University of Cyprus

DEPARTMENT OF COMPUTER SCIENCE

A MODEL-DRIVEN DEVELOPMENT FRAMEWORK FOR AUTOMATED DESIGN AND DEVELOPMENT OF UBIQUITOUS CONTEXT-AWARE RECOMMENDER SYSTEMS FOR COMMERCE

CHRISTOS METTOURIS

A Dissertation Submitted to the University of Cyprus in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

December 2018

VALIDATION PAGE

Doctoral Candidate: Christos Mettouris

Doctoral Thesis Title: A Model-Driven Development Framework for Automated Design and Development of Ubiquitous Context-Aware Recommender Systems for Commerce

The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Department of Computer Science and was approved on December 14th, 2018 by the members of the Examination Committee.

Examination Committee:

Research Supervisor _____

Professor George A. Papadopoulos

Committee Chair _____

Assistant Professor Georgia Kapitsaki

Committee Member _____

Assistant Professor George Pallis

Committee Member _____

Lecturer Andreas Konstantinidis

Committee Member _____

Assistant Professor Mihalis A. Nicolaou

DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Christos Mettouris

……

Περίληψη

Τα συστήματα συστάσεων (recommender systems), για περισσότερο από δύο δεκαετίες αποτελούν την απάντηση στον υπερβολικό όγκο πληροφορίας που υπερφορτώνει τους καταναλωτές στη σύγχρονη ζωή. Τα συστήματα συστάσεων είναι εργαλεία λογισμικού τα οποία στόχο έχουν να ανακαλύπτουν την απαραίτητη γνώση σχετικά με τους χρήστες τους, προκειμένου να τους προσφέρουν εξατομικευμένες συστάσεις (personalised recommendations). Οι συστάσεις προϊόντων σε πελάτες έχουν αποδειχθεί να αυξάνουν τις πωλήσεις, να προσφέρουν μεγαλύτερη ικανοποίηση στους πελάτες και να βελτιώνουν την εμπειρία των πελατών, καθιστώντας τα συστήματα συστάσεων ως ένα σημαντικό εργαλείο για τις επιχειρήσεις λιανικού εμπορίου. Ένα βασικό ερώτημα είναι πώς μπορούν οι επιχειρήσεις να αναπτύξουν συστήματα συστάσεων στα διαδικτυακά τους καταστήματα;

Παρά την επιτυχία των συστημάτων συστάσεων στη αγορά, περιορισμένη έρευνα έχει διεξαχθεί όσον αφορά σε θέματα μηχανικής λογισμικού τέτοιων περίπλοκων συστημάτων, και πιο συγκεκριμένα, ότι αφορά την υποστήριξη κατά το σχεδιασμό και την υλοποίηση τους. Η υλοποίηση αποτελεσματικών συστημάτων συστάσεων που να χρησιμοποιούν πληροφορία πλαισίου (context-aware) από μη εξειδικευμένους επαγγελματίες προγραμματιστές δεν είναι εύκολη διαδικασία, λόγω της πολυπλοκότητας στη δημιουργία των απαραίτητων μοντέλων δεδομένων, αλλά και στην επιλογή και διαμόρφωση (configuration) των αλγορίθμων συστάσεων. Η απουσία επιστημονικών εργασιών που να χρησιμοποιούν μεθόδους ανάπτυξης λογισμικού με γνώμονα μοντέλα

(Model-Driven Development), με σκοπό το σχεδιασμό, την υλοποίηση και την ανάπτυξη ευρέων (Ubiquitous) συστημάτων συστάσεων επίγνωσης πλαισίου (Context-Aware Recommender Systems), εν συντομία UbiCARS, που να στοχεύουν στην αφαιρετικότητα των τεχνικών λεπτομεριών, στη μείωση της πολυπλοκότητας υλοποίησης, αλλά και στην επίσπευση της υλοποίησης των προαναφερθέντων συστημάτων, έχει αποτελέσει κίνητρο για την εργασία της διατριβής αυτής.

Επιπλέον, η εργασία αυτή στοχεύει στην αντιμετώπιση της έλλειψης μεθοδολογιών που να συνδυάζουν άμεσες και έμμεσες τεχνικές απόκτησης δεδομένων ανάδρασης χρηστών (explicit and implicit user feedback acquisition techniques), τόσο από το σενάριο φυσικού εμπορίου (φυσικό κατάστημα) όσο και από το σενάριο ηλεκτρονικού εμπορίου (ηλεκτρονικό κατάστημα), με στόχο την αύξηση της ακρίβειας και της διαθεσιμότητας των συστάσεων στα σενάρια αυτά.

Η διατριβή ορίζει και χρησιμοποιεί την έννοια των UbiCARS και προτείνει το UbiCARS Model-Driven Development Framework που στοχεύει στη μείωση της πολυπλοκότητας υλοποίησης, στην αφαιρετικότητα των τεχνικών λεπτομερειών και στην επίσπευση της υλοποίησης συστημάτων συστάσεων για το εμπόριο. Προτείνεται μια νέα γραφική Γλώσσα Μοντελοποίησης Τομέα (graphical Domain Specific Modelling Language - DSML) για τα UbiCARS, που οδηγεί σε σχεδιασμό UbiCARS βασισμένο σε μοντέλα, σε δυναμική διαμόρφωση των UbiCARS στο φυσικό και ηλεκτρονικό εμπόριο, καθώς και στην ενσωμάτωση των UbiCARS σε προϋπάρχοντα ηλεκτρονικά καταστήματα. Το framework παρέχει στους επαγγελματίες την γραφική γλώσσα και ένα λογισμικό επεξεργασίας μοντέλων (modelling editor) για τον σχεδιασμό UbiCARS εφαρμογών.

Ως δεύτερη συμβολή της διατριβής, το προτεινόμενο framework επιτρέπει στους προγραμματιστές να χρησιμοποιούν κατά τη διαδικασία υπολογισμού των συστάσεων, δεδομένα προερχόμενα από την εφαρμογή άμεσων και έμμεσων τεχνικών απόκτησης δεδομένων ανάδρασης χρηστών, οι οποίες εφαρμόζονται τόσο στο φυσικό κατάστημα όσο

και στο ηλεκτρονικό κατάστημα. Ο συνδυασμός των τεχνικών αυτών αναμένεται να βελτιώσει περαιτέρω την ακρίβεια και τη διαθεσιμότητα των συστάσεων. Το framework αξιολογήθηκε με συμμετοχή προγραμματιστών μέσω μιας έρευνας χρησιμοποιώντας ένα ερωτηματολόγιο προσανατολισμένο σε εργασίες (Task). Τα θετικά αποτελέσματα της αξιολόγησης δείχνουν τις δυνατότητες του framework.

Abstract

Recommender systems have been the answer to the information overload modern life consumers experience for more than two decades. Recommender systems are software tools able to discover the necessary knowledge about users in order to offer them personalised recommendations. Recommendations of products to customers have been proven to boost sales, increase customer satisfaction and improve user experience, making recommender systems an important tool for retail businesses. A key question is how can businesses deploy recommender systems on their on-line retail stores?

Despite the success of recommender systems in industry, limited research has been conducted on the engineering aspects of such complex systems in terms of support during their design and development. The development of effective context-aware recommender systems by non-expert practitioners is not an easy task due to the complexity of building the necessary data models and selecting and configuring recommendation algorithms. The absence of works that use Model-Driven Development methods for the design, development and deployment of Ubiquitous Context-Aware Recommender Systems (UbiCARS) that specifically aim for technical abstraction, reduction of development complexity, and development expedition of the aforementioned systems has motivated the work in this thesis.

Moreover, this work addresses the lack of methodologies that combine explicit and implicit user feedback acquisition techniques from both the physical commerce scenario

(the physical store) and the electronic commerce scenario (the e-store), aiming to enhance recommendation accuracy and availability in these settings.

This thesis defines and uses the concept of UbiCARS and proposes the UbiCARS Model-Driven Development Framework that aims to reduce development complexity, abstract the technical details and expedite the development of recommender systems for commerce. A novel graphical Domain Specific Modelling Language (DSML) for UbiCARS is proposed that drives model-based design and dynamic configuration of such systems for physical and online commerce, as well as system integration with pre-existing e-stores. The framework provides a DSML and a Modelling Editor for practitioners to use to design their UbiCARS applications.

As a second contribution, the framework enables developers in utilising in the recommendation process data from explicit and implicit user feedback acquisition techniques applied in both the physical store and the e-store, a combination which is expected to further improve recommendation accuracy and availability. Accuracy is increased by including additional user feedback data on products in the recommendation computation process. Recommendation availability is increased by enabling the use of additional datasets in the recommendation computation process, to be used where data sparsity or low quality of datasets is observed (e.g. because of the cold start problem). The framework was evaluated with developers via a survey by using a task-oriented questionnaire. The positive results of the evaluation clearly show its potential.

Acknowledgements

I would like to express my indebtedness to my supervisor Professor George A. Papadopoulos for his guidance, support and encouragement in all of the years of my doctoral studies.

I would also like to thank all of my colleagues and friends at the Software Engineering and Internet Technologies Laboratory at the University of Cyprus for their support and assistance towards completing this thesis. Special thanks to Achilleas for his persistent support.

Finally, I would like to thank my wife Stella for her support and patience, as well as my family and friends who supported me through the years in many ways.

Thesis Contributions

This thesis is founded on the knowledge acquired by the author's involvement in the authorship of the following research publications:

Mettouris, C., Achilleos, A.P., Kapitsaki, G.M., Papadopoulos, G.A.: A Model-Driven Development Framework for Automated Design and Development of Ubiquitous Context-Aware Recommender Systems for Commerce. Journal Paper To be Submitted.

Mettouris, C., Achilleos, A.P., Kapitsaki, G.M., Papadopoulos, G.A.: The UbiCARS Model-Driven Framework: Automating Development of Recommender Systems for Commerce. In: Kameas A., Stathis K. (eds) Ambient Intelligence. AmI 2018. Lecture Notes in Computer Science, vol 11249. Springer, Cham, pp. 37-53, (2018), DOI https://doi.org/10.1007/978-3-030-03062-9_3.

Mettouris, C., Papadopoulos, G.A: Ubiquitous recommender systems. Computing, Springer, V. 96 (3), pp. 223-257 (2014), DOI https://doi.org/10.1007/s00607-013-0351-z.

Mettouris, C., Papadopoulos, G.A.: Using Appropriate Context Models for CARS Context Modelling. In Knowledge, Information and Creativity Support Systems, Series Advs in Intelligent Syst., Computing, Springer, Vol. 416, ISBN 978-3-319-27477-5, (2015), DOI https://doi.org/10.1007/978-3-319-27478-2_5.

Mettouris, C., Papadopoulos, G.A.: CARS Context Modelling. In Proceedings of the 9th International Conference on Knowledge, Information and Creativity Support Systems. KICSS 2014. pp. 60-71 (2014), Available: http://kicss2014.cs.ucy.ac.cy/files/KICSS2014Proceedings.pdf.

Mettouris, C., Papadopoulos, G.A.: Contextual Modelling in Context-Aware Recommender Systems: a generic approach. In: Haller, A., Huang, G., Huang, Z., Paik, H.-Y., Sheng, Q.Z. (eds.) WISE 2011 and 2012 Combined Workshops. LNCS, vol. 7652, pp. 41-52. Springer, Heidelberg (2013), DOI https://doi.org/10.1007/978-3-642-38333-5_6.

Mettouris, C., Achilleos, A.P., Papadopoulos, G.A.: A Context Modelling System and Learning Tool for Context-Aware Recommender Systems. In Scaling up Learning for Sustained Impact, D. Hernandez-Leo, T. Ley, R. Klamma, and A. Harrer, Eds. LNCS, Springer Berlin Heidelberg, Volume 8095, pp. 619-620, (2013), DOI https://doi.org/10.1007/978-3-642-40814-4_80.

Table of Contents

Table of Figures

List of Tables

Chapter **1**

Introduction

Recommender Systems (RSs) have been the answer to the information overload modern life consumers experience for more than two decades. RSs are essentially software tools able to discover the necessary knowledge about users in order to offer personalised recommendations to them. The terms "Traditional" or "un-contextual" RSs are used to refer to RSs that use limited or none contextual information to produce recommendations, as opposed to the Context-Aware Recommender Systems (CARS) that aim in using many contextual parameters to increase recommendation accuracy. Recommendation accuracy refers to the degree that the recommender is able to predict the items that a user likes, i.e., those items that the user would have rated the highest, had ratings for that user been available for all products. It has been proven that, in e-commerce settings, recommendations of products to customers boost sales [1, 2, 3], increase customer satisfaction [1] and improve user experience [4, 5]. Therefore, not only customers benefit from recommender systems, but electronic stores as well [2]. A key question is how can businesses deploy RSs on their on-line retail stores?

Many companies (vendors) offer commercial proprietary RSs [6, 7, 8, 9, 10, 11]. Such companies develop and deploy RSs in their business clients' websites or e-stores, aiming to increase sales [12]. In most cases such RSs are offered as a service (Software as a Service - SaaS). Vendor companies will manage themselves all the data needed for the recommendation process (content and context data, user profile information, user behaviour history, etc.), while their business clients are only responsible for displaying the recommendations under vendors' guidance and supervision. Service pricing is based on how successful the service actually is; therefore, vendors rely on their clients' success for revenue [12]. Consequently, vendors not only conduct the initial deployment strategy, but

also do training, optimisation and supervision of the whole recommendation process [12]. Moreover, while recommendation algorithms are key to all RSs, client businesses do not select their vendor based on who provides the best algorithms. Instead, clients are more interested in how the solution will actually affect revenue and costs, and how quickly results can be achieved [12]. Consequently, clients are not really interested in investing in the recommendation algorithms per se, allowing vendors to get away avoiding to share their recommendation algorithms, keeping them thus as their intellectual property. Another important point is that commercial RSs cannot be used for further exploration or extension by businesses themselves, as it is often difficult or even impossible to determine the algorithms they use, how they use them, whether they combine more than one, etc.

On another dimension, while e-commerce sales grow exponentially, reports during the past years have shown that physical shopping is still the main shopping mode in comparison to e-commerce [13, 14]. According to the International Council of Shopping Centres – ICSC [15], e-commerce is complementary to brick-and-mortar retail (i.e., physical retail), while consumers still prefer in-store shopping for reasons, such as to fill an immediate need or want, to "touch and feel" the merchandise, to interact with service professionals, and to have a social experience in the store. In [15], it is reported that 94% of total retail spending takes place in stores (USA report up to 2014). As customers nowadays have the option to select from a huge variety of high quality products at competitive prices, retailers are in need to provide better service to win the trust of customers and maintain a longer customer relationship. Information systems can play a significant role in sensing what customers like and the sales trend, so where information systems are not used and thus information cannot be acquired immediately, customer demands cannot be met in real time, risking their interest in shopping be reduced [16]. As it is noted in [16], with the change of market patterns and customer demand, it is particularly necessary for the retail industry to provide a shopping environment with pleasure, safety, convenience and consideration for customers.

The term ubiquitous computing, first introduced in the nineties, refers to the shifting of the computing paradigm from the desktop PC (Personal Computer) to a more distributed and embedded form of computing [17]. Together with Pervasive Computing (for many the terms are synonymous), ubiquitous computing introduced the concept of *"anywhere, anytime computing"*, allowing users to interact with computers embedded in every-day objects in an "anywhere and anytime" manner. Ubiquitous computing specifies that the interaction of users with such devices must be straightforward to the degree that the user remains unaware of such an interaction. In other words, in order for ubiquitous and

pervasiveness to be achieved, computers must disappear from the front-end, be embedded to common objects that humans use daily and provide computational and informational services without expecting from users to explicitly and consciously interact with them.

Ubiquitous Recommender Systems use ubiquitous devices in order to facilitate the user through her task on-location, by providing her with personalised recommendations. As such, Ubiquitous RSs combine characteristics from both the Ubiquitous domain and the Recommender Systems domain. In [18], we have defined the Ubiquitous Context-Aware Recommender Systems – UbiCARS as systems that, while being categorised under the umbrella of Ubiquitous Recommender Systems in that they are using ubiquitous context sensing and user feedback acquisition methods to enable the provision of recommendations on location via mobile devices, in addition, they also adopt "contextual preference elicitation and estimation" methods used by CARS [19], where modelling of user preferences is conducted by using intelligent context-aware recommendation algorithms to compute personalised recommendations for users.

While RSs for e-commerce track user behaviour by acquiring user feedback data on products online, such as product ratings and purchase history to compute personalised recommendations of products to users, in ubiquitous settings, other context-aware methods are applied to track user interest on products, such as tracking customers' in-store shopping path or the customers' staying time in the product area [13, 20]. The aim is to recommend to users similar products or other related things such as brand stores in a shopping mall or provide reviews and ratings on a product.

Model-Driven Development (MDD) aims at the abstract representation of the knowledge and activities of a particular application domain. Besides abstraction, the second core concept associated with MDD is automation [21]. Application models are defined at an abstracted level and, using automated transformations or interpretations, they are converted into applications, eliminating or minimising the need to write code. MDD provides a clear separation between technology-neutral and technology-specific concerns of a system during the development cycle [22]. It is meant to increase productivity by maximising reuse of standardised models, simplifying design via models and promoting communication between professionals via standardisation of the terminology in the application domain. Moreover, models can be created at different abstraction levels to represent various aspects of the system and are expressed in Domain-Specific Languages (DSL) [23].

1.1 Motivation

While the benefits of using product recommender systems in e-commerce are prominent, it is not clear how businesses can deploy RSs on their e-stores without involving vendor companies, i.e. outsourcing the process. Although open source recommendation frameworks (e.g. EasyRec [24], LensKit [25], LibRec [26], Mahout [27], Duine [28], GraphLab Create [29], MyMediaLite [30], Apache PredictionIO [31]) are available for retail businesses to use as recommendation engines in building their own RS applications, it is nevertheless difficult for practitioners (e-store developers, software engineers) that are not experts in recommender systems to achieve such a task [32, 33, 34, 35]. These frameworks do not offer an abstraction from the technical details concerning the embedding of recommendations to an e-store, requiring from practitioners to work on code level, and furthermore, they are not extensible with new functionalities [33].

Especially, since recent research on recommender systems has proven that Machine Learning (ML) algorithms are the most efficient in providing recommendations, it is even more difficult for non-RSs experts to utilise such algorithms in real applications because of their high complexity. In [34], the authors investigate how ML algorithms in RSs are studied and used, as well as trends in ML algorithm research and development. They discuss that, due to the substantial number of ML algorithms and their variations proposed in the literature, a clear classification scheme for them does not exist, making it even more difficult for practitioners to select the ML algorithm that best fits their needs when developing a RS. As they state, even researchers may find it challenging to track how ML algorithms in RSs are used [34]. The paper presents a review on such algorithms. Further proof that high-level expertise in recommendation algorithms is needed to effectively implement RSs in commerce is the existence of vendors that offer commercial proprietary RSs.

Although a number of works in the literature propose methods to tackle RS development complexity, none of these works manages to adequately abstract the technical details and eliminate the need to write code by developers, such that RSs can be designed, developed and deployed expeditiously by non-RSs experts. The motivation for this research is to use model-driven development methodologies and tools to enable businesses (in particular their developers that have no prior knowledge or experience in the recommender systems domain) to design, develop and deploy state-of-the-art context-aware recommender systems on their electronic and physical stores in reduced time and with reduced development complexity.

This research is also motivated by the recent technological developments in ubiquitous computing, that suggest that the success of RSs in the virtual world can be replicated in the real world, i.e., in physical retail settings [2]. There is already a trend towards personalisation in the physical retail market and RSs, together with ubiquitous computing, can be the means to make it more sophisticated and effective [2]. Using the notion of the Ubiquitous Context-Aware Recommender Systems – UbiCARS [18] that combine ubiquitous context and user feedback acquisition methods with intelligent context-aware recommendation techniques, this work aims to enable the usage of user feedback acquisition techniques from the ubiquitous scenario of the physical retail store, and together with user feedback acquisition techniques from the online scenario of the electronic store, to increase the available user-product interaction data upon which context-aware recommendation algorithms function, aiming to further improve recommendation accuracy and recommendation availability. To the best of our knowledge, none of the works in the literature uses a combination of user-item interaction data from both the physical scenario and the online scenario with the aim to enhance recommendation accuracy and availability in both settings. We describe the latter through an example of use.

