, "Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols," *User Modeling and User-Adapted Interaction*, vol. 24, no. 1-2, pp. 67–119, feb 2013.

[105] M. Harvey, B. Ludwig, and D. Elsweiler, "You are what you eat : learning user tastes for rating prediction," in *String Processing and Information Retrieval: 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, O. Kurland, M. Lewenstein, and E. Porat, Eds. Cham: Springer International Publishing, 2013, pp. 153–164.

[106] S. Wu, Q. Liu, L. Wang, S. Member, and T. Tan, "Contextual Operation for Recommender Systems," vol. 4347, no. c, pp. 1–15, 2016.

[107] I. Benouaret and D. Lenne, "Personalizing the Museum Experience through Context-Aware Recommendations," in *Proceedings - 2015 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2015*, 2016, pp. 743–748.

[108] M. S. Hossain and G. Muhammad, "STCAPLRS : A Spatial-Temporal Context-Aware Personalized," vol. 7, no. 4, pp. 1–30, 2016.

[109] H. Bagci and P. Karagoz, "Context-aware location recommendation by using a random walk-based approach," *Knowledge and Information Systems*, vol. 47, no. 2, pp. 241–260, 2016.

[110] V. C. Storey, J. C. Trujillo, and S. W. Liddle, "Research on conceptual modeling: Themes, topics, and introduction to the special issue," *Data & Knowledge Engineering*, vol. 98, pp. 1–7, 2015.

[111] B. Grady, "Object-oriented analysis and design with applications," 1994.

[112] B. Lamche, Y. Rödl, C. Hauptmann, and W. Wörndl, "Context-aware recommendations for mobile shopping," *CEUR Workshop Proceedings*, vol. 1405, pp. 21–27, 2015.

[113] N. Landia, "Building Recommender Systems for Fashion," *Proceedings of the Eleventh ACM Conference on Recommender Systems - RecSys '17*, pp. 343–343, 2017.

[114] L. Baltrunas, B. Ludwig, S. Peer, and F. Ricci, "Context relevance assessment and exploitation in mobile recommender systems," *Personal and Ubiquitous Computing*, vol. 16, no. 5, pp. 507–526, 2012.

[115] W. N. Morris, "More on the mood-emotion distinction," *Psycoloquy*, vol. 3, no. 7, 1992.

[116] C. Rana and S. K. Jain, "A study of the dynamic features of recommender systems," *Artificial Intelligence Review*, no. November 2012, pp. 1–13, 2012.

[117] D. L. Moody, G. Sindre, T. Brasethvik, and A. Solvberg, "Evaluating the quality of information models: empirical testing of a conceptual model quality framework," in *25th International Conference on Software Engineering, 2003. Proceedings.*, may 2003, pp. 295–305.

[118] O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *IEEE Software*, vol. 11, no. 2, pp. 42–49, mar 1994.

[119] D. L. Moody, "Measuring the quality of data models: an empirical evaluation of the use of quality metrics in practice," *ECIS 2003 Proceedings*, p. 78, 2003.

[120] ——, "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions," *Data & Knowledge Engineering*, vol. 55, no. 3, pp. 243–276, 2005.

[121] K. Oku, S. Nakajima, J. Miyazaki, and S. Uemura, "Context-aware SVM for context-dependent information recommendation," in *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*. IEEE, 2006, p. 109.

[122] Y. Zheng, B. Mobasher, and R. Burke, "Context Recommendation Using Multi-label Classification," in *Proceedings of the 13th IEEE/WIC/ACM International Conference on Web Intelligence (WI 2014)*. IEEE/WIC/ACM, 2014.

[123] M. Elahi, M. Braunhofer, F. Ricci, and M. Tkalcic, "Personality-Based Active Learning for Collaborative Filtering Recommender Systems," in *International Conference of the Italian Association for Artificial Intelligence, 2013.*, M. Baldoni, C. Baroglio, G. Boella, and R. Micalizio, Eds. Cham: Springer International Publishing, 2013, pp. 360–371.

[124] Y. Zheng, B. Mobasher, and R. Burke, "CARSKit: A Java-Based Context-aware Recommendation Engine," in *Proceedings of the 15th IEEE International Conference on Data Mining Workshops*. NJ USA: IEEE, 2015.

[125] A. Košir, A. Odic, M. Kunaver, M. Tkalcic, and J. F. Tasic, "Database for contextual personalization," *Elektrotehniki vestnik*, vol. 78, no. 5, pp. 270–274, 2011.

[126] P. Adamopoulos, "ConcertTweets: A Multi-Dimensional Data Set for Recommender Systems Research."

[127] L. Baltrunas, M. Kaminskas, B. Ludwig, O. Moling, F. Ricci, A. Aydin, K.-H. Lüke, and R. Schwaiger, "Incarmusic: Context-aware music recommendations in a car," in *E-Commerce and Web Technologies*. Springer, 2011, pp. 89–100.

[128] M. Genero, M. Piattini, and C. Calero, "A survey of metrics for UML class diagrams," pp. 59–92, 2005.

[129] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[130] S. Counsell, X. Liu, S. Eldh, R. Tonelli, M. Marchesi, G. Concas, and A. Murgia, "Revisiting the 'Maintainability Index' Metric from an Object-Oriented Perspective," *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, pp. 84–87, 2015.

[131] Z. Naboulsi, "Code Metrics Depth of Inheritance (DIT)," 2011. [Online]. Available: https://blogs.msdn.microsoft.com/zainnab/2011/05/19/code-metrics-depth-of-inheritance-dit/

[132] P. Smacchia, "NDepend Code Metrics Definitions," 2016.

[133] T. Shatnawi, Raed and Li, Wei and Swain, James and Newman, "Finding Software Metrics Threshold Values Using ROC Curves," *J. Softw. Maint. Evol.*, vol. 22, no. 1, pp. 1–16, 2010.

[134] M. A. S. Bigonha and K. A. M. Ferreira, "A Catalogue of Thresholds for Object-Oriented Software Metrics," in *SOFTENG 2015 : The First International Conference on Advances and Trends in Software Engineering A*, no. Dcc, 2015, pp. 48–55.

[135] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[136] Z. Naboulsi, "Code Metrics - Class Coupling," 2011. [Online]. Available: https://blogs.msdn.microsoft.com/zainnab/2011/05/25/code-metrics-class-coupling/

[137] I. S. O. ISO and T. R. IEC, "9126-2: Software Engineering-Product Quality-Part 2: External Metrics," *International Organization for Standardization, Geneva, Switzerland*, 2003.

[138] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44–49, 1994.

[139] I. Samoladas, I. Stamelos, L. Angelis, and A. Oikonomou, "Open source software development should strive for even greater code maintainability," *Communications of the ACM*, vol. 47, no. 10, pp. 83–87, 2004.

[140] Z. Naboulsi, "Code Metrics - Maintainability Index," 2011. [Online]. Available: https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/

[141] G. Booch, *Object oriented analysis & design with application*. Pearson Education India, 2006.

[142] A. Troelsen, P. Japikse, A. Troelsen, and P. Japikse, "ADO. NET Part III: Entity Framework," *C# 6.0 and the. NET 4.6 Framework*, pp. 929–999, 2015.

[143] J. Lerman and R. Miller, *Programming entity framework: code first*. " O'Reilly Media, Inc.", 2011.

[144] G. Uddin and M. P. Robillard, "How API Documentation Fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015.

[145] M. Ellmann, "On the similarity of software development documentation," *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, pp. 1030–1033, 2017.

[146] Y. Zhou, R. Gu, T. Chen, and Z. Huang, "Analyzing APIs Documentation and Code to Detect Directive Defects," no. May, 2017.

[147] M. Barth, "API Evaluation An overview of API evaluation techniques," 2015.