UbiCARS Example of Use

We assume an electronics store (named SmarTech) with an e-commerce website and a number of physical showrooms with a large variety of electronic devices and peripherals. Nicolas is a regular user of the SmarTech e-store; he has purchased a few products in the past and also rated some of them online. When Nicolas uses the SmarTech e-store, he likes the personalised product recommendations the website offers, as most of the times the recommended products match his preferences, so he checks them out. Nicolas is now at a SmarTech physical store showroom browsing the products. He is quite interested in technology products and, in particular, in high-end laptops. SmarTech only includes a limited number of products in its showroom, compared to those listed in the online catalogue. Nicolas approaches a laptop on a shelf that has caught his attention. As he reads a few specifications on the laptop from the small label, he thinks he would like to know more information on its features, as well as to find more products he might be interested in. While in the store, Nicolas receives a number of product recommendations on his smartphone, similar to those he receives while browsing the SmarTech e-store. Two of the recommended products are currently in the showroom, so he checks them out next. Another recommended product is located in another SmarTech showroom downtown;

Nicolas thinks he can probably stop by the showroom tomorrow afternoon after work, as he likes to physically check out the devices before buying. The fourth recommendation is for a product that is available only through the SmarTech e-store; Nicolas can check that product from the convenience of his home PC. Nicolas finds it informative and entertaining to be able to receive recommendations while in the store.

Besides tracking Nicolas' virtual interaction with products through the SmarTech e-store (e.g. his ratings on products) to offer him personalised recommendations, SmarTech has also tracked his physical interaction with the products in the showroom using innovative ubiquitous technologies. In particular, SmarTech is aware of the products Nicolas has interacted with during his stay at the showroom and uses this information as (implicit) product preference indication for Nicolas. Thus, recommendations for Nicolas the next time he will visit one of the SmarTech physical stores or the SmarTech e-store can be more *accurate*, since more information relevant to Nicolas' preferences will be used during their computation. Moreover, the usage of user behaviour tracking techniques from the physical scenario in the recommendation process as alternative to a number of corresponding techniques from the online scenario increases recommendation *availability*, as recommendation computation can be conducted using data from any of these techniques.

1.2 Contributions

In this research, we apply the Model-Driven Development (MDD) paradigm on the physical and electronic commerce recommendation domain and propose a novel UbiCARS (Ubiquitous Context-Aware Recommender Systems) MDD Framework [36] that aims:

i. To reduce development complexity, abstract the technical details and expedite the design, development and deployment of state-of-the-art context-aware RSs in physical commerce environments (physical retail stores) and electronic commerce environments (e-stores) by developers/practitioners that are not recommender system experts.

ii. To enable these developers/practitioners to acquire ubiquitous user feedback data from tracking user-product interaction in the physical store (ubiquitous scenario), and use this information in the recommendation process, together with user

feedback data acquired from tracking user-product interaction from the e-store (online scenario), a combination which is expected to further improve recommendation accuracy and availability.

A novel graphical Domain Specific Modelling Language (DSML) for UbiCARS is proposed [36] that drives model-based design and dynamic configuration of such systems for physical and online commerce, as well as system integration with pre-existing e-stores. To the best of our knowledge, a DSML for UbiCARS does not exist.

This work does not claim to advance research in terms of defining novel explicit and implicit user feedback acquisition techniques in physical and online commerce settings. The novelty of this work lies in the amalgamation of already existing techniques through a model-driven development framework, via which these techniques are offered to developers to design, develop and deploy UbiCARS with reduced development complexity, by abstracting technical and domain specific details, and by expediting the development and application of UbiCARS, while aiming also to increase the accuracy and availability of recommendations.

Based on the above, the following research hypotheses are presented:

Hypothesis 1: The MDD methodology proposed and realised by the UbiCARS MDD Framework can assist developers in building state-of-the-art Ubiquitous Context-Aware Recommender Systems for commerce by reducing the complexity of development and expediting the development without the need for RS expertise.

Hypothesis 2: Using the UbiCARS MDD Framework can enhance the recommendation accuracy of the developed Ubiquitous Context-Aware Recommender Systems, by tracking and utilising user behaviour in both online and physical settings.

Hypothesis 3: Using the UbiCARS MDD Framework can increase the recommendations availability for Ubiquitous Context-Aware Recommender Systems, by tracking and utilising the user behaviour in both online and physical settings.

Research Questions

Based on the above research hypothesis the following research questions are defined in this work:

RQ1: Does the UbiCARS MDD Framework reduce the development time (expedite the development) of state-of-the-art Ubiquitous Context-Aware Recommender Systems for commerce for developers that are non-experts in RS?

RQ2: Does the UbiCARS MDD Framework reduce development complexity in developing state-of-the-art Ubiquitous Context-Aware Recommender Systems for commerce for developers that are non-experts in RS by:

1. Reducing the lines of code and database (DB) queries developers need to write?
2. Supporting software maintainability[1] and software evolution[2] in terms of future enhancements?

RQ3a: Does the UbiCARS MDD Framework enhance recommendation accuracy when used for the development of Ubiquitous Context-Aware Recommender Systems for commerce by tracking and utilising the user behaviour in both physical and online settings?

RQ3b: Does the UbiCARS MDD Framework increase the availability of recommendations when used for the development of Ubiquitous Context-Aware Recommender Systems for commerce by tracking and utilising the user behaviour in both physical and online settings?

1.3 Research Methodology

The research methodology we have followed is outlined next.

Firstly, the UbiCARS MDD Framework is designed and developed based on the requirements gathered from long-term and extensive research on the Ubiquitous and

[1] Software maintainability is defined as the degree to which an application is understood, repaired, or enhanced.
[2] Software evolution in software engineering refers to the process of developing software initially, then repeatedly updating it for various reasons

Context-Aware Recommender Systems domains, as well as from lessons learned from earlier works of the author [37, 38, 39, 40, 18]. The new graphical Domain Specific Modelling Language for UbiCARS delivered reflects these requirements and is evaluated based on how well it can drive model-based design and dynamic configuration of UbiCARS for physical and online commerce, including system integration with pre-existing e-stores.

Secondly, the UbiCARS MDD Framework was thoroughly evaluated by a group of users that belong to the target group of the framework. The primary target group of the framework constitute e-store developers and practitioners with no expertise in RSs. As secondary target group of the framework we define two other groups of users: recommender system experts and model-driven development practitioners/experts. The aim of involving the secondary target groups is to acquire expert evaluations for the two main areas of the framework, mainly the Recommender Systems domain and the MDD domain. Furthermore, involving in the evaluation professionals and experts from different domains reduces the risk of subjective judgement.

The evaluation of the framework was conducted via the survey method by using a task-oriented questionnaire. Through the questionnaire, participants were given a set of tasks to complete using the framework and, at the same time, they were asked to report their findings and respond to various questions about their experience with it. The evaluation was conducted with one participant at a time, while the author, as the responsible researcher, was observing the entire evaluation process with each participant.

The results of the evaluation were analysed using quantitative data analysis and descriptive statistics. Quantitative data analysis refers to the calculation of frequencies of variables and differences between variables with the aim to find evidence to either support or reject the hypotheses, as well as to answer the research questions we have formulated. The selection of quantitative data analysis over qualitative stems from that our evaluation uses data in the form of metrics (e.g. percentage of users that have completed a task, or the average time users took to complete a task), which constitute quantitative data.

This thesis is organised as follows.

Chapter 2 provides the background areas related with this thesis, namely Ubiquitous Computing and Recommender Systems. In terms of recommender systems, the chapter also introduces Context-Aware Recommender Systems (CARS) in Section 2.3, discusses Machine Learning approaches in recommender systems in Section 2.4, and presents certain aspects of research on User Experience on recommender systems in Section 2.5. In Section 2.6, other aspects of recommender systems are presented, that, although not at the centre of the work of this thesis, they are referred to in the thesis. The last section of Chapter 2, Section 2.7, introduces Model-Driven Development and briefly discusses literature works that use MDD in various domains.

Chapter 3 presents related work. The first two sections discuss explicit and implicit user feedback acquisition techniques for the online scenario and the physical scenario. The UbiCARS Framework adopts such techniques, includes them in its MDD methodology and offers them to developers with a high level of abstraction. Section 3.3 lists the available recommendation frameworks and engines, while Section 3.4 discusses literature works on using modelling and other software engineering techniques to tackle RS development complexity. To the best of our knowledge, the systems presented in this section constitute the most relevant systems to our work found in the literature. The chapter closes with a discussion on early works on context modelling the author has proposed, that acted as previous knowledge in the conceptualisation of the work presented in this thesis. In particular, in these works we have used modelling techniques and e-learning methodologies to provide tools for developers to model the context in the application layer of context-aware recommender systems that also acted as a learning tool for context related concepts of such systems. The section describes these works, their strong points, as well as their limitations that eventually led to the conceptual birth of the work presented in this thesis.

Chapter 4 describes Ubiquitous Recommender Systems and introduces the Ubiquitous Context-Aware Recommender Systems – UbiCARS and the challenges in designing such systems. UbiCARS constitute a principal concept in this work, and as such, they are discussed in a dedicated chapter.

Chapter 5 presents the UbiCARS MDD Framework. Section 5.1 describes the multi-layered software architecture of the UbiCARS MDD Framework, emphasising on the modelling layer and the configuration layer. Section 5.2 describes the UbiCARS Domain Specific Modelling Language. A dedicated section is given to the ubiquitous concepts of the DSML in Section 5.3, where UbiCARS app (mobile application) specific elements and

technology usage are described in detail. Section 5.4 discusses how the UbiCARS methodology enhances recommendation accuracy and availability. This is mainly done by discussing in detail the example of use presented in Section 1.1 "Motivation". Chapter **5** concludes by presenting the UbiCARS Framework architecture. Besides discussing the architecture, Section 5.5 presents activity diagrams for two user roles: the developer and the user/customer, as well as the configuration steps followed by the framework from start (model parsing) to finish (recommendations presentation).

Chapter 6 titled "UbiCARS Framework Demonstrator & Evaluation" is divided into two sections. Section 6.1 presents the UbiCARS modelling editor with its tools, as well as an instance model. This section also describes system implementation and set-up of example e-stores for testing and evaluation purposes. Section 6.2 describes evaluation methodology and results. In Section 6.2.2, a summary of evaluation results presents the most important outcomes of the framework evaluation. In Section 6.2.3, a detailed presentation of the results is provided. The UbiCARS MDD Framework was evaluated mainly by conducting evaluation sessions with 20 participants for a total of 19.5 hours. The survey method was used via a task-oriented questionnaire and by involving one participant at a time, with the author observing the entire evaluation process with each participant. After these 20 sessions were completed, the framework's learnability was evaluated through the "Framework revisit sessions" with two participants (for details see Appendix A). In parallel, the framework was evaluated by three more participants remotely, as discussed in Appendix B. In Section 6.3 we answer to the research questions established in Section 1.2.

Chapter 7 concludes the thesis through a discussion of the contributions of the thesis, and defining future directions for this research.

Chapter **2**

Background

2.1 Ubiquitous Computing

2.1.1 Definition

Ubiquitous computing focuses on promoting computing beyond the notion of the PC. It deals with integrating technology in everyday objects aiming to provide people with technological means that ease everyday life. The most inspiring work among all research works in ubiquitous computing comes from Weiser [41], in which he describes his ubiquitous vision. Mark Weiser, twenty seven years ago, envisioned a world in the future in which easy to use, context-aware, pervasive electronic devices would be connected to one another and seamlessly embedded to the environment. The words *"pervasive"* and *"context-aware"* are the most suitable ones to describe this vision. Context refers to any information related to the user, the computing system, the environment of the user and the interaction of the user with the system [42]. As depicted in his scenarios in [41], Weiser wanted those devices to be: (i) pervasive to the degree that people would not be aware that they are using them (easy to use, straight forward human-device interaction), and (ii) context-aware to the degree that devices would be aware of information regarding users (e.g. their characteristics, profile, background, social status, and their relations to each other), as well as other people in the proximity, various devices around, location information, etc. In his scenario in [41], Weiser proposes displaying information concerning other people located inside or outside the house on a window glass, something that is challenging even in our times, almost thirty years later. Weiser's vision was adopted by the research community gaining hundreds of citations. Most of these works struggled to realise this vision of drawing away from the traditional desktop PC by implementing novel

ubiquitous applications, by using mobile devices and networking technologies such as RFID, Wi-Fi and Bluetooth, by using sensors, wearable devices, public displays and more. The ultimate goal has always been to achieve context-awareness and pervasiveness to the level of extent of Weiser's thoughts.

However, as much as we would like to believe that technology has progressed regarding pervasiveness, the vision of Weiser as stated by him has not yet come to life. Even though devices and technologies similar to the ones foretold by him have already been developed and used, they are still not pervasive enough to be seamlessly embedded to the environment: *"...'Ubiquitous computing' ... does not just mean computers that can be carried to the beach, jungle or airport. Even the most powerful notebook computer, with access to a worldwide information network, still focuses attention on a single box"* [41]. Literally every mobile device or computer people have built and used until now is a "computer that can be carried to the beach, jungle or airport", provided that there is a wireless network coverage at that place. We have successfully built devices that extend the notion of the desktop computer anyplace where networking availability exists, having "anywhere and anytime" access to information which can be offered in a personalised manner, but we have not succeeded in the two fundamental rules of Weiser: (i) to make these devices seamlessly embedded to the environment, and (ii) to make the devices more context-aware than the basic information gathered from a user's profile or a few sensors. Waller and Johnston [43] note that, because ubiquitous computing is mainly application driven utilising technologies such as RFID, sensors and wearable devices, it endorses the risk of focusing on technical capabilities and challenges, ignoring at the same time Weiser's vision. They state that the common ubiquitous devices we use tend to get in the way of what we want to do and that they interact with a *representation* of the real world instead of the real world per se. This is true if we consider that most tasks we are able to do via a computing device have a meaning only to other computing devices or, to other people, only if they are also using a computing device themselves. For example, when a person sends an email to a colleague she actually does not inform her colleague about the matter (no action occurs in the real world towards this direction); instead, her action reflects on her colleague's computer or smartphone, which will keep the information for itself until the colleague checks her emails. In contrast to the aforementioned, Weiser's devices could automatically project information relevant to the context at the time needed and in a way that users did not have to interact at all with any strange, complex and not human friendly device. In fact, Weiser's scenarios minimise the human-computer

interaction to a minimum, enabling at the same time the user to act on real objects instead of acting on representations of them [41].

The discussion above results in that, in order to achieve pervasiveness, we should move the user away from being a self-conscious user to being a more abstracted recipient of information. By that it is meant that the user should not be bound to use any device that would make her conscious about using it in order to be informed. At this point a question arises: What are the challenges that researchers face in their attempt to realise Weiser's vision? Are there only technological challenges or do other kinds of challenges exist as well? We discuss this issue in the following section.

2.1.2 Ubiquitous Computing Challenges

A number of ubiquitous computing challenges exist, ranging from technological such as wireless technology limitations and power management issues, to challenges related to context-awareness, tracking user intentions and privacy concerns. In the following we list and categorise the most important challenges discussed in the bibliography.

General Challenges

Want and Pering [17], categorise the challenges in ubiquitous computing to: (i) power management issues - how mobile devices deal with processing power and storage space and the kind of wireless technology to use in every given situation, (ii) limitations in connecting devices – how are all these small devices going to be connected and managed, (iii) user interface issues – since ubiquitous computing demands for many different small-scale devices of various types of interfaces and displays of various sizes, the challenge lies in developing user friendly interfaces (e.g. many functionalities of the PDA are not used because of the complexity of the interface: too many functionalities provided in too little space to display them), (iv) issues related to Location-Aware Computing – location-based APIs must be built to be used by a wide variety of devices, along with a variety of wireless technologies.

Henricksen et al. [44] add to the above list the challenge of managing heterogeneous devices of different hardware and software specifications, such as sensors and actuators, embedded devices in objects (e.g. shoes), home and office appliances (e.g. videos), mobile devices and traditional desktop computers, in order for these devices to interact seamlessly. Another challenge they mention has to do with maintaining network connections while devices move between networks of different nature and characteristics. During this setting,

network disconnections have to be managed in a way that is abstracted from the user, i.e. the user must have the feeling that he is continuously connected to the network despite any movement on his behalf. Besides connectivity problems, user mobility also demands for software and data mobility as well. In ubiquitous environments, people tend to use many devices simultaneously, therefore there is a need for these devices to communicate and exchange data. As Henricksen notes, program and data migration, as well as synchronisation and coordination of components should not concern the developers; rather, a ubiquitous computing infrastructure facilitating interoperability is needed [44].

Davies and Gellersen [45] add that, since ubiquitous systems operate independently, each in its own context, the challenge is how to build integrated ubiquitous computing systems that will be able to easily communicate and decide together based on an integrated, common context.

Tracking User Intentions

Satyanarayanan [46] notes that tracking user intentions is important in Pervasive Computing in order for the system to understand what system actions could help the user and not hinder her. The author uses an example to make a point: suppose a user who is viewing a video over a network connection that suddenly drops; what should the system do: (i) reduce the fidelity of the video, (ii) pause briefly to find another higher-bandwidth connection, or (iii) advise the user that the task can no longer be accomplished? The correct choice will depend on what the user is trying to accomplish. According to [46], current applications fail to correctly track the intentions of the user, or they do not consider them at all.

Davies and Gellersen [45] also categorise the process of tracking user intentions as challenging, as well as determining the user's task accurately and react based on them in order to assist the user. According to the authors, tracking user intentions has only been achieved in extremely limited application domains.

Context-Awareness and Adaptation

An important challenge in context-awareness is to build context-aware systems that detect and manipulate the context in a human-like manner, i.e. making decisions proactively based on the context and provoke actions based on those decisions that assist the user through her task; the aforementioned should be done without any user participation or disturbance, except maybe in case of emergency [46]. Another important issue is how to

obtain contextual information. Contextual information can be any information related to the user, the computing system, the environment of the user and any other relevant information regarding the interaction of the user and the system [42]. The user's personal computing space [46] can be used as the user's context (any information regarding the user taken from her personal profile, calendars, to-do lists, etc.), various types of context can be sensed in real time like location, people and objects nearby, while contextual parameters could incorporate the current emotional and physiological state of the user as well [46]. Contextual challenges also include the way context is represented (ontologies can be used or other context modelling techniques), the way context information is combined with system information, as well as how frequently should context information be considered.

Hinze and Buchanan [47] differentiate static context (e.g. user's profile information) from fluent context (dynamic, real-time context, e.g. time) and propose that a context model should be defined for each important entity, such as the user, the locations, etc. The authors mention as challenges the capturing of the context (should it be done automatically at particular times or manually by the user?) and the process of storing the context (should it be stored on the client, on the server or on both?).

On the process of accessing contextual information, Hinze and Buchanan propose that context-awareness can help in reducing the amount of data to be accessed in real time, by pre-retrieving any relevant pre-known data, e.g. the static context [47]. This increases efficiency. In addition, the environment and the services it can offer are also very important, as different environments support different services, as well as different contextual parameters [46]. A question that arises is: what are the minimal services that an environment needs to provide to make context-awareness feasible?

The process with which a system adapts its behaviour to the needs and preferences of its users is called adaptation. Adaptation is based on user related and context related information. Any components/devices should adapt based on the context without interfering with user's task: no user explicit interaction should be necessary. These components/devices should adapt separately and at the same time, while the user maintains a consistent view of the system/application. In ubiquitous computing, the need for adaptation often stems from poor resource availability when such resources are in demand, e.g. poor bandwidth available to a user that desperately needs email access before boarding on a flight. Based on the aforementioned, three adaptation strategies can be mentioned [46]: (i) the system guides the adaptation of applications towards using less of a resource that is scarce, (ii) the system asks from the environment to guarantee a minimum level of resource availability, aiming to meet the client's demand for that resource and (iii) the

system recommends to the user certain actions that the user can do to optimise system behaviour. Research issues in adaptation include the correct choice between the various adaptation strategies for a particular application, which factors should the ideal adaptation strategy consider, how these factors should be weighted, whether the user should play any role in this process and how should seamless transition between strategies happen [46].

An important challenge regarding context-aware and adaptation strategies in ubiquitous systems being used by non-expert individuals has to do with how to prevent feelings of anxiety and frustration that such users may feel due to sudden system changes. Kjeldskov and Skov [48] have extended and tested a ubiquitous system for the healthcare domain and concluded that users (nurses not familiarised with context-aware computing systems) demand for less automatic changes in system behaviour (even though such changes better adjust the system based on the environment), since such sudden changes seem inscrutable to them: even the smallest automatic information update could provoke unhappy feelings to them if the update is done without the user knowing what is happening, understands it, approves it and has at all times the ability to disable it.

Proactivity and Transparency

A big issue in ubiquitous systems is the trade-off between proactivity and transparency. If a system is proactive, then in many situations it will act proactively based on internal rules and procedures and unless carefully designed, it can easily annoy a user, reducing therefore its transparency levels [46]. A question that arises is how proactive should a system be, at what circumstances, and what is the right balance between proactivity and transparency? User preferences can play an important role in this decision, as well as user experience with the system, the process, the application domain and the environment, since an experienced or expert user may have less patience and tolerance for proactivity and may therefore expect more transparency from the system; the system can then act less proactively than with other inexperienced users.

Changes in User Roles

In ubiquitous settings a user often changes roles according to the context and the current environment she acts within: one challenge is how to capture these changes and how to react on them. As an example, consider a user that is new in town, likes foreign cuisine and on Saturday nights she likes visiting good restaurants in the proximity with the help of her mobile device. However, one Saturday night she has to work for a few hours, therefore she

seeks for easy to prepare, fast solutions for her dinner. The challenge lies in capturing and utilising both personal context (she likes foreign cuisine, it is Saturday night) and business context (she has to work this Saturday) to infer changes in the user's role and react accordingly [47]. Therefore, to successfully capture changes in the user's role, both capturing the current context (i.e. the environment and the various circumstances, events and facts) and user modelling (what possible role could a person play according to context) are necessary.

Location Issues and Mobile Privacy Concerns

Location, as an important contextual parameter, plays an important role in ubiquitous systems. The type of location sensing technology to be used is one issue, privacy is another - user privacy should not be sacrificed for location awareness - while a third issue is the semantic (and contextual) representation of the location, in order to utilise more contextual parameters than just the location itself. For example, by semantically representing locations, one can attach to them various information resources such as a webpage, a user profile, various objects with semantic representation, etc. Location-based annotations e.g. on Social Networks, constitute an example of information resources attached to real locations. Annotations could be information of any format (text, pictures, video, audio, etc.). Ubiquitous annotation systems allow their users to attach such information resources not only to locations, but also to objects and people.

Schilit et al. [49], propose the movement from the simplified concept of location to more contextually rich notions of place where people and activities should also be considered. Possible problems towards this concept include the difficult management of large scale positioning data, privacy concerns regarding location-awareness and the challenge of how to associate information resources with a real-world location. According to the authors, privacy issues regarding location-awareness are related to human psychology: users often consider privacy issues when their location is to be known by a system, but at the same time they provide private information such as credit card numbers and addresses to online systems without hesitation. This happens because, in the first case, they simply do not see the benefit of providing their location to be used by a simple application (e.g. finding friends in the proximity), while at the latter case, they clearly see the benefit of buying goods online. The authors also argue that the centralised nature of most location tracking applications (having a central server on which all user personal data are stored) discourages users from providing any personalised information, because centralised data can be accessed by anyone, not only illegally (e.g. hackers), but also by the

government, corporations with interest in user data (e.g. advertisers), etc. A solution is a decentralised schema where any personal data are stored and computation takes place only on the client side (the user's device). An example of such a technology is the well-known Global Positioning System - GPS: the client device uses satellite links to calculate locally the user's current position.

A qualitative study on mobile privacy by Mancini et al. [50] revealed a number of privacy related user boundaries while users were mobile. These boundaries were related to socio-cultural factors rather than physical ones, such as a user's personal boundaries regarding the personal information that should be revealed to the public and to members of their social networks, or boundaries relevant to the physical proximity of people that are not members of their social networks. It is important to investigate mobile privacy issues in terms of when do they occur, how do users feel when such issues arise and how does this affect their behaviour. Mancini et al. [50] argue that privacy issues are sensitive, difficult to study and poorly understood, that survey methods on privacy such as questionnaires and interviews provide only limited insight into users' feelings and needs and that, instead, observing user behaviour in real-time should be preferred. However, in situations where users are mobile, directly observing them may lead to undesirable modification of what would otherwise be spontaneous behaviour: *"any observing agent that was following the participants around would hardly go unnoticed and would therefore end up intruding into their privacy and altering their behaviour"* [50]. Therefore, in order to obtain meaningful and useful information regarding user privacy in mobile settings, experience sampling is normally used: participants answer questions regularly or on particular events that concern their feelings and behaviour in daily life situations. Since this method demands from mobile users to spend much time in answering questions at inconvenient times and environments, the authors propose a combination of experience sampling and semi-structured interviewing, during which the user is given a memory phrase in order to better recall certain past events and their context and, by that, assist her to answer privacy related questions relevant to the event [50].

Authenticity of Information and Trust

Another trust related issue met in ubiquitous systems is researched by Lenders et al. in [51]: in a system where users provide their own content from particular locations, how can one trust the authenticity and quality of the information published by individuals? They state that, if spatial and temporal knowledge regarding user content is known (from where was the content provided and when), then this content is more trusted and valuable, and

they propose an approach where user device location and content creation time is used in a user authentication process to ensure the authenticity and quality of the content.

Scalability

Domnitcheva [52] notes that scalability is also an important challenge in location-aware systems. Such systems should be able to cope, on the one end with many sensors providing raw data and, on the other end, with many devices consuming information. The number of sensors and devices must not be fixed; rather, truly ubiquitous systems must be able to facilitate an arbitrary number of them without compromising efficiency or security.

This section described the most important challenges and issues in ubiquitous computing research. Table 1 summarises these challenges by categorising them accordingly.

Table 1: Challenges in Ubiquitous Computing.

Technological
Power (energy concerns) and storage issues due to the small size of mobile devices.
Wireless technologies issues.
Connectivity issues: how to connect and manage many small mobile devices.
Networking issues: manage resources, scalability, devices must operate across different types of networks.
HCI related
User interfaces in small devices.
Devices should be able to function without user action or attention.
User friendly devices/interfaces: any user should be able to use them.
Aim to improve user satisfaction.
Usability: new novel types of interaction.
Any system adaptation should not interfere with the user's task.
Track user intentions: the system should be able to infer what the user wants to do in order to assist her.
The amount of input that is needed from users should be reduced by using context–awareness.
Proactivity vs. Transparency: a proactive system knows how to react to an event. Risk: may not be transparent to user.
Presentation: how will annotations be presented - on the object or not, attached or detached from the location.

Anchoring: how annotations are being attached to places/objects.
Authoring/editing of annotations: how, when, why?
Wearable computing: may be proved inconvenient for users.
Context-Awareness and Adaptation related
Modelling the context: which method is more appropriate to use.
Observing the context: automatically or manually?
Context sensing: retrieving context data from various sources (e.g. sensors), data inconsistencies may occur.
Accuracy of contextual information should be well known during the design of ubiquitous systems.
Storing the context: on server (privacy issues), on client or on both?
Systems should be more context-aware than just the location. A place is more than a location.
How will the application modify its behaviour (be adapted) based on the context.
Devices should not operate based only on their own context, but based on the context of the whole system.
Context-awareness should be used to reduce the amount of input that is needed from users.
Capture changes in the user's role by (i) capturing the current context, (ii) user modelling.
Components adapt based on context without interfering with user's task; user maintains a consistent view of system.
Privacy and Trust related
Users do not want to give up their location privacy, unless for a significant reason: no motive.
Centralised location tracking is not as easily trusted by users as client side location tracking is (e.g. GPS).
Sensitive user data should be distributed only with user's consent.
Due to the wireless networks, ubiquitous applications are difficult to be trusted.
User boundaries regarding privacy are often related to socio-cultural factors rather than physical ones.
Mobile privacy issues are difficult to investigate: need to observe user mobile behaviour in real-time.

The UbiCARS MDD Framework addresses the following of the abovementioned challenges:

- HCI Related: Devices should be able to function without user action or attention: particular attention has been given to utilise user feedback acquisition techniques that do not interrupt users' task while shopping in both physical and online scenarios. Staying time if front of products does not require any form of user action or attention, as opposed to NFC scanning (see Section 5.1 for both methods). Moreover, tracking browsing history and purchase history of users is also done without the need for user action.
- HCI Related: Reducing the amount of input needed from users: the framework utilises implicit user feedback acquisition techniques (tracking users' staying time in front of products, tracking online browsing history and online purchases of users – see Section 5.1) to infer user preferences on products. As these techniques track user behaviour instead of demanding user input, they eliminate any cognitive effort needed from users to explicitly state their preferences on products, e.g. through ratings.
- Modelling the context: The framework, through the DSML, provides to developers context-related model elements to capture the context of users while interacting with products. More on context-related model elements can be found in Section 5.2.2.
- Context-awareness & Context sensing: The framework provides the necessary infrastructure to store context information sensed from various sources, provided that context-related model elements have been defined during the modelling process. Developers can define context parameters of any type; e.g. in terms of the physical store, location as context can be defined in terms of GPS coordinates, or in terms of custom defined attributes such as "store electronics section" or "basement" (see Section 5.2.2).

2.2 Recommender Systems

Recommender systems, as the name implies, are systems that use a variety of filtering techniques and recommendation methods to provide personalised recommendations to their users. RSs are important due to the information overload modern life experiences at

all fields, and their importance will be further increased as this overload is expanding exponentially. The terms “traditional” or “un-contextual” RSs are used to denote RSs that use limited or none contextual information to produce recommendations, as opposed to the Context-Aware Recommender Systems (CARS) that aim in using many contextual parameters to provide better recommendations [19].

Traditional RSs use information retrieved from the user profile, from user’s usage history, as well as information related to the items to be recommended in order to calculate the recommendations. The user profile includes valuable information regarding the user and her personality, her preferences, habits and more. Such information could be used by recommender systems to filter out any recommendations not suitable for the user. User’s behavioural data are data produced based on user actions (behaviour). User actions reflect user preferences and needs, thus analysing such actions during the recommendation computation process is valuable. For example, e-commerce RSs may use any browser history information and logging data to opine what the user likes. Finally, information related to the items to be recommended is used. Such information depends on the particular items that are recommended. For example, in movie recommendations where the items under study are movies, such information may include the title of the movie, its genre, its duration, the actors etc.

The most well-known recommendation approaches are the Collaborative Filtering (CF), the Content-based filtering and Hybrid recommendation techniques.

2.2.1 Collaborative Filtering

The Collaborative Filtering technique recommends items that *similar users to the active user* have highly rated (hence like). There are two types of methods followed in CF, the *Neighbourhood methods* that use similarity functions (Pearson Correlation or Cosine Distance) to compute the user’s neighbourhood, and the *Model-based methods* that use user feedback on items (i.e. ratings) to learn a model for the user that is then used for computing recommendations [53].

Neighbourhood Methods

The Collaborative Filtering technique is composed of two basic steps: the neighbourhood formation and the recommendation extraction [54]. During neighbourhood formation, the similarity between users is computed. For each user, a list of similar users is computed. To compute user similarities, proximity measures are used [53, 54]. Proximity measures

measure the distance between users: users that are closer have more similar preferences. The proximity measure is usually calculated using the Correlation Measure (for users a and b, the distance among them is measured by calculating the Pearson correlation) or the Cosine-based approach (users a and b are perceived as vectors in the m dimensional item space - proximity is calculated by measuring the cosine of the angle between the two vectors) [53, 54]. After finding similar users, the neighbourhood formation can follow the Centre-based schema or the Aggregate one [54]. The Centre-based schema gives higher priority to those users that are closest (more similar) to the active user. The Aggregate neighbourhood schema on the other hand gives higher priority to the users that are closer to the centre of the current neighbourhood (schematically the neighbourhood changes form each time a user is being added). The second step, the recommendation extraction, extracts the top-N recommendations which are the items that have the highest scores among the list of neighbours of the active user.

The assumption made in CF is that those who agreed in the past tend to agree also in the future. Thus, since the active user's neighbours tend to agree with her (they are similar users), the items these neighbours like the most might be included highly in the list of preferences of the active user as well, and hence can be recommended to her. The biggest advantage of Collaborative Filtering is that it does not depend on any system representations of the items to be recommended and thus can function well with complex items such as music and movies [55].

Model-Based Methods

More recently, model-based approaches - or latent factor models - use machine learning techniques such as Matrix Factorisation to decompose the multidimensional user-item-context matrix to learn latent factors for each user and item in the data [56, 57]. Thus, the latent factor models approach tries to estimate ratings by characterising both items and users on factors (e.g. 20 to 100) inferred from the ratings patterns [4]. The underlying assumption here is that both users and items can be modelled by a reduced number of factors [56]. It has been shown that Matrix Factorisation models are superior to neighbourhood methods for producing product recommendations, as they allow the incorporation of additional information, such as implicit feedback, to be utilised in the recommendation process [4].

Collaborative Filtering Challenges & Issues

A number of CF issues are reported in the bibliography [53, 54, 55]. The most well-known problem is the *"new user"* problem, which states that a new user has to rate a certain amount of items before the system may effectively apply the algorithm. This is true if we consider that neighbourhood formation, which finds user similarities, is based on previous user ratings. Therefore, a new user with no previous ratings is left with no recommendations. Moreover, the *"new item"* problem is also very common where the recommender cannot recommend a new item until it has been rated by a number of users. The "new user" and "new item" problems are also known as the *"cold start"* problem of recommender systems, which states that, for RSs to be able to produce meaningful recommendations, user interaction with items first needs to take place.

Another problem related to the user-item matrix (the n × m rating matrix consisting of n users and m items that is used as input in the algorithm) is *sparsity*: if very few people rate some items, then even if these ratings were high, those items will rarely be recommended. This is because for such an item to be recommended, at least one of these few users must be in the neighbourhood of the active user. Thus, the fewer the users, the more unlikely it is to be included in the neighbourhood of the active user. CF also performs poor for users with unusual taste, because, for such users, not many similar users will exist, thus their neighbourhood is expected to be relatively small [53, 55]. Finally, in occasions where different product names refer to the same or similar items, CF cannot associate such products and therefore it treats them differently [54].

2.2.2 Content-Based Filtering

Content-based filtering suggests that the active user will be recommended with those items that are most similar to the items she has rated highly (or bought the most) in the past. Items are similar in terms of their content (features and attributes of the items are used). Some of the most important Content-based issues are [53, 55]: items suffer from over-specialisation, since the algorithm focuses only on items already rated (or purchased) by a user as well as other similar to those, excluding different types of items. The "new user" problem also applies here, where a new user has to rate a certain amount of items before the system can apply the algorithm. This happens because in order for the system to learn any user preferences, the user must rate (or buy) a number of items. Moreover, Content-based filtering is limited to use only features that are explicitly associated with the items to be recommended. Another problem is that too similar items should often not be

recommended (e.g. 2 articles in different newspapers but for the same subject): Content-based filtering cannot distinguish such occasions. Finally, it is difficult to recommend items that were not manually annotated with information (e.g. multimedia items).

2.2.3 Hybrid and Other Recommenders

Hybrid recommenders constitute a combination of CF and Content-based recommenders. The combination can exist either within a system, i.e. one system uses a combination of CF and content-based methods, or by using two separate systems, i.e. a CF recommender and a Content-based recommender accordingly [53]. In the latter case, one can either combine the output of the two systems in one common recommendation list, or choose which of the two recommendation sets is going to be displayed according to appropriate metrics.

Besides CF, Content-based filtering and Hybrid recommenders, other methods have been proposed in the bibliography such as Demographic recommender systems, Utility-based recommenders and Knowledge-based recommenders [55]. For a review of these approaches, as well as a more extended review on the three methods described in this section, the reader is referred to [54, 55, 58, 59].

2.3 Context-Awareness in Recommender Systems

Recommender systems have attracted the research community's interest for the past fifteen years. Many techniques have been proposed, as well as many extensions and improvements, but it was not until recently that the research community realised that recommenders have only been using a part of the available information for producing recommendations. The problem was that traditional recommenders do not utilise the context. Instead, they focus on two dimensions: the user and the items (also called two-dimensional recommenders), excluding other contextual data that could be used in the recommendation process, such as the day/time, with whom the user is with, weather conditions, etc. Adomavicius et al. were among the first to prove that contextual information incorporated in the recommendation process indeed improves recommendations; they proposed that the recommendation procedure should not be two-dimensional but rather multi-dimensional, introducing the Context-Aware Recommender Systems – CARS [19, 53].

According to Adomavicius and Tuzhilin [19], contextual information can be used in two ways for producing recommendations. They named the first *"Recommendation via Context-Driven querying and search"*, where systems use contextual information from the environment (e.g. location), the user (e.g. profile information, user actions) and the system as searching parameters to search for the most relevant items in a repository to recommend. Examples of such systems are ubiquitous systems and location-based systems that utilise contextual information, often from the environment by using sensors, to recommend appropriate items in the proximity (e.g. restaurants, touristic attractions, etc.). The second way for producing recommendations is called *"Recommendation via Contextual preference elicitation and estimation"* [19] where systems focus on modelling user preferences by using various methods, e.g. observing the user while interacting with a system or by utilising feedback from the user regarding the items and recommendations.

2.3.1 Recommendation via Context-Driven Querying and Search

The first way of including contextual information in the recommendation process mostly refers to ubiquitous systems, where context is sensed (e.g. via sensors) or retrieved (e.g. from the user profile, the web, etc.) in order to be included in the search for appropriate recommendations. Early works included only few contextual parameters such as the location, user identity and people and objects around, aiming at informing people about things in their proximity [60, 61, 62, 63].

Besides trivial contextual parameters such as the location, many ubiquitous systems use more contextual information in their search for recommendations, such as date, time, season, temperature, user's interests, user's emotional status, etc. Cena et al. [64] propose a system called UbiquiTo that recommends items (places to visit, accommodations, restaurants) by using three types of context: user preferences, current context (location, time, etc.) and device type. The recommender calculates a score to each item and then orders these items to produce a list to be presented to the user. The scoring procedure takes under account the user's interest (e.g. if she does not like visiting museums then she will not be recommended such items) and the user's location (only items close to her location will be recommended). M. van Setten [65] offer context-aware recommendations based on the context (user location, time) and user interests. Böhmer et al. [66] propose a system for recommending applications for mobile devices. Due to the nature of the system (i.e. recommending mobile applications to mobile device users), much of the context can be captured automatically by the system via the mobile device. For example, a system that recommends books cannot be aware of information like how often the user reads each

book or for how long she reads it (unless perhaps in case it is an e-book); on the contrary, this can be achieved with mobile applications. Therefore, the system acquires context automatically (implicitly) via an application which runs in the background as a service and acquires as much contextual information as possible (user's location, time, what applications she is currently using or has been using in the past, when and for how long, etc.).

Current research on "context-driven querying and search" ubiquitous systems is mainly driven by context sensing and context identification and focuses at utilising as much of the available contextual information as possible in order to provide better, more personalised results. However, as Jannach mentions [67] such systems mostly filter the presented information content according to users' current location and preferences in a rather static approach, not utilising any of the sophisticated recommendation algorithms with machine learning aspects met in traditional recommender systems, such as Content-based and Collaborative Filtering.

2.3.2 Recommendation via Contextual Preference Elicitation and Estimation

Regarding "Recommendation via Contextual preference elicitation and estimation", three approaches were proposed: the Pre-filtering approach, the Post-filtering approach and the Contextual Modelling approach [19, 53]. CARS in this group do not use two-dimensional datasets as with traditional recommenders: *Users* × *Items* → *Ratings*; rather they face the challenge of coming against multidimensional datasets that include additional contextual dimensions besides "users" and "items": Users × Items × Context → Ratings [19].