[148] P. Gruenbaum, "Hello World! - A Coder's Guide to Writing API Documentation," 2014. [Online]. Available: https://msdn.microsoft.com/en-us/magazine/gg309172.aspx

[149] G. M. Rama and A. Kak, "Some structural measures of API usability," *Software: Practice and Experience*, vol. 45, no. 1, pp. 75–110, 2015.

[150] B. De, "API Management," in *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization*. Berkeley, CA: Apress, 2017, pp. 15—-28.

[151] T. J. Mccabe, "A Complexity Measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[152] H. Bhasin, D. Sharma, R. Popli, and A. D. Metrics, "On the reliance of COM Metrics for a C # Project," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 4288–4291, 2014.

[153] Microsoft, "Code Metrics Values," 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/bb385914.aspx

[154] L. H. Rosenberg, "Applying and Interpreting Object Oriented Metrics," *Object Oriented Systems*, 1998.

[155] M. Perepletchikov, C. Ryan, and K. Frampton, "Cohesion metrics for predicting maintainability of service-oriented software," *Proceedings - International Conference on Quality Software*, no. Qsic, pp. 328–335, 2007.

[156] T. Akanmu, S. Olabiyisi, E. Omidiora, C. Oyeleye, M. Mabayoje, and A. Babatunde, "Comparative Study of Complexities of Breadth- First Search and Depth-First Search Algorithms using Software Complexity Measures," *Proceedings of the World Congress on Engineering*, vol. I, 2010.

[157] I. Heitlager, T. Kuipers, and J. Visser, "A Practical Model for Measuring Maintainability," *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, pp. 30–39, 2007.

[158] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*, third edit ed. CRC Press, 2014.

[159] D. N. Card, "The Challenge of Productivity Measurements," *Pacific Northwest Software Quality Conference*, pp. 1–10, 2006.

[160] M. A. Lapré, J. D. Blackburn, and L. N. Van Wassenhove, "Brooks law revisited: Improving software productivity by managing complexity," *Organizational Learning: Individual Differences, Technologies and Impact of Teaching*, pp. 49–70, 2015.

[161] S. Clarke, "Measuring API usability," *Doctor Dobbs Journal*, vol. 29, no. 1, pp. S1—-S5, 2004.

[162] W. D. ISO, "9241-11. Ergonomic requirements for office work with visual display terminals (VDTs)," *The international organization for standardization*, 1998.

[163] A. J. Albrecht, "Measuring application development productivity," in *IBO Conference on Application Development*, 1979, pp. 83–92.

[164] R. Thackston and D. Umphress, "Individual Effort Estimating Not Just for Teams Anymore," *Crosstalk*, vol. 25, no. 3, pp. 4–7, 2012.

[165] M. Cohn, *Agile estimating and planning*. Pearson Education, 2005.

[166] W. Albert and E. Dixon, "Is this what you expected? The use of expectation measures in usability testing," in *Proceedings of Usability Professionals Association 2003 Conference, Scottsdale, AZ*, 2003.

[167] J. Brooke, "SUS-A quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, p. 194, 1996.

[168] J. Sauro, "Measuring usability with the system usability scale (SUS)," 2011.

[169] A. Bangor, P. Kortum, and J. Miller, "Determining what individual SUS scores mean: Adding an adjective rating scale," *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.

[170] S. Inzunza and R. Juárez-Ramírez, *A Comprehensive Context-Aware Recommender System Framework*, 2018, vol. 143.

[171] Y. Zheng, "A User's Guide to CARSKit," pp. 1–7, 2015.

[172] X. Ning and G. Karypis, "SLIM : Sparse Linear Methods for Top-N Recommender Systems," no. December 2011, pp. 1–10, 2011.

[173] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, 2013.

[174] Y. Zheng, R. Burke, and B. Mobasher, "Splitting approaches for context-aware recommendation," *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, pp. 274–279, 2014.

[175] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering," *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*, p. 79, 2010.