The Contextual Pre-filtering approach aims at pre-processing the input data without affecting the actual recommendation process. It filters the initial multidimensional contextual dataset before it is provided as input to the recommender system. This dataset is multidimensional, as many contextual parameters have been considered. For example, a two-dimensional (2D) dataset includes data only on users and items, a three-dimensional (3D) dataset may also include data on the time, while a multidimensional one may include data on an arbitrary number of contextual parameters. The reduction-based approach is an example of Contextual Pre-filtering [19]. The aim of the reduction-based approach is to filter this multidimensional dataset in order to produce a 2D dataset which can be used as input in any of the classical pre-existing 2D recommendation methods. In this way the actual recommendation method does not change (it is still 2D). The filtering of the initial multidimensional dataset is done by assigning specific values to particular contextual parameters and then selecting only the dataset records that satisfy this assignment; the

result is that these contextual parameters are being excluded from the initial multidimensional dataset of the particular application, resulting in a 2D dataset. For example, let's consider a movie recommender with a 4D initial dataset and contextual parameters the day/time a movie is watched and the company with which it is watched (the other two entities/dimensions are the user and the movies). Suppose a user would like to watch a movie on Wednesday night with her boyfriend and wants to be provided with recommendations. The reduction-based approach will assign specific values to the contextual parameters day/time and company as follows: day/time="weekday/after 8 pm" and company="boyfriend" and select only the dataset records that include day/time="weekday/after 8" and company="boyfriend". By this, the reduction-based approach not only reduces the dimensions of the initial 4D dataset to two (by assigning specific values to contextual parameters the corresponding contextual dimensions in the initial dataset are eliminated and the resulting dataset is reduced, in this example from 4D it becomes 2D and can be provided as input to any pre-existing recommendation method), but also to exclude any irrelevant data from participating in the recommendation process (all records considering day/time="weekend" are irrelevant because the user is not interested in watching a movie at the weekend; considering irrelevant information in the recommendation process may lead to ineffective recommendations).

Another Pre-filtering example is the work of Baltrunas and Amatriain [68] who proposed a Pre-filtering method for recommending music to users. As they note, music recommendations have different characteristics than other types of recommendations because users tend to repeatedly use the same items more than once, i.e. when they listen to the same songs repeatedly (in book recommenders for example this does not apply). Moreover, besides user ratings on music items, a music recommender may also consider implicit feedback retrieved automatically: the songs a user has listened to, the number of times she has listened to each song, the artists she chooses more often, etc. According to the authors, an issue considering implicit feedback data is that information regarding negative user preferences could be missed, because the recommender only knows the items the user likes but is not aware of what applies to the rest of them: does the user like them or not? The authors propose a time-aware RS where the number of times a user has listened to an artist suggests how much she likes the particular artist. Instead of the whole user profile, they use micro-profiles which are basically snapshots of the user profile in certain time periods, e.g. morning, noon, night. The main problem with this method is how to divide a continuous contextual variable, such as time, into many distinct slots. For example, for one person the morning period can be 6-9 am while for others be 8-12 am, etc. By using only

the time-based micro-profile of the user instead of the whole profile the authors report that they have better accuracy results. This is a Pre-filtering approach since by using micro-profiles the input dataset of the recommendation algorithm is reduced. More works that use the Pre-filtering method are [69, 70, 71].

The Contextual Post-filtering approach does not involve any contextual filtering of the input dataset, nor does it involve the context in the recommendation process: recommendations are produced in the same way as in the traditional 2D recommendation systems. Rather, the Contextual Post-filtering approach filters the results of the 2D recommender based on some contextual parameters [19]. The filtering excludes any irrelevant recommendations based on the context and then prioritises the resulted recommendations according to their relevance to the context.

Contextual Modelling

The Multidimensional Contextual Modelling approach, as the name implies, is the only approach of the three that incorporates the multidimensional context in the actual recommendation process. According to Adomavicius and Tuzhilin [19], while the Pre-filtering and Post-filtering approaches can use traditional 2D recommendation methods, the contextual modelling approach promotes truly multidimensional recommendation methods, which essentially represent machine learning predictive models or heuristic calculations that incorporate contextual information in addition to the user and item data. The input data include more dimensions besides users and items, thus there is a need for developing appropriate methods that will include these dimensions in the recommendation process.

A Contextual Modelling approach is the work of Oku et al. [72], which extends a 2D classifier method called Support Vectors Machine (SVM) to support contextual parameters. Their method is called Contextual-SVM and adds a general contextual dimension to the 2D method, thus making it 3D. For given contextual information, the method results in a plane (the one dimension of the 3D is known - the context) which determines the preferences of each user for that context. To opine about how similar the preferences of two users are, the authors apply a similarity function that is related to the positive and negative data of each user (positive data include items the user likes, while negative data include items that she does not like). The similarity function is used in order to find the most similar users to the active user and form a neighbourhood, as in the CF recommendation method. To test their approach, the authors applied it to a recommender system for restaurants with generally good results. As they note, it is possible to classify user preferences based on the context by using the Contextual-SVM.

Yu [73] built a media recommendation system that incorporates context in the recommendation process by considering 4 different types of context: content context, operating context, user context and terminal context. According to the authors, content context is the context of an item, e.g. for a media file it could be information on the actor, genre, language etc, operating or situational context is the user's current location, time and activity, user context consists of user preferences and terminal context regards to terminal capabilities. For context representation they use an ontology-based context model. To produce recommendations, they use a hybrid method that combines appropriately the following three algorithms: for comparing media items against user preferences they use a content-based method with cosine similarity involving i) characteristics that are common between items and users' profile and ii) the weight the user has specified and the weights of the categories of the characteristics that they have in common. For comparing media items against situational context they use Bayesian-classifier to determine what is the probability of a media item to belong to (a) certain class(es) of context. Finally, for comparing media items against terminal context (format, frame size, frame rate etc.) they use a rule-based approach that determines which choice should be followed according to the situation. Based on the above, their recommendation approach initially compares media items against user preferences to opine about what a user likes, it calculates the probability an item fits the situational (current) context and it combines these two results to get the overall score for a user and an item under the situational context. Then, they compare media items against terminal context to find the appropriate form that the items chosen to be recommended should be presented. According to the authors, results were encouraging.

Bogers [74] on the other hand notes two important challenges in considering context in the recommendation process: collecting the context and representing it in a computable format. The author addresses the second problem by using a graph representation of a movie web site. The graph does not only represent users and items (movies) as in traditional recommenders, but includes more contextual attributes such as actors, tags, movie genres etc. The links between user nodes and movie nodes in the graph are weighted by the ratings assigned by the users on the corresponding movies and used in the recommendation process. The author argues that the proposed method allows for easy inclusion of different types of contextual parameters, such as actors, genres etc. in the recommendation process.

Bourke [75] propose a photo recommender that recommends scene compositions according to the context. The authors perceive any available data that could be obtained from the user's mobile device as the context, e.g. location data via GPS, direction, lighting

conditions, in order to recommend similar high quality, well composed photographs taken by other people in the same environment that are perceived as good photographs. The recommendation method compares the user's current context with user defined image metadata to measure the utility of each image on the current context: the more relevant an image is to the current context the higher its score will be. The utility also accounts for image popularity.

Cantador and Castells [76] propose a context-aware recommender system for recommending news. Their method uses Natural Language Processing (NLP) tools to break down articles into important concepts which are then compared to an ontology in order to find similarities between article concepts and ontology entities. If similarities above a certain threshold are found, then the particular article is annotated with the most similar entities. The contextual parameters used here are the concepts of the articles. During user evaluation, users commended that although contextualising the recommendations is useful, the annotations should have been disambiguated because often words can have different meanings to different persons or according to the context.

Baltrunas and colleagues [77] state that user preferences (and hence ratings) on items depend on the context and therefore context dependent rating data on items should be available. Context dependent rating data refer to user ratings on items that have been rated based on particular contexts: a user may rate multiple times an item, each rating taking place in different context. Therefore, in this setting the user has to provide ratings on items in different contexts. The authors discuss that receiving user ratings on items in many different contexts is a difficult task since the user cannot be forced to rate an item in all possible conditions, but even if she was willing to do so, it may not be possible to try the item in all conditions. Besides, Asoh et al. [78] have shown that supposing the contexts (as opposed to acting in real ones) is not sufficient, thus their usage cannot be guaranteed (see previous paragraph). However, the approach of Baltrunas et al. [77] is similar to the "supposed contexts" concept: for collecting context-dependent ratings on an item, the authors ask the user under which contexts she would like that item. Thus, instead of having the user actually rate an item under many contexts, the user is asked about her preferred rating on an item under imagining several contexts. The proposed method predicts a user's "best context" for each item and then, the items whose best context is most similar to the current context are recommended to the user. The best context is a vector consisting of all the contextual parameters with values zeros or ones based on what the user has specified liking. In this way one best context for each user-item is created. After finding the best

contexts for all user-item pairs, the method provides recommendations to a user in a given context as follows: return all items with best-context similar to the given context.

Aciar [79] addresses the research problem of how to extract useful contextual information from user reviews and comments in order to feed a RS. The proposed technique detects any sentences in the text that include any contextual information. According to the author, shallow parsers and classification algorithms based on term frequencies do not provide good results; instead, the author used rule based classification where sentences are being classified in two categories: "contextual" sentences include information about the context while "preference" sentences include information about user preferences. User reviews and comments are analysed and each sentence is classified as "contextual", "preference" or "none". The authors reported overall good results.

The above are representative examples of CARS that provide recommendations via "Contextual preference elicitation and estimation", and, in specific, contextual modelling approaches. The next section discusses related research issues and challenges.

2.3.3 Challenges in RSs and CARS

Table 2 categorises the challenges met in the different recommendation methods in RSs and CARS.

Research issues regarding the Contextual Pre-filtering approach include choosing the right generalised pre-filter in order to obtain the right dataset that will produce the best recommendations [19], dealing with potential computational complexity due to context granularity (context has often great detail which introduces computational complexity in the recommendation algorithms) and choosing the appropriate contextual parameters in application domains where context has great granularity and allows many possibilities. In addition, an open issue relates to combining the reduction-based approach with many 2D recommendation techniques (CF, Content-based filtering, Hybrid methods, etc.) to infer which combination is best for all application domains, if such a combination exists, or whether different combinations apply better in different domains.

As with the Pre-filtering approach, research issues related to the Post-filtering approach include dealing with potential computational complexity due to context granularity, choosing the appropriate contextual parameters in application domains where context has great granularity and allows many possibilities, as well as combining this approach with many 2D recommendation techniques to infer which combination is best.

Research issues concerning the contextual modelling approach have to do with selecting the dimensions to be included in the multidimensional recommendation model,

how can classical 2D recommendation techniques be extended into multidimensional and what new multidimensional techniques can be developed.

A question is which general method is best: Pre- or Post-filtering? Panniello in [80] compared one Pre-filtering approach against two Post-filtering approaches. They used the Exact Pre-filtering approach, the Weight Post-filtering approach and the Filter Post-filtering approach [80]. They resulted in that the Filter Post-filtering dominates the Exact Pre-filtering approach, while the Exact Pre-filtering approach dominates the Weights Post-filtering approach. They concluded that which method is best depends on which Post-filtering method is used. Therefore, one cannot argue that Post-filtering uniformly dominates Pre-filtering in all occasions, or vice-versa. Moreover, the authors state that it is not clear which of the two methods should be used at all times.

Another question proposed in [81] is whether the recommendation task to be used affects the performance of a context-aware recommender system and if there are occasions where traditional (un-contextual) recommender systems perform better than CARS. The authors argue that, despite the fact that more than often CARS have better performance than traditional un-contextual recommender systems, in some occasions this may not apply. The two recommendation tasks under study were Top-N: recommend only the best N items to the user (since only N items are recommended, it is important not to recommend irrelevant items that the user will not like) and Find-All: recommend all items that the user will potentially like (here it is important to recommend all good items without missing any). In the experiments, the authors compared an un-contextual recommender system against three CARS systems: one using Pre-filtering, one using Weight Post-filtering and one using Filter Post-filtering, all in combination with CF. All experiments were made over the two different recommendation tasks described above. The authors concluded that the recommendation task (Top-N or Find-All) indeed affects which recommendation technique is better among the un-contextual 2D RS and CARS. According to [81], un-contextual RS dominates CARS in "Top-1" (Top-N for N=1) in all recommendation accuracy measures (precision[3], recall[4] and F-measure), the un-contextual RS dominates CARS in "Top-4" only in recall while the other two measures are similar, and finally, the un-contextual RS dominates CARS in "Find-All" only in recall, while at the other two measures the CARS outperforms the un-contextual RS. In addition, they note that the particular CARS method used (Pre-filtering, Weight Post-filtering or Filter Post-filtering) does not significantly affect the recommendation process, while many conditions

[3] Number of relevant recommended items/number of recommended items
[4] Number of relevant recommended items/total number of relevant items

do affect it such as number of recommendable items and number of items recommended. The conclusions were that, when aiming at improving the recommendation performance by including context in a traditional RS, the recommendation task should be taken into account because it affects the recommendation process. Moreover, other characteristics should be considered such as the total number of items and the number of items in the recommendation list.

For more information on context-aware recommender systems the reader is referred to [19].

Table 2: Challenges/issues met in Traditional RSs and CARS.

Challenges/Issues
Collaborative Filtering
"New user" problem: a new user has to rate a certain amount of items for the algorithm to be applied.
"New item" problem: the recommender cannot recommend a new item until it has been rated by a number of users.
Scalability issues.
Performance problems for users with large information set.
Content-based Filtering
"New user" problem: a new user has to rate a certain amount of items for the algorithm to be applied.
Over-specialisation of items: focus only on items rated or purchased and similar ones.
Exclude other different types.
Limited use of information: use only features that are explicitly associated with the items to be recommended.
Pre-filtering
Choosing the right generalised pre-filter to obtain the right dataset that will produce the best recommendations.
Pre-filtering, Post-filtering
Computational complexity due to context granularity.
Choose appropriate context parameters in domains where context has great granularity: allows many possibilities.
Combining the approach with 2D recommendation techniques: 1. Which combination is best for all application domains? 2. Can different combinations apply better in different domains?

Contextual Modelling
Choosing the appropriate dimensions to be included in the multidimensional recommendation model.
Extending classical 2D recommendation techniques to multidimensional.
Develop new multidimensional techniques.

2.4 Machine Learning in Recommender Systems

In Section 2.2.1 we have presented the two methods used in Collaborative Filtering, the neighbourhood methods and the model-based methods, and discussed that the latent factor models (machine learning approaches) attempt to estimate ratings by characterising both items and users on latent factors inferred from the ratings patterns [4]. Some of the most successful realisations of latent factor models are based on Matrix Factorisation; it has been shown that Matrix Factorisation models are superior in terms of accuracy to neighbourhood methods for producing product recommendations, allowing also the incorporation of additional information besides explicit user feedback, such as implicit feedback, temporal effects, and confidence levels [4]. Explicit and implicit user feedback techniques are discussed extensively in Sections 3.1 and 3.2.

While factorisation approaches are known to provide high accuracy in several prediction problems such as recommender systems, their application to new prediction problems is a nontrivial task and requires a lot of expert knowledge [82]. In [83], Rendle introduced Factorisation Machines as a new model class that combines the advantages of Support Vector Machines with factorisation models. Factorisation Machines, like Support Vector Machines, are a general predictor working with any real valued feature vector; however, Factorisation Machines model all interactions between variables using factorised parameters. Thus, they are able to estimate interactions even in problems with huge sparsity (like recommender systems) where Support Vector Machines fail [83]. In [82], the LIBFM was introduced as a software implementation for factorisation machines that features stochastic gradient descent and alternating least-squares optimisation, as well as Bayesian inference using Markov Chain Monte Carlo.

In [84], Rendle et al. proposed applying Factorisation Machines to model contextual information in the recommendation process and to provide context-aware rating predictions (see Section 2.3 in terms of including context information in RSs). According

to the authors, this approach results in fast context-aware recommendations, as the model equation of Factorisation Machines can be computed in linear time both in the number of context variables and the factorisation size. The proposed method outperformed the best performing method at the time for context-aware rating prediction (the Multiverse Recommendation based on the Tucker tensor factorisation model), in terms of prediction quality and time of execution.

Baltrunas et al. [85] presented Context-Aware Matrix Factorisation (CAMF), an approach that extends Matrix Factorisation by modelling the interaction of the contextual factors with item ratings, introducing additional model parameters. Their approach provides comparable results to state-of-the-art but more complex approaches, in smaller computational cost.

Hidasi and Tikk [86] proposed the General Factorisation Framework that takes as input a user preference model and computes latent feature matrices. The General Factorisation Framework allows for easy experimentation on any context-aware recommendation task that is based either on explicit or implicit user feedback. The authors state that experiments with implicit feedback datasets have shown that proper preference modelling significantly increases recommendation accuracy, and that novel models in GFF outperform state-of-the-art factorisation algorithms.

In this thesis, we use three machine learning recommendation algorithms implemented within CARSKIT recommendation engine: two Context-Aware Matrix Factorisation algorithms (CAMF_CU and CAMF_ICS), and a Tensor Factorisation algorithm (CPTF) [87, 88]. Both types of algorithms (CAMF and TF) directly incorporate the context into the recommendation process: Tensor Factorisation by using independent modelling, which assumes contexts are independent with users (and items), and CAMF by using dependent modelling, which exploits the dependencies among users, items and contexts [88].

2.5 User Experience in Recommender Systems

Recommender systems research does not only concentrate on recommendation algorithms; much research has also been conducted on aspects related with user experience in using such systems. In this section we discuss such aspects that are related with the work in this thesis.

Konstan and Riedl [89] state that measuring user experience can often be challenging in recommender systems research, as it requires carrying out field studies with long-term users; that, being the only reliable way of measuring behaviour in a natural context [89]. Pu et al. [90] state that recommendation accuracy alone is not sufficient to achieve user satisfaction; other important factors are the system's effectiveness in presenting the recommendations, providing explanations for the presented recommendations and enhancing users' confidence to make decisions.

In terms of recommendation presentation, Pu et al. [90] discuss the *recommendation label* as the screen area where the recommended items are displayed; e.g. Amazon uses *"Customer who viewed this also viewed"*, *"Customers who bought this also bought"* and *"Recommendation for you"*. Another point is the location on screen where recommendations are displayed [90]: the right-hand longitudinal display places recommendations on the right hand side of the screen, while the lower latitudinal display places recommendations at the bottom of the screen. Example of the former is YouTube and of the latter is Amazon. Another parameter concerns the internal structure of the recommendations set, whether it will be a list, or a grid.

Minimising the effort users need to make during initial preference elicitation processes is important. Such processes require from users to complete tasks, e.g. rate items, so that RSs acquire initial user preference data to use for computing recommendations. In case users need to conduct such processes, entertaining means should be used to motivate users to provide input [90]. Another point is enhancing users' perceived accuracy (i.e. the extent to which users think that the recommendations they receive are accurate) by adopting an attractive layout design, effective labels, and recommendation explanations [90]. The aforementioned can increase users' perception of the system's effectiveness, as well as their satisfaction and trust [90].

Recommendation explanations are important as they can enhance users' trust and satisfaction [90, 91, 92]. Explanations educate users on the reasoning of the recommendation algorithm, making them aware of the recommender system logic, resulting in increasing system transparency, as well as users' trust towards the recommender. Moreover, explanations increase users' perception of the system's competence [92]. Research on whether users like explanations to be personalised, i.e. related with their preferences, showed that, while users were positive (users liked personalised explanations), these may have actually been less effective than generic explanations [91, 92]. Less effective explanations means not contributing in the selection of the recommended items by users. In [93], an interactive interface is proposed which

provides visual-based explanations. Through the interactive interface, users control aspects of the recommendation functionality of the tool. The interactive interface itself functions as an explanatory mechanism, where users can dynamically update their preferences during the recommendation session, as well as explore the interactive functionalities of the system. Evaluation showed that users found the system to be informative and helpful in making users understand how they receive their recommendations.

2.6 Other Aspects of Recommender Systems

In this section of the chapter, other aspects of recommender systems are presented that are related to the thesis.

2.6.1 User Privacy and the General Data Protection Regulation (GDPR)

In [94], authors conducted research on the factors that influence the quality of user experience in ubiquitous recommender systems. They state that privacy is a critical factor. The interpretation of privacy used is that "users are ensured and decide on what ways their data will be processed" [94]. In terms of users' dilemma on whether they will allow for their information to be processed for receiving more personalised recommendations, the authors distinguish between three groups of users: those that will provide any kind of information in exchange with high level of personalisation in recommendations, those that will give some information to receive a limited amount of personalisation in recommendations, and those that will not give information due to privacy concerns [94].

New EU regulation on the General Data Protection Regulation (GDPR) defines several articles that can be mapped to software functions of software systems, demanding that all systems conform to these articles. RSs, as systems functioning exclusively on user data, are affected by the regulation. In the following, we discuss some of the articles that concern RSs.

"Data Minimisation" is one of the principles related to the processing of personal data, stating that only personal data relevant and necessary for the processing purposes of each specific system should be asked, collected or in any way processed, except if the user chooses to provide more personal data. Data Minimisation affects preference elicitation processes of recommender systems, since such processes attempt to compile the initial profile of users by explicitly asking them for personal information or implicitly tracking

their behaviour. Moreover, as context information is critical in the computation of context-aware recommendations, context sensing techniques used by CARS need to be redefined in terms of the data they process under the general concept of context, that is thought to be *any* information related to the user, the computing system, the environment of the user and the interaction of the user with the system [42]. While by acquiring users' consent, the acquisition and usage of personal data is permitted, yet, this definition of context may be proved problematic in the near future when used by context-aware systems.

The "Right to Erasure" is defined as the right of users to claim from systems the erasure of all of their personal data. In such a case, systems are obligated to immediately erase the personal data. Users' "Right to Erasure" has an immediate effect on recommender systems performance in terms of recommendation accuracy, as it reduces the amount of data upon which the recommendations are computed. Moreover, it enhances the cold start problem (see Section 2.2.1), as it may reduce the ratings of particular items that have few ratings in overall.

The "Right to Rectification" states that users should be able to require and obtain the rectification of inaccurate personal data, or the completion of incomplete personal data. In such cases, the software system should immediately provide users with the option to edit all of their personal data or to add data where no data was previously given. In the case of recommender systems that model user preferences, especially where this is done implicitly, e.g. tracking users' clickstream data or browsing history, it is interesting to see how the "Right to Rectification" can be implemented. Should the system provide means for users to be able to update such data, i.e. be able to change data related with their interaction with the system? Such a task would interfere with the recommendation process, rendering it inaccurate.

To conclude, even if users initially agree in providing their personal information in exchange with receiving personalised recommendations, the new GDPR regulation provides them with the option to alter or retrieve such consent at any time, forcing the recommender system to comply with the request, which in turn might have an immediate impact on its performance. GDPR is discussed in Section 7.2 "Future Work".

2.6.2 Social Networks and Recommender Systems

A social network is a network of social bindings between people. Since the 1960s, research and industrial efforts have investigated advanced collaborative systems for leveraging human connections and improving human interactions in workspace environments.

Computer-Supported Cooperative Work (CSCW) has contributed much in offering advanced collaborative systems for leveraging human connections and improving human interactions in workspace environments, but these systems mostly focus on business-driven interactions where connections among people tend to be formal and structured [95]. Recently however, social and computing disciplines focused specifically on the design of social-networking services, i.e. applications that support human social interactions and can be more informal.

Recommender systems have been active on social networks, forming the *Social Recommender Systems (SRS)*. SRS aim at leveraging social relationships to improve recommendations [96]. The idea is that preferences of friends are more relevant, and thus are utilised more in the recommendation process, than the preferences of unknown (yet similar in terms of CF) users [96]. SRS provide recommendations of people and content from social networks that might be of interest to users. SRS are important, not only for users, but for social networks as well, as they enhance the adoption and engagement by users [97]. Domains of usage of SRS include, among other, community websites, blogs, multimedia, microblogs, and news. An example of multimedia recommendations is YouTube, where an advanced recommender system recommends videos that are diverse and relevant to the user's recent actions [97, 98]. The huge number of YouTube users renders the design of such a recommender system very challenging.

Another example of a unique recommender system is the one for the professional network LinkedIn where, as Guy notes [97], unique recommendation challenges can be met. In LinkedIn, recommendations of people, companies, professional groups and jobs are provided to users. For such recommendations to be accurate, many parameters should be taken into consideration that concern, not only the professional profiles of users (e.g. domains of professional experience), but also their personal information such as current home location and preferences (e.g. alternative location preferences). Other SRS can be found in Twitter and Facebook. In such social networks, the relationships of users are considered in terms of whether they are symmetric or asymmetric (whether connected users are reciprocal or not, i.e. one "follows" the other or "subscribes" to their profile), as well as whether they are confirmed or non-confirmed (both users' agreement is required for a relationship to be established or not).

More information on social recommender systems and their relation with e-commerce is provided in Section 3.1, where SRS for e-commerce are discussed.

2.7 Model-Driven Development

Model-Driven Development (MDD) aims at the abstract representation of application domains. Models through automated transformations or interpretations are converted into applications, eliminating or minimising the need to write code. Many works in the literature use MDD in various domains.

In [21], Hoyos et al. discuss that context management is a complex problem that developers of context-aware systems face, and present MLContext, a textual Domain-Specific Language (DSL) specially tailored for modelling context information. MLContext automatically generates software artefacts from context models, by applying MDD techniques [21]. MLContext has been defined as a metamodel, while model-to-text transformations have been written to generate the desired software artefacts. MLContext is focused on providing a high-level of abstraction, as well as to be easy to learn, and to promote reuse of context models. The authors support their decision in implementing a DSL in that, while UML has been used in most proposed approaches in the literature (by using UML profiles), DSL are considered more appropriate because they allow modifying the metamodel, whereas UML profiles do not. Similarly, in [22], Ou et al. examine how Model-Driven Architecture (MDA) can be applied to tackle the issues of context modelling and context-aware application modelling and development. They argue that using ontologies to model context introduces additional complexity in the development process of pervasive services, and they propose using MDA for modelling of context ontology and for automatic development of ontology-based context-aware applications [22].

In [99], White et al. propose a model-driven development approach for automated enterprise application configuration. As enterprise applications are large-scale software programs hosted on multiple application servers that support a great number of users simultaneously, while at the same time they have a large number of components, complicated configuration files and complex interdependencies between components, the authors argue that such applications are hard to configure, and thus techniques for automated configuration of such applications are needed. The proposed MDD approach builds a model that specifies the application's configuration rules. Configuration artefacts, such as XML configuration files, are then generated from the model. The approach uses the model to execute a series of probes to verify configuration properties, formalises feature selection as a constraint satisfaction problem, and applies constraint logic programming techniques to derive a correct application configuration [99].

Model-driven development has recently been applied on cloud computing as well. In [100], Achilleos et al. discuss that, recently, businesses have been provided with a vast pool of choices and a diversity of cloud infrastructures, platforms, and tools offered by cloud computing, creating thus several challenges. As they note, this diversity hinders interoperability, promotes vendor lock-in, and prevents businesses from making informed and optimal decisions when transitioning to the cloud [100]. In particular, there might be a need for businesses to utilise many different cloud providers, for example in the case where there is a need for firms to reduce costs, therefore a hybrid cloud deployment could be desirable, where the applications' servers can be deployed in a public cloud, while the database servers are deployed in the private cloud of the firms. In the context of the PaaSage FP7 EU funded project[5], an open source integrated platform has been developed, to support design and deployment of Cloud applications. The PaaSage platform allows model-based development, configuration, optimisation and deployment, supporting existing and new applications independently of the existing underlying cloud infrastructures. It offers a model-driven approach, which incorporates workflow-driven, script-based deployment of applications [100]. In addition, the PaaSage platform supports monitoring and adaptation (that enable for scalability, elasticity, optimisation and cloud bursting) through the developed probes for CPU and memory, as well as by enabling the application owner to develop additional monitoring probes [100].

In Chapter 2, we have presented and discussed the background areas related with this thesis, as well as challenges met in the corresponding areas. The following chapter discusses related work.

[5] PaaSage (Model Based Cloud Platform Upperware) was a project funded by the European Union under the Seventh Framework Programme - Objective ICT-2011.1.2: Cloud Computing, Internet of Services and Advanced Software Engineering (www.paasage.eu/).

Chapter **3**

Related Work

Chapter 3 initially discusses related work in terms of explicit and implicit user feedback acquisition techniques for the online scenario (Section 3.1), as well as the physical scenario (Section 3.2). Next, available open source recommendation frameworks and engines are discussed in Section 3.3. The section continues with related works from the literature that are considered to be closest to the work presented in this thesis (see Section 3.4). This chapter closes with discussing previous research works of the author that have led to the conception of the current work in Section 3.5.

3.1 User Feedback Acquisition Techniques for the Online Scenario

Collaborative Filtering constitutes the most adopted method for RSs since it relies on user's *behaviour history*, such as previous transactions or ratings of items [53]. It is also independent of any domain. The most efficient CF approaches are the model-based approaches that utilise intelligent machine learning techniques such as Matrix Factorisation. In e-commerce settings, CF is preferable since it relies only on user behaviour and, therefore, explicit profiles for users and products are not needed [5]. Hence, users can receive recommendations without firstly being asked to complete a profile, a task users normally do not like.

RSs for e-commerce use *explicit user feedback on products* such as product ratings by users to elicit and model user preferences and offer personalised recommendations of products users would enjoy [5, 20]. In case explicit feedback is not available, RSs use *implicit user feedback on products* by *tracking user behaviour* such as user transaction data

(purchase history), clickstream data[6], click-through rate[7] (CTR) and browser history information [101, 102, 103, 104]. Implicit techniques have been used in RSs for products in online retail stores (e-commerce websites), as well as movies, music, journal articles, web documents and content, online news articles, books, television programs, and other RSs applications. In [105], Jawaheer et al. discuss explicit and implicit user feedback techniques in RSs, listing possible applications of the techniques, as well as related considerations. Beyond ratings, they note other explicit techniques being user comments and reviews, as well as tagging of items by users. In terms of implicit user feedback, the authors state that this is based on observable behaviours exhibited by users and, thus, such behaviours are directly related with the domain under study. For example, in e-commerce settings users may browse products through the e-store, while in a music RS users can listen to tracks, skip tracks or search for their favourite ones.

Problems with explicit techniques are that they require cognitive effort from users, and that they interrupt their task at hand as they have to stop and rate items. Problems of using user's implicit feedback are: (i) no negative feedback is available (user actions denote what the user likes but not necessarily what the user does not like); (ii) implicit feedback is inherently noisy: while user behaviour can be tracked, true user preferences and motives can only be guessed [5]; e.g., a purchased item could be a gift, or the user could eventually be disappointed with the product; (iii) explicit feedback indicates preference, whereas implicit feedback indicates confidence [5]; (iv) evaluation of RSs using implicit feedback requires more refined appropriate measures.

Early works adopting implicit user feedback methods, such as [106], mention that the cognitive effort needed from users to accurately rate content may act as disincentive, leading thus to data sparsity when it comes to user-item rating matrices. The authors identify three types of implicit feedback categories, namely examination, retention and reference and categorise implicit user feedback from the literature into these categories (e.g. user behaviour such as selection, duration, and purchase are categorised under examination, while saving a reference goes under retention).

In [101], the authors discuss implicit feedback techniques applied to RS for electronic books. They argue that, while in most cases of RSs explicit user feedback is used (e.g. ratings), this may cause inconvenience to users as it changes their regular navigation and reading patterns when they have to stop and rate the items. The authors use photo books to conduct an analysis and comparison between implicit and explicit

[6] Data about which webpages users visit, e.g. product webpages.
[7] The ratio of users who click on a link to the number of total users who view the webpage.

feedback. User explicit feedback consists of one to five star ratings by which the user can rate content (e.g. photos). User implicit feedback consists of the following parameters which are recorded by the system: duration of the session/content size, number of clicks, reading time of a piece of content, number of visits to a content, reading time of a category, number of accesses to a category, number of comments, number of recommendations to a friend. Their findings are: (i) the more time a user spends on a content indicates that the more he/she likes that content, since higher ratings exist from the user for that content - there is a direct relation between displaying time and explicit ratings; (ii) more user visits on a content mean the user is interested in it - there is a direct relation between the number of visits and explicit ratings; (iii) multiple user accesses to the same content shows user interest in that content.

In [102], the authors focus on defining and collecting user preferences in online scenarios where user explicit feedback is not available, however, interference with the website source code is possible in order to observe user actions. The authors propose considering the context of the user interaction as well, such as the context of the currently visited page or the context of the user's browsing device, since such contexts can potentially affect user behaviour. Their approach firstly integrates multiple implicit feedback indicators on page objects (view count, dwell time, distance travelled by mouse, total time the mouse cursor was in motion, scroll distance, scroll time, purchase) using machine learning methods, and secondly, it engineers in the method appropriate contextual features (total number of page links, total number of images in page, total length of the text presented in page, page dimensions, browser window dimensions, visible area ratio between browser and page dimensions, whether user uses a hand-held device). As the authors' goal was to evaluate the contribution of context to the recommendation quality in an e-commerce website, they have constructed appropriate multidimensional datasets using both implicit feedback parameters and contextual parameters and used them in the recommendation process. The authors have incorporated the context features into the user implicit feedback feature space. In terms of results for the e-commerce website, the authors state that the usage of context clearly improved performance in predicting user purchased objects.

In [103], it is shown that users' diverse implicit feedbacks can be used to improve recommendation accuracy. Various user activities are categorised into positive and negative feedback groups. The authors log the specific user behaviour on an item. CF is used, but instead of using user ratings on items, the authors propose using a function that takes into account user's behaviour on items. The authors effectively replace user ratings in

CF with the proposed behaviour based function. The proposed method was tested using a music recommender system, for which positive user behaviour included explicit song play (with user's intention), implicit play (without intention), play completion (without skip), search and add an item to playlist, and register a new song. Negative user behaviour included explicit skip (by clicking skip button), implicit skip (by playing other song), and delete an item from playlist. The authors report good results, outperforming baseline methods, and point out that play completion has more effect on the performance than other user behaviours.

Dwell time as an implicit user feedback parameter has also been researched by Yahoo in [104]. The authors discuss that, when the click-through rate is used as implicit user feedback, it is not always reliable as it does not capture any post-click user engagement (e.g. a user may have clicked on an item by mistake or because of link bait and not be truly interested in the presented content), and propose using dwell time on content items on the Yahoo home page to measure how likely a content item is relevant to a particular user. The authors consider also context in their approach, since users' consumption of content items can vary by context; e.g., users have on average less dwell time per article on mobile or tablet devices than on desktop computers [104]. Thus, for each consumed item, user's dwell time across different context is extracted (e.g. types of content and devices). One of the two methods used in this work is CF with Matrix Factorisation using a user-item interaction matrix where, instead of user ratings on items, user dwell time on content items is used. For evaluation purposes, the proposed dwell-based method was compared to click-based methods, yielding competitive or better performance [104].

In [107], Nguyen et al. propose a RS for a mobile application of an online fashion portal that aims to be completely unobtrusive by not utilising at all explicit user feedback acquisition techniques. Rather, the system relies solely on implicit feedback, i.e., on users' behaviour. In particular, the system uses users' "intent to purchase" history, users' click history on products, as well as products that are wanted by users. The authors order the user feedback data in the sense that they utilise the most important first. Thus, wanting an item is a stronger indicator than clicking on it, while intending purchase is the most powerful signal [107]. As products are fashion related, the authors consider in their method the time period that user feedbacks on products are recorded; e.g. if a product was viewed a long time ago, that will probably be less relevant today. The authors infer a user's implicit score for an item by blending the abovementioned methods. For testing purposes, different recommendation algorithms have been utilised with this method and compared to a

baseline that always recommends the globally most popular items. According to the authors, results demonstrate the effectiveness of their method.

Hwangbo et al. [108] propose a recommendation method that combines users' online click history on products on the e-store of a fashion company[8], with users' product purchase history from the same company's physical store, to reflect the online and physical preferences of customers respectively. The authors argue that, since customers' purchases in physical shopping malls reflect the preferences of these customers, by combining such purchase data with click data on products from online customers, the performance of online recommendations can be improved. In their setting, the same products are available in the online and physical stores of the company, but the online and physical store customers are different, and, as they point out, there is no information that can be used to link these two groups of users together. This scenario differs from the settings of our work, where a customer has an online presence on the e-store, as well as a physical presence in the physical store, where these two are linked together via one user account in the system. Another critical difference of this work from our method is that, in Hwangbo's work, the physical scenario consists of user purchases on products [108]; in this sense, no ubiquitous user-product interaction data are utilised. Their recommendation method, besides considering the e-store click history dataset and the purchase history dataset from the physical store, it also considers product metadata, as well as a "preference decay function" that reflects customers' change in preference over time. The authors conducted an experimental evaluation with real users of the company over a three week period. Their recommendation method was compared against a conventional item-based CF system that pre-existed on the company's e-store that only uses online click data. The results indicate better performance of the author's method in terms of number of user clicks on the recommended items, which acts as proof that utilising implicit and explicit user feedback acquisition techniques can improve recommendation accuracy.

While based on e-commerce, *social commerce* is more focused on the role that users' social network relationships play in users' decision-making process [109, 110]. In an interesting review of social commerce related literature, Han et al. [111] note that there are three main streams of defining social commerce: (i) that it is the fusion of social media with e-commerce, (ii) that it is a subset of e-commerce using social media to facilitate

[8] Company statistics: 5 million members; 40 000 products sold yearly online; 1.5 million clicks and 10 000 online transactions are available per month; 1 300 physical stores in Korea; 20 000 products sold yearly.

social interactions and enhance the online shopping experience, and (iii) as an internet-based commercial application, making use of Web 2.0 technologies and social media while supporting user-created content and social interactions [111]. As Li et al. discuss in [112], social commerce is helping people buy where they connect by integrating social media into e-store sites and adding e-commerce functionality to social networks.

In the context of social commerce, *Social Recommender Systems for Commerce* utilise the preferences of social relationships of users, in addition to e-commerce related information, to recommend products to users. In [112], the authors address the problem that e-commerce RSs do not consider user behaviour on social networks, while SRS do not consider user behaviour on e-stores. They propose a social recommender mechanism that considers preference similarity, recommendation trust, and social relationship, with the aim to increase the accuracy of product recommendations in e-commerce. Preference similarity between two customers refers to the similarity of their product rating records [112]. Recommendation trust refers to the success rate of product recommendations made by a customer to other customers, based on the latter's product rating records. Social relationship refers to the degree of closeness between two customers in terms of implicit interactions or explicit ratings between them in a social network [112]. Experimental results of this method have shown that the proposed RS outperforms other benchmark approaches. In [109], the authors utilise the difference between a user's rating score for an item and the "real" score (meaning the average score of the item) to estimate the reputation for that user in an attempt to achieve reputation-based trust.

Hooda et al. in [113] propose utilising explicit user feedback from user ratings, combined with implicit user feedback from the users' social friends network to build a hybrid product recommender system. As they note, rating prediction using users' friends network assumes that items that users' friends rate highly, will also be a good recommendation for them. The underlying assumption is that a person has relationships with people who have similar preferences to her. The recommendation method proposed in the paper unifies the matrices obtained from both user-item product rating and friend's network. According to the authors, experimental evaluation showed that their social hybrid product recommender can deal with data sparsity problems.

3.2 User Feedback Acquisition Techniques for Physical Retail Stores

3.2.1 Introduction

In [2], it is discussed whether the scenario of providing recommendations on-line on e-commerce settings could also be applied in physical settings, i.e. in physical retail stores. The authors state that *"facilitated by recent developments in ubiquitous computing, it seems reasonable that the success of such recommender systems in the virtual world can be replicated in the real world"*. There already is a trend towards personalisation in the (physical) retail market and RSs along with ubiquitous computing could be the means to make the personalisation more sophisticated and effective [2].

The authors in [13] point out that indoor (physical) shopping is still the main shopping mode (in comparison to e-commerce) and argue that, while promising, ubiquitous mobile recommendations are still absent from this mode of shopping mainly due to positioning technologies limitations with regards to successfully identifying user behaviour: detecting the location of a customer in the shop, determining the item(s) a customer is currently looking at, etc. We note that, from the time of publication of this article (2012), ubiquitous technologies have evolved and can be used for user behaviour identification, as discussed in this section.

An important question is whether all retailers would benefit equally from personalisation. In [2], it is stated that this depends on the variety of their product assortment and the diversity of their customer base. They distinguish 4 basic store categories: RSs are most helpful in large shops with many product categories and extensive variety of products in each category (i) where the customer base is likely to be very heterogeneous. In this scenario, RSs can help customers to orient themselves in the store and to compare products [2]. In speciality shops (ii) that have a few product categories but a large variety of products in each category (e.g. a tie shop), personalisation is crucial and sales staff play an important role to that. A good RS can possibly replace or supplement the sales person. In stores with limited choice over many categories - many product categories but limited variety of products in each category (iii), a RS is useful to customers to decide whether the quality of the products offered matches their requirements and whether they do or do not need to go to a more specialised shop [2]. Finally, in little convenience stores or grocery shops that only offer basic products in a few categories (iv), customers do not expect to find a good match for their taste and, in this case, a RS may not be very useful.

In addition, by using a RS, trying different, unknown brands from those that are recommended to the user is not as costly as trying brands without receiving

recommendations [2]. This is because, since RSs recommend products the user would like, the risk that the customer chooses something that eventually will not like is much lower than without using a RS.

RSs significantly improve cross-selling, i.e., selling related or complementary products, in addition to products that the customer originally aimed to purchase. As an example, in e-commerce settings, Amazon display related or complementary products under the label *''Customers who bought this ... also bought ...''* or *''You might also find these ... interesting''*. Similarly, RSs provide more opportunities for physical retail stores to increase cross-selling. Often, cross-selling is realised by placing related items close to each other (in nearby shelves). RSs relax this requirement by allowing the advertisement of related or complementary products, without the need for these products to be in close proximity. For example, based on what a customer has in their shopping cart and their location in the store, a RS could suggest related products that are also in the user's proximity.

In [16], it is supported that technology can be used to offer context-awareness and personalisation in the shopping paradigm, by inferring the products a customer is interested in and supporting them with information on those products in the form of personalised recommendations and advertising. According to the authors, while interactive displays have been used for this purpose, personalised information based on individual needs is still insufficient [16]. They state that, to place customers in authentic shopping contexts, it is critical to put them in shopping procedures that combine both real and virtual shopping environments.

From the above discussion it is shown that physical retail stores would benefit from the application of recommender systems. In the following section we provide the most important works from the literature regarding the application of RSs in physical retail stores.

3.2.2 Techniques

Research has been conducted on using the knowledge obtained from e-commerce RSs to offer recommendations to customers while being in physical business locations, such as a theme park, a shopping mall, a grocery store, or a restaurant. While most of these works base their recommendations mostly on customer's purchase history, the authors in [20] suggest offering recommendations based, among other, on customers' in-store shopping path: the RS recommends shops a customer should visit next by using information on store

visit paths of users in the shopping mall. The recommendation algorithm used is CF based on user's behaviour history. While in e-commerce settings user similarity can be defined as a function of ratings on items, purchase frequency of customers on items and page views (browsing history record), in [20] the authors have proposed a similarity function that uses the customer's *"staying time"* in each area, which can be calculated from the customer's shopping path data. According to the authors, among the data that can be extracted from customers' shopping path data, staying time at each selling area is considered as an important piece of information on user preferences in purchasing of goods. Other works [114] have revealed that staying time in an area is related to the level of interest of the user in the item in that selling area. It is also highly possible that, customers who stay for a long time in a selling area make unplanned purchases of items in that selling area [20, 115, 116, 117]. The authors have evaluated their RS performance by comparing their results with real sales data, resulting in that, not only sales history, but also customer's shopping path data make a RS highly accurate [20]. According to the authors, this verifies the usefulness of a RS using in-store movement history data.

In [13], a mobile RS for indoor shopping is proposed, that relies on a novel indoor mobile positioning approach (achieving store level accuracy) by using received signal patterns of mobile phones to recommend brand stores to users in a big shopping Mall. The proposed RS records users' past activities and context, among other the time spent in each store during every shopping process. Recommendations are based on users' current position, weights of preferences in terms of brand stores, as well as store profile information. The authors have tested their method by using two user groups, one where user preferences on brand stores are implicitly inferred by their method (experiment group), and a second one where users explicitly state their preferences by rating brand stores (contrast group). The authors report that there are no significant differences concerning accuracy of recommendations for the two groups, despite the fact that the proposed method only uses users' location and the time spent in each store to infer preferences. According to the authors, participants in the experiment group felt more usefulness, more ease of use, and more satisfaction than participants in the contrast group, since they were not forced to rate brand stores.

In [118], a RS that recommends shops in a mall is described. To detect customers' location, RFID devices and related infrastructure have been deployed in a modern large-scale shopping mall. The work in [2] describes a scenario, where the customer is recommended with products in store via a store-owned device called personal shopping assistant (PSA). The recommendations are based on the items that are currently in the cart,

the location of the customer, and their purchasing history. Customers are able to rate products and set their preferences. The PSA can assist customers in finding products with particular features, while it provides descriptions of products and product comparison.

In [119], a RS for shopping is proposed that estimates user preferences on items based on their physical distance from the items (whether they are near the item), whether a user picks up an item or whether a user scans an object using a RFID reader device. For example, the system can detect whether a user has moved near a product and assign a user preference score 2 for that item, while if the user picks up the item the preferences score is 5. Scanning the item is a 10 preference score. These preference scores are then used with CF to offer personalised recommendations to the users. In a similar approach, GECKO [120] estimates user preferences on items via the use of visitor observation triggers that "capture the information required for predicting a visitor's activities and interests", and 'capture visitor's behaviour suggested by the space'.

Pfeiffer et al. [121] present a RS which is worn by the user during in-store purchase decisions. The system is context-aware in the sense that eye-tracking technology is applied to learn from the users' attentional processes when standing in front of a product shelf. The system can automatically detect the user's needs and provide appropriate product information and recommendations. Eye-tracking is used to (implicitly) elicit preference information in a minimally intrusive manner, in order to reduce the user's effort [121, 122]. Being aware of the user's context, such as location information, eye movement and gestures can aid in implicit user preference elicitation. As opposed to approaches that work with mobile devices such as smartphones, in [121] glasses are used as a more ubiquitous object than smartphones: glasses are very personal devices with distinct ownership that makes them acceptable for this kind of systems [121].

Reischach et al. [123, 124] have researched how item-related ratings, textual comments and recommendations of users to other users can be applied in real shopping scenarios by using ubiquitous computing. Such recommendations are more often met in E-commerce settings where users buy online through websites that also use for posting their feedback. Facilitating users during their actual shopping process (i.e. at the actual store) can be achieved by recommending them with products they might like and by providing them with comments, suggestions and ratings on products they are about to buy. Recommendations to customers while shopping can be offered when the mobile device or wearable is able to identify the product the user shows interest in and provide her with ratings and comments on that particular item, as well as recommendations of other products she may like [123]. The most interesting approaches in the literature are works

that deploy NFC (Near Field Communication) to enable mobile devices to read RFID (Radio-Frequency Identification) tags, in order to identify real products in-situ and provide the user with information on that products [124, 125, 126, 127]. The ubiquitous system in [124] enables users to scan any RFID tagged product by using a mobile device and receive/produce recommendations, reviews and ratings on that product. The system also enables the user to comment on an item during usage in real-time via a mobile device. A user study on the above system [125] revealed that almost all users would appreciate the system on their mobile phone, and that they were interested in receiving recommendations from the system, in addition to just product information. Moreover, some users were reluctant on relying on other people's opinion on products, while others were in general reluctant towards mobile applications, but as they mentioned, they would try the system at least once. Privacy and data security were not a concern for the respondents.

The work in [16] combines both real and virtual shopping environments by employing RFID technology in a store to provide shoppers with relevant information. In specific, a smartphone application is able to identify customers' preferences by scanning RFID tags of products using smartphones with RFID readers and provide information related to the product, as well as product recommendations in real time in an automatic manner. Recommendations are computed by clustering similar customers based on customer preferences in terms of products and purchasing records.

The mobile city guide CityVoyager proposed by Takeuchi and Sugimoto [128] is an intelligent city guide for mobile devices equipped with GPS which retrieves and recommends shops in the proximity that match each user's preferences. The authors differentiate from typical context-aware city guides on that they focus on applying the techniques used in E-commerce recommendation systems to real shopping. In particular, they use the item-based Collaborative Filtering algorithm to filter shops in a similar way traditional recommenders filter items. Instead of user ratings on items, the authors use the location history of a user. Similarities between shops are calculated using a similarity function. The basic function of the system is to estimate users' preferences from the history of their location data and offer tailored shop recommendations upon request.

3.2.3 Summary

There is literature supporting that the provision of recommendations in e-commerce settings could also be applied in physical settings, i.e. in physical retail stores [2, 13, 16], especially in large shops with many product categories and extensive variety of products in each category where the customer base is likely to be very heterogeneous. It has been

proven that recommendations of products to customers in e-commerce settings boost sales [1, 2], increase customer satisfaction [1] and improve user experience [5], and this would be the aim as well by enabling RSs in physical retail stores. Since RSs significantly improve cross-selling, by enabling such systems in physical retail stores, the requirement of placing related items close to each other (e.g. in nearby shelves) is relaxed [2].

For providing recommendations to users while shopping in physical retail stores, CF can be used, since similar circumstances apply as in the online scenario. CF relies only on user behaviour and, therefore, explicit profiles for users and/or products are not needed. It would be inconvenient, unpleasant and difficult for users to fill-in a preferences questionnaire or a user profile before they actually shop at a store, as it would interfere with their imminent shopping task.

In terms of facilitating shopping in physical retail stores, ubiquitous (mobile and wireless) technology can be used to offer context-awareness, while RS technology and methods can offer personalised content to users. More to the point, in cases where explicit user feedback, such as user ratings on products is not available to elicit user preferences, implicit user feedback can be used by tracking user behaviour. While in e-commerce settings, tracking user behaviour essentially means analysing user transaction data (purchase history), clickstream data and browser history information, in ubiquitous settings, other context-aware methods must be applied to identify user interest on products, such as tracking customers' in-store shopping path [20, 118], the staying time in the area of a product [20], the time spent in each store (in the Mall scenario) during the shopping process [13], whether the user has looked at a product (via wearables) [121], picked it up [119] or scanned it [119, 16, 124], whether the user is near the product [119], the items that are currently in the user's shopping cart [2]. Table 3 summarises the ubiquitous technologies used in the above works.

This thesis aims to enable the usage of explicit and implicit user feedback acquisition techniques from the physical retail store and the electronic store through an MDD methodology that offers high level of abstraction. The aim of proposing the use of such techniques is to increase the available user-product interaction data upon which context-aware recommendation algorithms function, with the goal to further improve recommendation accuracy and recommendation availability. Our methodology is described in Chapter 5.

Table 3: Ubiquitous technologies in works that facilitate shopping in physical retail stores.

a: RFID/ NFC Product Scanning;

b: Wi-Fi Location Tracking;

c: 3G Location Tracking;

d: Mobile Received Signal Strength;

e: Ultrasonic 3D tagging system;

f: Wearables

Paper		Methodology
[20]	**a, b**	Tracking customers' in-store shopping path. Shopping carts with attached RFID tags enable for tracking the shopping cart's position in the store and record its shopping path.
[2]	**a**	Uses a Personal Shopping Assistant for tracking products in the user's shopping cart and customer's location. All products must be tagged with RFID chips. Able to identify products the customer is currently looking at or has added in their shopping card.
[13]	**d**	Indoor mobile positioning approach using received signal strength (RSS) from mobile phone received signals to recommend brand stores to users in a big shopping Mall (store level accuracy).
[119]	**a, e**	Ultrasonic 3D tag system for product indoor positioning is used to track user's physical distance from the items. "Near an object": the distance between the user and a physical object is within 50 cm. "Picking up an object": user picks up an object over 20 cm. "Scanning an object": user obtains the data of physical object using a RFID reader device.
[118]	**a**	RS that recommends shops in a mall: to detect customers' location, RFID devices and related infrastructure have been deployed.
[121]	**a, b, c, f**	Eye-tracking technology is used via special glasses to identify user interest on products on shelves. The system uses different technologies for localisation (GPS, Wi-Fi, 3G, NFC, Accelerometer, Gyroscope, and Compass). UI enables using speech, gaze and gestures for communication.
[124]	**a**	Users scan RFID tagged products by using a mobile device to receive/produce recommendations.
[16]	**a**	A smartphone application is able to identify customers' preferences, by scanning RFID tags on products using smartphones with RFID readers, and provide product information and recommendations in real time.

3.3 Available Recommendation Frameworks and Engines

A number of open source recommendation frameworks and engines are available for the development of recommender systems, as well as the experimental evaluation of new recommendation algorithms with existing ones. In this section we briefly present the most well-known frameworks.

Duine

The Duine Framework [28] is a collection of software libraries that allows developers to create recommender systems for their own applications. It is a software module that calculates how interesting information items are for a user. The Duine Recommender currently offers 7 prediction techniques: Collaborative Filtering, Information Filtering, Case-based Reasoning (based on how similar items have been rated by the user in the past), GenreLMS (reasoning on genres), TopNDeviation (popularity), AlreadyKnown and UserAverage [28].

Apache Prediction IO

Apache PredictionIO [31] is an open source machine learning server for developers to build smarter applications with machine learning and data rapidly. It enables developers and data scientists to create predictive engines for machine learning tasks. It uses an interface for the developer to select options for the recommender.

EasyRec

EasyRec [24] is an open source recommendation engine with which website owners (e.g. online shops) can add recommendations to their websites (a web application written in Java) and provide personalised recommendations to their users using RESTful Web Services. User Actions are sent to the easyrec using a REST API. Possible actions are: viewing, buying or rating an item. These user actions are stored in the database. Analysers periodically analyse all recorded data for identifying patterns to generate recommendations. These Recommendations can be accessed through calls to the easyrec webservice API and presented to users.

GraphLab Create (Turi create intelligence)

Turi Create [29] is a machine learning modelling tool for developers and data scientists. It simplifies the development of custom machine learning models.

MyMediaLite Recommender System Library

MyMediaLite [30] is an open source, lightweight, multi-purpose library of recommender system algorithms. It addresses the two most common scenarios in Collaborative Filtering: 1. rating prediction (e.g. on a scale of 1 to 5 stars), and 2. item prediction from positive-only feedback (e.g. from clicks, likes, or purchase actions).

LibRec

LibRec [26] is a GPL-licensed Java library for recommender systems. It implements a suite of state-of-the-art recommendation algorithms. LibRec aims to solve two classic tasks in recommender systems, i.e., rating prediction and item ranking by implementing a suite of state-of-the-art recommendation algorithms.

Apache Mahout

Apache Mahout [27] is designed to enable mathematicians, statisticians and data scientists to quickly implement their own algorithms.

LensKit

LensKit [25] includes a set of tools for Recommender Experiments. It enables researching recommender algorithms, evaluation techniques, and user experience, as well as to build recommender applications.

CARSKIT

CARSKIT Recommendation engine is an open source Java-based context-aware recommendation engine which is recognised as the first open source recommendation library specifically designed for CARS. It implements the state-of-the-art context-aware recommendation algorithms.

In [33] a systematic comparison between Hybreed (the authors' framework), Mahout, LensKit, MyMediaLite, GraphLab and Duine has been conducted. The authors state that only Hybreed from these software frameworks covers aspects such as context-awareness, hybridisation, and recommendations for groups. A comparison of these frameworks (except Hybreed) is also conducted in [87], including CARSKIT. The authors discuss the frameworks and state that only MyMediaLite and LibRec are able to provide state-of-the-art recommendation algorithms, in addition to the classical ones. Also, it was stated that

only CARSKIT recommendation engine provides context-aware recommendation algorithms [87].

3.4 Related Work on Addressing RS Development Complexity

A number of works in the literature propose using modelling or other software engineering techniques to assist developers in the development of RSs by tackling RS development complexity.

In [129] it is mentioned that, despite the high practical relevance of RSs in industry, little research has been conducted on the engineering aspects of such systems, while no comprehensive software framework is yet available that supports component-based development approaches for such complex systems. An important challenge in the design of such a framework lies in the choice of the appropriate level of abstraction so that the individual components can be re-used in different application domains, and a number of possible data sources can be easily integrated [129].

The authors in [33] address the abovementioned issue by proposing a recommendation framework named Hybreed for assisting developers build CARS and hybrid RSs [33]. They state that research into RSs has been concentrated on the development, optimisation and evaluation of recommendation algorithms, giving thus less attention on the support and facilitation of the development of RSs from a software engineering and architectural perspective. Hybreed supports the development of RSs from four functional aspects: (i) in combining methods and algorithms into hybrid RSs, (ii) in including the context in the recommendation process, (iii) in providing group recommendations and (iv) in applying a decentralised, service-oriented approach in developing RSs. The framework is component-based that also offers a high level of abstraction. In specific, Hybreed incorporates a set of standard recommendation algorithms and provides templates for combining them into hybrids with a significantly reduced amount of effort. According to the authors, Hybreed utilises a dynamic contextualisation approach to provide a high-level abstraction for developers, reducing programming effort as well as increasing comprehensibility and usability when creating complex recommenders. It is the case however, that Hybreed requires developers to write code. To develop an application using Hybreed, a two-step process must be followed: (i) developers need to utilise Hybreed's View Java interface to create customised views (views describe

what constitutes the context for a particular task or goal, as well as how to deal with it to provide recommendations), and (ii) instantiate a Kernel object by passing the implemented views to it and delegate to it all functionalities regarding state and profile management, virtual state creation and view application [33]. The authors have evaluated their framework using three use cases implemented by the authors, as well as through an evaluation with five developers who had basic knowledge on recommender systems and CF that Hybreed uses, while the tasks they had to execute constituted a rudimentary test of the framework capabilities.

In [35], the authors deal with the problem of web developers needing assistance, by means of proper methods and tools, for dealing with complexity issues in adopting recommendation techniques in their web applications. They explicitly mention the lack of model-driven methodologies for the specification of RS algorithmic and interface elements. The authors define a modelling proposal that integrates recommendation algorithms into the specification of e-commerce applications. In specific, the authors use OOWL (Object-Oriented Web Solution), a model-driven development process for web applications that uses a code generation approach from UML-compliant models. The authors use UML classes to model an un-contextual Item-to-Item CF recommendation approach. However, by modelling the recommendation algorithm itself, the complexity an algorithm can have to be able to be used in the proposed model-driven process is somewhat limited, in the sense that too complex recommendation algorithms will be difficult to be modelled. The authors state that the development of their case studies of an online travel agency and a bookstore took about half the time of the development time of an alternative content-based recommender system; the approach has not been evaluated with developers, as the case study has been implemented by the authors.

In [32] a user modelling framework for CARS is proposed, named UM4RS that serves as a tool for developers and researchers to build data models for CARS. In specific, the authors' aim is to increase the productivity of developers while building the data model (user, item and context) for CARS. A CARS model schema and UML class description is provided. The paper suggests using a Dataset Generator that helps developers export the data from the database to datasets that recommendation algorithms can use as input source. The interaction of developers with the system mainly consists of creating objects from the classes of the framework and performing actions such as create, retrieve, update and delete. The framework was evaluated based on how effectively it could create CARS data models out of datasets from the literature with positive results. However, the framework was not evaluated with developers.

A framework that facilitates the development of CARS for mobile users is proposed in [130]. The framework proposes a pull-based architecture for pull-based mobile recommendations; i.e., recommendations are provided as an answer to a query explicitly submitted by the user and evaluated by the system as a continuous query [130]. The architecture makes use of sensor information, as well as uses the perceived context from users by asking them of their current situation. The proposed recommendation method automatically updates and compares the context of the user with the context of the items; in essence, a content-based filtering of user context against item context is conducted to find similarities. The item with the most similar context with the user's context is recommended. By comparing contexts the system understands whether the user is in a similar situation as in the past, or whether she is in a similar situation as others in the past. A question, however, is whether the user should still be recommended with the same item as she was recommended then. The framework utilises specific algorithms from external toolkits and libraries (Weka and Mahout): a pre-filtering, a post-filtering and a contextual modelling; additional algorithms from the literature seem not to be easily supported though. The paper describes the architecture in detail but does not provide the actual steps as to how developers can use the framework to develop CARS. In our understanding, this can be done by setting up and using the proposed infrastructure and algorithms, without the possibility for any changes or additions.

The work presented in this thesis differs from related work in five important aspects:

i. It defines a novel Domain Specific Modelling Language for UbiCARS (UbiCARS DSML), abstracting technical details to a higher level;
ii. It focuses on the ubiquitous scenario of UbiCARS, aiming to enhance product recommendation accuracy and availability in physical commerce;
iii. It supports intelligent, complex algorithms and data models from the state-of-the-art of RS literature;
iv. It is easily extendable, as new model elements can be added to support additional functionality (e.g. additional explicit and implicit user feedback acquisition techniques); and
v. It is directly integrable with existing e-stores.

Table 4 lists the available tools for aiding the development of RSs, including their key characteristics.

Table 4: Tools for aiding RSs development and their key characteristics.

System	Methodology	Purpose	Evaluation	Developers do
Hybreed [33]	Component-based approach.	Reduce development effort.	5 developers: found to reduce effort.	Write code: utilise interfaces and instantiate objects.
Rojas et al. [35]	OOWL and code generation from UML.	Reduce development time.	No evaluation.	Work on code level.
UM4RS [32]	Propose a UML class description for CARS.	Increase productivity in building data models.	No evaluation.	Write code, instantiate classes create objects.
Rodríguez-Hernández et al. [130]	Pull-based architecture.	Assists the development of pull-based mobile CARS.	No evaluation.	Set up and use the specific infrastructure and algorithms.
UbiCARS MDD Framework	Use MDD, defines a DSML and a Modelling Editor.	Reduce development complexity & time for non-RSs experts, as well as increase recommendation accuracy and availability	Evaluated with 25 participants: found to reduce development complexity and expedite the development of UbiCARS (see Section 6.2).	(i) Use the UbiCARS DSML and modelling editor to design models; (ii) use the system UI to configure e-stores (see Section 5.1).

3.5 Early Work on Tools for Aiding the Development of CARS

As part of prior research work of the author that has led to the conception and realisation of the contributions of this thesis, in this section, we briefly present four research works in which we discuss frameworks we have proposed for aiding in the development of CARS. Please note that these frameworks are not an inherent part of the work of the thesis; rather,

they are considered prior work. More information, though, can be found in the corresponding papers.

In [39], a contextual modelling framework for CARS is presented, aiming to enable developers to uniformly model contextual parameters for CARS at the application layer of recommender systems, as well as provide a good reference for developers in using the framework and extending it both at model level and code level, in order to build their own application oriented models. In particular, a model template was designed and built as a UML class diagram by using the Eclipse EMF, as shown in Figure 1. The modelling framework enabled for automatic generation of Java code by using the EMF.

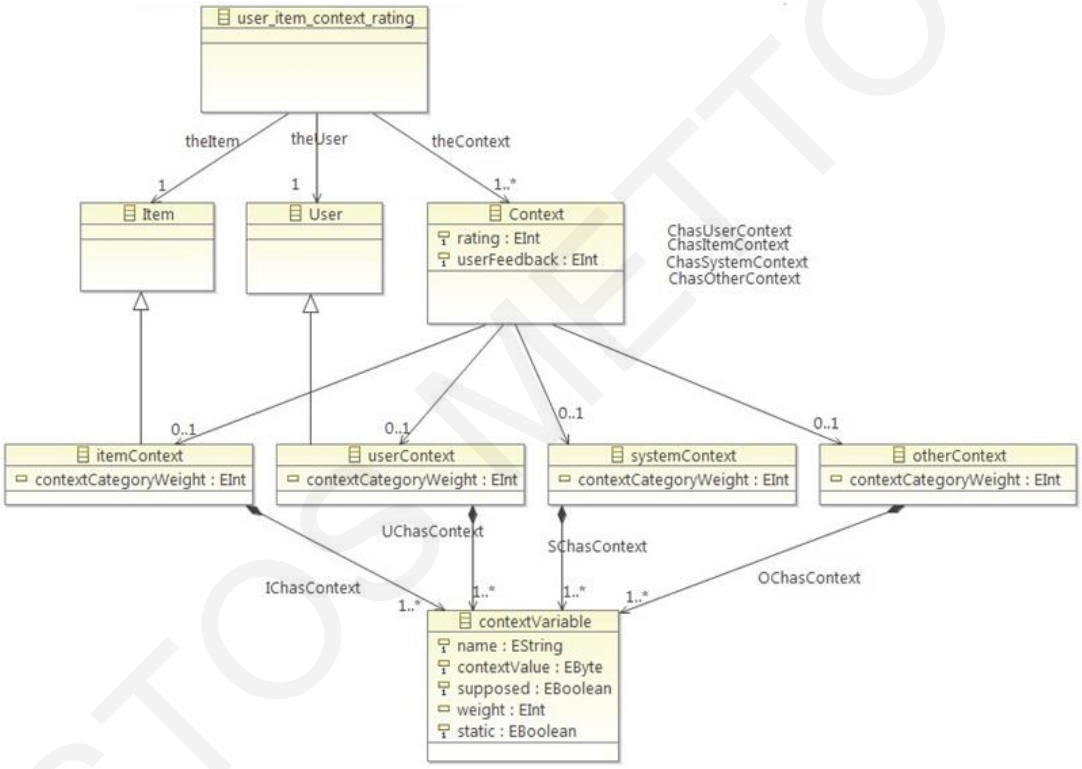

Figure 1: The proposed Contextual Modelling Framework for CARS in [39].

As future work, the paper proposed, among other, to extend the framework by conceptually modelling functionality (i.e. recommendation algorithms) in addition to the context. The goal is to research whether by including conceptual sub-models of recommendation algorithms in the framework, the development of more efficient CARS can become easier for developers and researchers. There is at least one work that proposed conceptually modelling functionality [35], where the authors use UML to model an un-contextual RS algorithm (see Section 3.4). At the same time though, by modelling the recommendation algorithm itself, the complexity of the model highly increases, while too complex recommendation algorithms will be difficult to be modelled.

Certain limitations of the UML model proposed in [39], in particular that it was time consuming for developers to extend it and that it required programming knowledge to use it, acted as the motivation for the work we proposed in [40]. To address the aforementioned issues, the work in [40] developed an easy to use, easy to extend web-based system called "Context Modelling System and Learning Tool". The system acts as a context modelling system and learning tool that is able to guide CARS developers through the process of CARS context modelling, i.e. on how to build their own context models at the application layer of RSs, in a way that offers sharing and reuse. At the same time, the platform acts as a CARS learning tool, by displaying and explaining concepts derived by CARS research, aiming to advance developers' knowledge in the field. Examples of concepts constitute the *"context dependent rating data"*, *"supposed context"*, *"static/dynamic context"*, and *"context weights"*. More on these concepts can be found in [40]. Figure 2 depicts a context model and part of the learning functionality of the tool.

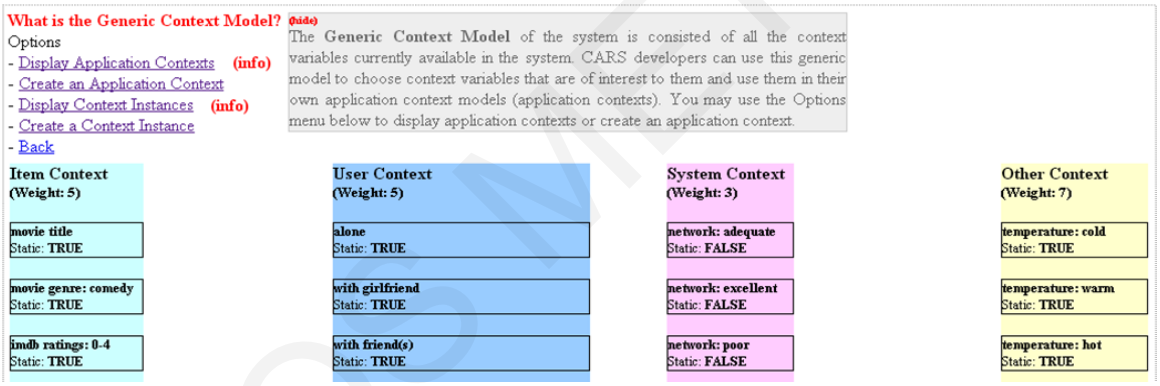

Figure 2: Structure of a Context Model Example: Four Context Categories are displayed with their Context Variables. Explanation of CARS Concepts (by clicking or mouse-hover on links) is also shown. Figure taken from Poster of Paper [40].

In [37] and [38] we have extended the "Context Modelling System and Learning Tool" by offering extended functionality upon the user defined context models, the most important of which are: recommendation of similar application context models to the user (see Figure 3); validation of application context models through context instance models; and comparison of application context models between them regarding their common context variables. More information on the aforementioned functionality can be found in [37] and [38]. At the time of publication, the system provided context models as outcome in the form of images. Future work included supporting the extraction of the models in xml and txt formats, as well as aim to support CARS developers in incorporating the context models within their RSs by providing appropriate tools.

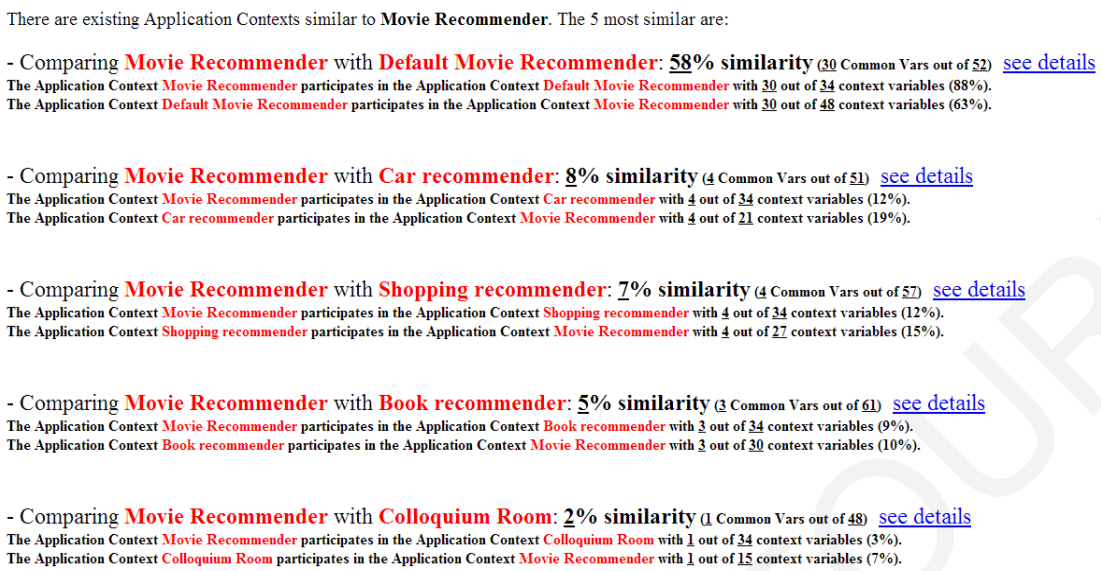

Figure 3: Recommending Application Context Models that are similar to the user's Context Model in [37].

Although the abovementioned works aimed to assist developers in developing CARS, a number of limitations of these works eventually hindered them from this goal. Firstly and more importantly, their contribution lies in modelling only the context and not the entire recommendation process. Although helpful, context models could not by themselves enable the development of full RSs. Secondly, the nature of the proposed frameworks required coding from developers, while the support provided was not to the expected level (context models were outputted only in the form of visual aids). This led to the need for a new framework that would be fully model-based and would require the least coding effort from developers. Furthermore, explicit support for the ubiquitous aspect of RSs was needed.

In the next chapter we discuss the types of systems explored in this thesis, the Ubiquitous Context-Aware Recommender Systems - UbiCARS.

Chapter **4**

Ubiquitous Context-Aware Recommender Systems - UbiCARS

4.1 Definition of Ubiquitous Recommender Systems

Ubiquitous recommender systems (UbiRS) facilitate users on-location by providing them with personalised recommendations of items in the proximity via mobile devices. Intelligent tourist guides, navigation aids, and shopping recommenders that recommend based upon user activities and behaviour patterns are examples of such systems. In [18], we provide a definition for ubiquitous recommender systems and ubiquitous context-aware recommender systems. In this section we discuss these two types of systems.

In the sense of facilitating users on-location by recommending items in the proximity, ubiquitous recommender systems are similar to ubiquitous systems that provide "Recommendations via Context-Driven querying and search" (see Section 2.3.1). However, we distinguish ubiquitous recommender systems in terms of the recommendation methods they follow: they do not rely on "search in repositories"; rather, they are more intelligent, similar to those of traditional recommender systems.

The aforementioned are depicted in Figure 4 [18]. The broad area of Ubiquitous Computing includes two sets: *"Ubiquitous Systems"* and *"Ubiquitous Recommender Systems"*. The set "Ubiquitous Systems" depicts systems that provide "Recommendations via Context-Driven querying and search", the first way for producing recommendations in CARS, as stated in [19] and Section 2.3.1 (for ease of reference, we refer to "Ubiquitous Systems" as "CARS-a"). The set "Recommender Systems" depict traditional un-contextual recommender systems, as discussed in Section 2.2.

The set "CARS-b" depicts systems that provide "Recommendations via Contextual preference elicitation and estimation" (the second way for producing recommendations in CARS) as stated in [19] and Section 2.3.2. In these systems, model-based CF algorithmic techniques are used, as discussed in Section 2.2.1 (subsection "Model-based methods") and in Section 2.4. The arrow from the set "Recommender Systems" to the set "CARS-b" depicts the usage of un-contextual recommendation algorithms within Pre- and Post-Filtering techniques used in "Recommendations via Contextual preference elicitation and estimation" (see Section 2.3.2). Moreover, as "CARS-b" systems use different methods for utilising context in the recommendation process than "Ubiquitous Systems", the line patterns in the corresponding sets in the figure are different.

The set "Ubiquitous Recommender Systems" includes ubiquitous systems that use the context to facilitate users on-location via mobile devices in the same way as systems in the set "Ubiquitous Systems" do, but with the important difference that the recommendation procedure they follow does not rely on "search in repositories". The set "Ubiquitous Recommender Systems" includes systems that use intelligent recommendation methods, similar to those recommender systems use. This is depicted via the "INTELLIGENT RECOMMENDATION METHODS" arrow connecting the set "Recommender Systems" with the set "Ubiquitous Recommender Systems".

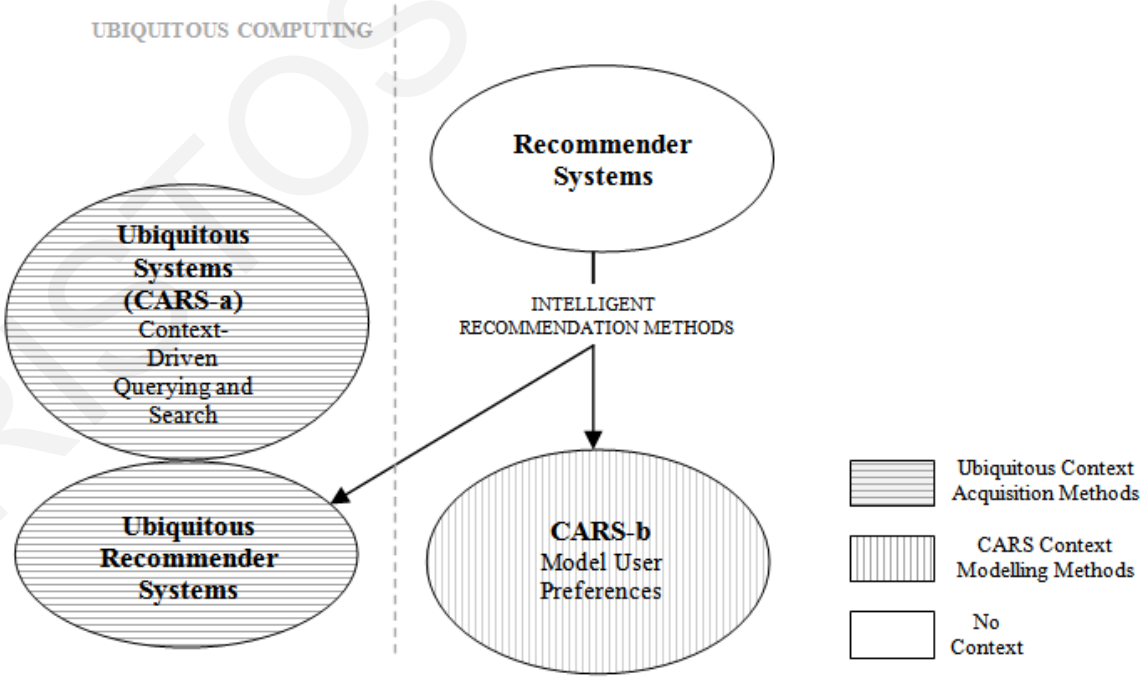

Figure 4: Classification of Ubiquitous Recommender Systems [18].

The most representative and well known ubiquitous recommender systems are the *ubiquitous recommender systems for products* met in commerce, as such systems manage

to effectively combine the benefits of the very popular, widely used e-commerce web recommenders with the benefits that ubiquitous systems offer.

4.2 Definition of UbiCARS

Although many works in the literature focus on ubiquitous recommender systems [123, 124, 125, 126, 127, 128, 131], to the best of our knowledge, no work had attempted to analyse in detail and classify ubiquitous recommenders in terms of ubiquitous computing and recommender systems research, so that their similarities and fundamental differences are presented. In [18] we have attempted the aforementioned, by providing a definition for Ubiquitous Context-Aware Recommender Systems – UbiCARS, which we discuss in this section.

As systems originated from two very different and solid research areas, ubiquitous recommender systems are not easily classified, as one may observe from Figure 4. As systems in both ubiquitous and recommendation areas become more and more complex, more subclasses of systems appear that tend to combine characteristics, methods and solutions from both areas to provide novel applications. The ubiquitous recommender systems have managed to effectively combine state-of-the-art solutions from the ubiquitous computing research area with well defined, effective and solid recommendation methods to provide a new user experience that may improve our lives, as well as stimulate researchers to continue this research trend.

However, since ubiquitous recommender systems use sophisticated recommendation methods most often met in traditional recommender systems instead of context driven searches (see Figure 4), could these recommendation methods become enhanced with context modelling methods used in CARS-b systems, i.e. systems that recommend "via Contextual preference elicitation and estimation"? The aforementioned are depicted in Figure 5. The new set of systems called *Ubiquitous Context-Aware Recommender Systems - UbiCARS*, would constitute a subset of "Ubiquitous Recommender Systems" and would utilise both types of context methods, the ubiquitous context acquisition methods (horizontal lines pattern in Figure 5) to enable the provision of recommendations on location via mobile devices (i.e. identification of near-by products, acquisition of user feedback data on products in ubiquitous settings, ubiquitous context sensing), as well as the CARS context modelling methods (vertical lines pattern in Figure 5) which refer to

state-of-the-art model-based CF recommendation algorithms that solve the multidimensional recommendation problem: Users × Items × Context → Ratings (see Section 2.3.2). The resulting context pattern for UbiCARS includes both horizontal and vertical lines.

Note that in Figure 5, the UbiCARS set differs from the CARS-b set in two points: (i) UbiCARS lies under Ubiquitous Computing and (ii) it uses the ubiquitous context acquisition methods and ubiquitous user feedback methods. The two sets have been positioned adjacent to depict their similarities: using CARS context modelling methods through state-of-the-art model-based CF recommendation algorithms. Considering the above discussion, we make the assumption that:

UbiCARS systems would be able to provide better recommendations than ubiquitous recommender systems that currently do not consider the context in the actual recommendation process.

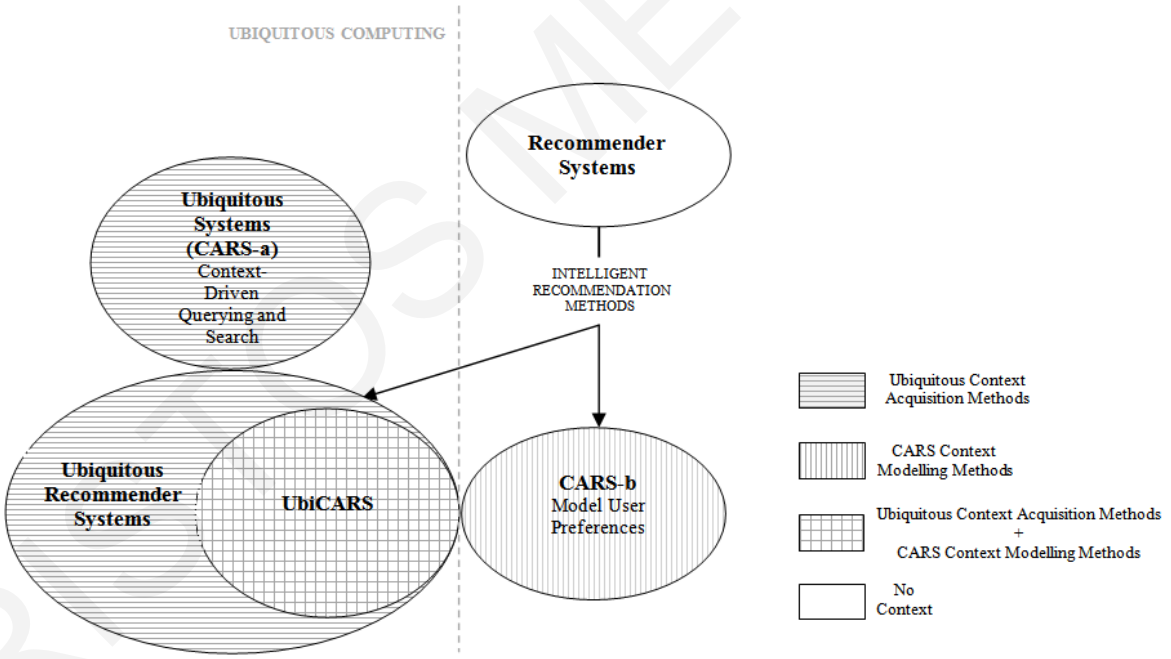

Figure 5: Classification of UbiCARS Systems [18].

In this research, the concept of UbiCARS is used for enabling user feedback acquisition and computation of context-aware recommendations in both the physical store (ubiquitous scenario) and the e-store (online scenario). In terms of the ubiquitous scenario, UbiCARS supposes the usage of a mobile application. In the next section, the challenges of UbiCARS are discussed.

4.3 UbiCARS Challenges

In this section, we discuss challenges met in the development of UbiCARS. Such challenges are related to Ubiquitous Computing research and Recommender Systems research.

Challenges Related to Ubiquitous Computing that Also Apply to UbiCARS

- Due to operating in ubiquitous environments via mobile devices, UbiCARS face important technological challenges such as energy concerns, storage limitations, wireless technologies issues, connectivity issues and networking issues (network connections must be maintained while devices move between various networks of different characteristics).
- The mobile device must be able to track user intentions in order for the system to understand what system actions could help the user accomplish her goals. For example, the mobile device of a UbiCARS for products must be able in real time to identify the item the user is interested in, i.e. she is having in front of her or holding at any given time. Appropriate state-of-the-art technologies should be used effectively towards this aim.
- Users interact with small, often non usable devices that need much user attention. UbiCARS need to offer user-friendly mobile devices with usable User Interfaces. Moreover, it is important that UbiCARS do not interrupt user's task.
- UbiCARS devices may not be transparent to users, since they operate on-field by considering various contextual parameters that trigger system actions and adaptations. Not transparent devices often provoke feelings of frustration to users.
- Context sensing: appropriate technologies and sensors must be utilised in order to infer the context in real-time. Context sensing should be done automatically and system actions based on context changes must be transparent to the user.
- Usage of available context: in a ubiquitous setting, context includes information from the user, the environment, the mobile device, as well as the interaction between them. A UbiCARS for products considers (among other) user preferences (what the user likes) in combination with environmental context (e.g. the current day/time of day and the particular location) in order to provide users with personalised recommendations.

- Privacy concerns arise: the user must trust the system in order to agree to provide sensitive information such as location and preferences. Furthermore, privacy-preserving techniques need to be applied.
- Apply appropriate authentication mechanisms in order to ensure the authenticity and quality of any user created information, such as ratings and comments on products.
- Scalability is also an issue: the system must be able to support many UbiCARS devices that will consume/provide information at the same time, without compromising efficiency or privacy.

Challenges Related to Recommender Systems that Also Apply to UbiCARS

- Building effective, user preference models using state-of-the-art model-based CF recommendation approaches.
- Such recommendation approaches have high complexity: difficult to be used by non-experts in recommender systems.
- The "new user" and "new item" problems appear with use of CF.
- Users of UbiCARS have a completely different task than users of CARS on the web. The former users have limited mental and physical resources available, since, being mobile forces them to simultaneously do other things as well, such as shopping, talking to others and observing their surroundings. Moreover, the amount of time they are willing to spend on the mobile device for receiving recommendations is much less - a study suggests that mobile users get distracted on average after four seconds when waiting for a computational result in a mobile context [124]. Therefore, few and accurate recommendations must be provided to UbiCARS users on-location.
- Mobile devices have many interface limitations that traditional PCs do not, in terms of the small screen, small physical or software buttons and the absence of mouse and keyboard. When designing UbiCARS mobile applications there is a need to consider the abovementioned in the way recommendations and recommendation explanations will be displayed to the user.

The UbiCARS challenges described above have been considered during the design and development of the UbiCARS MDD Framework.

Chapter 5

The UbiCARS MDD Framework

In this work, model-driven development is used as a methodology for conceptually modelling the Commerce Recommendation Domain as a Domain Specific Modelling Language (DSML) for UbiCARS, aiming to abstract the technical details, reduce development complexity and expedite the design, development and deployment of recommender systems for electronic and physical commerce. The UbiCARS MDD Framework aims to address the limitations of early works the author has proposed, as discussed in Section 3.5.

In specific, the UbiCARS DSML models the entire recommendation process for UbiCARS, as opposed to modelling only the context. Context models cannot by themselves enable the development of full RSs. Another important limitation was the requirement from developers to write code in order to instantiate classes and objects. The UbiCARS MDD Framework facilitates the design, development and deployment of UbiCARS on e-stores (whether these pre-exist or not) and physical stores in an automated manner, during which developers do not need to write code. In case developers need to extend the functionality of the framework with additional user feedback acquisition techniques, the UbiCARS DSML facilitates the inclusion of such techniques in the UbiCARS models as elements, while the UbiCARS Framework considers the new techniques in the UbiCARS infrastructure that is deployed on e-stores. As such, developers will only need to write (limited) code for the part of acquiring the new data from the users. The framework supports the development of UbiCARS in this aspect.

A third limitation was that context models were outputted only in the form of visual aids. Instead, the models outputted by the UbiCARS DSML are further used by the

UbiCARS MDD Framework for the configuration and deployment of UbiCARS on e-stores and physical stores.

In addition to addressing the limitations of early works as described above, the UbiCARS DSML enables developers to acquire ubiquitous user feedback data from the physical store, and use it in the recommendation process, together with user feedback data acquired from the e-store, aiming to improve recommendation accuracy and availability.

In this chapter, the UbiCARS MDD Framework is described in detail.

5.1 UbiCARS MDD Methodology

The proposed UbiCARS MDD framework [36] defines a Domain Specific Modelling Language for UbiCARS, and a corresponding graphical modelling editor. Via the editor, practitioners can use the DSML to drive model-based design and dynamic configuration of UbiCARS in commerce settings, as well as system integration with pre-existing e-stores. In Figure 6, the multi-layered software architecture of the framework is presented. The UbiCARS DSML, through the modelling editor, acts on the Modelling layer and enables practitioners to design their UbiCARS models.

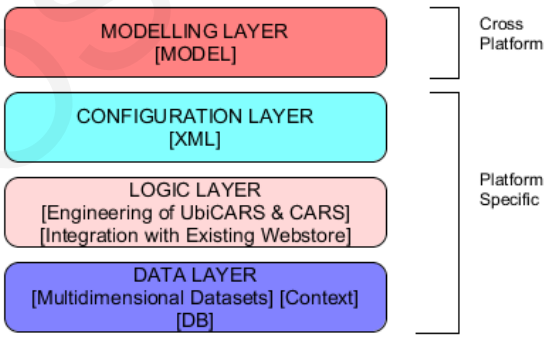

Figure 6: Multi-layered Software Architecture.

When UbiCARS design is completed, the framework generates configuration files in eXtensible Markup Language (XML) to be utilised by the Configuration layer, where a Parser extracts all user defined information and passes it to the Logic layer that undertakes the engineering of UbiCARS.

The Logic layer builds the necessary data models and database (DB) tables, whereas it also implements the system configurations indicated by the Configuration layer, including system integration with pre-existing e-stores.

The Data layer prepares the necessary context-aware user-product interaction datasets to be fed to the recommendation engine.

The framework receives as input the outcome of the modelling layer, i.e. the model, via an interface named *system UI*, to then proceed with the configurations dictated by the three other lower level layers, as described above. Developers can use the system UI to feed their model to the framework in order to configure the e-store to compute, accommodate, and present recommendations to users.

In terms of configuration, the UbiCARS MDD framework configures a *UbiCARS app* and a *CARS system* as follows:

UbiCARS app is a mobile application that:

- **Enables user feedback acquisition from the physical store by tracking of user interaction with products on-location**. Specifically, similarly to the dwell time used in online settings as user implicit feedback [102, 104] (see Section 3.1), we propose utilising the *"Staying Time in front of a product"* and the *"NFC tag Scanning of a product"*.
- Displays product recommendations to users on-location.

CARS is a server-side system (including the recommendation engine) that:

- **Enables user feedback acquisition from the e-store by tracking of user interaction with products on-line**. In specific, users' ratings on products, browsing history and purchase history are tracked.
- Computes context-aware recommendations.
- Displays product recommendations to users through the e-store.

To use the MDD framework to build UbiCARS, a practitioner only needs to use the DSML editor to design the UbiCARS model and feed it to the system UI. Then, automatically, the CARS system and UbiCARS app are configured and deployed.

5.2 The UbiCARS DSML

5.2.1 Introduction & Novelty

Domain-Specific Modelling Languages, DSML, are high-level languages[9] that are specific to a particular application domain. DSML are closer to the problem domain and its concepts than general purpose modelling languages such as UML, or general-purpose programming languages (GPP) such as Java. In this manner, DSML are capable to better model a domain by offering a higher level of abstraction from technical specifications, as well as offer customisation in terms of the modelling language, the produced modelling editor and the toolbox. DSML are expected to improve productivity and comprehensibility [132].

We have implemented the UbiCARS DSML to offer the highest possible level of abstraction from the technical details concerning the physical and electronic commerce recommendation domain, as well as to expedite development of RSs, simplify usage and instantiation of models by practitioners, as well as increase system maintainability.

In particular, the novelty of the UbiCARS DSML is summarised as follows:

- It is domain specific for the physical and electronic commerce recommendation domain: as such, any domain specifics are abstracted from developers in order to reduce development complexity and expedite the design of UbiCARS for commerce.
- Details of complex, state-of-the-art recommendation algorithms (such as model-based CF approaches) are abstracted from developers. Their usage is offered in an easy and straightforward manner.
- It is cross-platform, meaning that any platform specific implementation details are abstracted from developers at design time. As such, developers need not worry about technical details while designing their UbiCARS models, which reduces development complexity and expedites design and development.
- It offers automation: UbiCARS models are defined at an abstracted level and, using automated interpretations, they are converted into UbiCARS systems, eliminating (or minimising in case additional user feedback is needed) the need to write code.
- It increases productivity by enabling for dynamic integration with existing e-stores, as well as by offering reuse of UbiCARS models: models can be reused for configuration of e-stores.

[9] High-level programing languages offer abstraction (are independent) from a particular type of computer.

- It is easily extendable, as new model elements can be added to support additional functionality (e.g. additional explicit and implicit user feedback acquisition techniques).
- It increases system maintainability by providing the means to easily enhance a UbiCARS model by adding (or removing) elements and reflect these updates in the CARS system and UbiCARS app in an automated fashion.
- It enhances comprehensibility of the physical and electronic commerce recommendation domain, as well as promotes communication between recommender system professionals via standardisation of the terminology in the domain.

5.2.2 UbiCARS DSML Description

The DSML was designed as an Ecore metamodel in Sirius [133], an open source Eclipse project which allows leveraging the Eclipse Modelling technologies (EMF, GMF) to create custom graphical modelling workbenches. The Eclipse Modelling Framework [134] (EMF) itself is a framework and code generation facility for building tools and applications based on a structured data model. As the authors in [132] note, designing an editor using the Eclipse platform with EMF and GML results in a high quality product, especially regarding usability aspects. Several useful services offered that can be used with little effort are listed in the paper: loading/saving (in XMI); zooming; tool palettes; overviews; exporting to images; offering extension points for other applications to access the models created; and multi-platform support. Moreover, it is possible to efficiently add new editor functionality after a first version of the editor is created, adapting the editor to changes in the metamodel is simple, while in case new attributes, class, or associations are added to the metamodel, then the editor can still open files created with the previous version. However, deleting or renaming classes or attributes can lead to backward incompatibility problems [132]. We can confirm the aforementioned from our experience in designing the UbiCARS DSML editor.

In [132], a comparison of development tools for DSML is conducted (tools compared are: GME, Tau G2, RSA, XMF-Mosaic, Eclipse EMF+GEF), where the Eclipse platform is noted as the tool offering the highest level of graphical completeness (it is able to fully reproduce notations with advanced concepts with fidelity), as well as the best usability in terms of user experience, tool feedback, and overall number of features. The paper also mentions drawbacks of the technology, such as the programming effort required to implement the various shapes and connectors, multiple undo/redo, label editing, and

property sheets. Please note that from the time of publication of [132] (2006) the Eclipse platform and related technologies have significantly evolved.

For the purposes of DSML development, we have used the Sirius with Obeo Designer Community edition [135] that simplifies the creation of graphical modelling workbenches.

Figure 7 presents the Modelling Language as a metamodel. The `Application` element is the core element that represents a recommender system for commerce, consisting of a `CARS` system and a `UbiCARS` app. `CARS` defines the user ratings on products via the `Rating` element as an explicit user feedback element, and the `PurchaseHistory`, `ClickStream` and `BrowsingHistory` as those representing the user implicit feedback on products (see Figure 8). A CARS can instantiate zero or one of these elements, while a number of custom `NewUserFeedback` elements can be defined by the user which can be explicit or implicit (default value is explicit). `NewUserFeedback` elements constitute a means for the developer to define new types of user feedback on products within the model. An example of new user feedback on products could be the number of times a customer has watched a product video description through the e-store: the more times watched, the more interested the user is in that product.

In the physical commerce (ubiquitous) scenario, `UbiCARS` app uses two implicit user feedback elements, the `StayingTime` element representing the staying time in front of products and the `Scanning` element representing the scanning of NFC tags of products. `UbiCARS` has zero or one of these elements, as well as a number of custom explicit or implicit `NewUserFeedback` elements to be defined by the user if needed. Figure 9 depicts the DSML elements for the ubiquitous scenario. UbiCARS app specific elements are discussed in detail in Section 5.3.

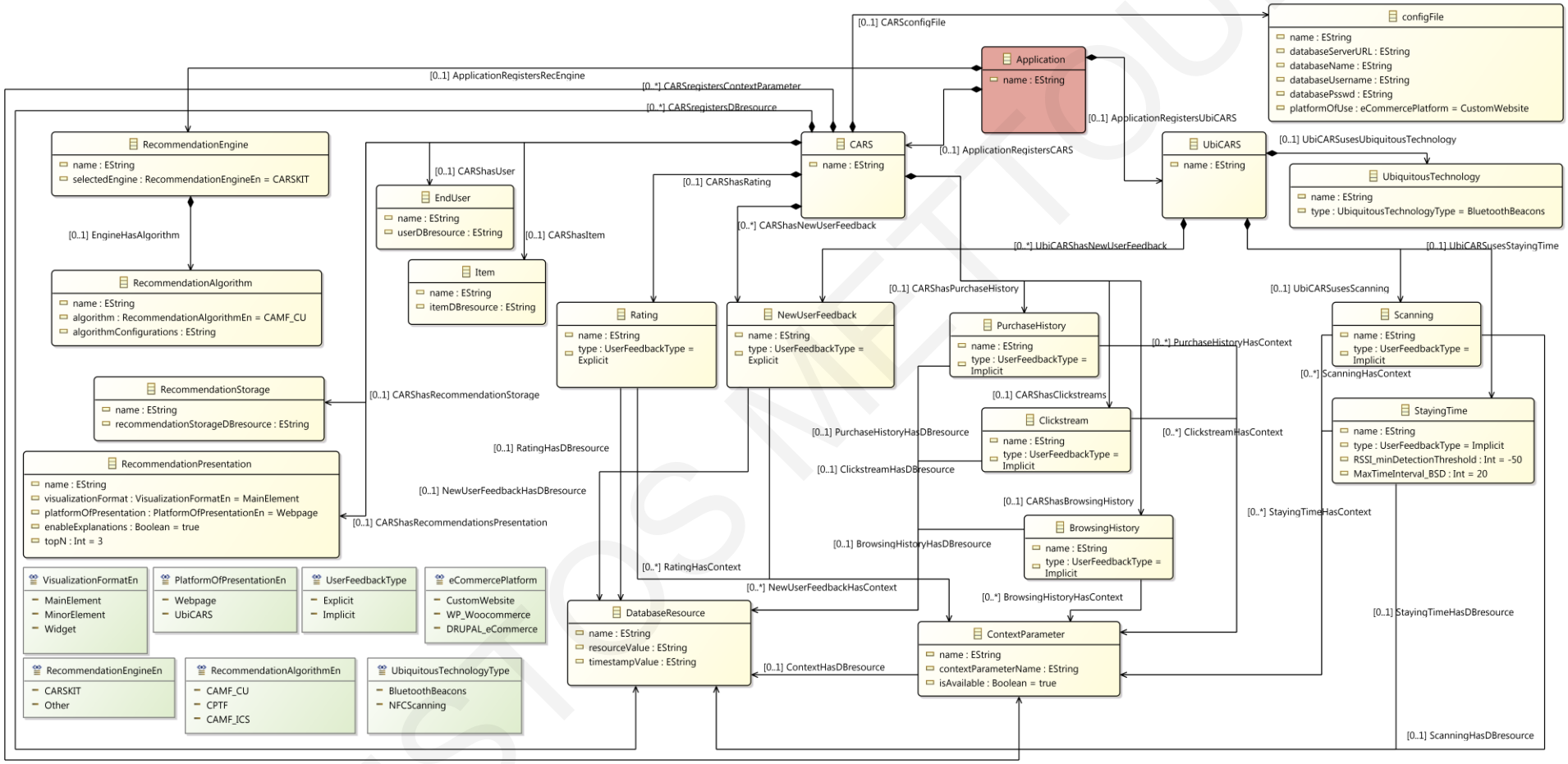

Figure 7: The proposed UbiCARS DSML.

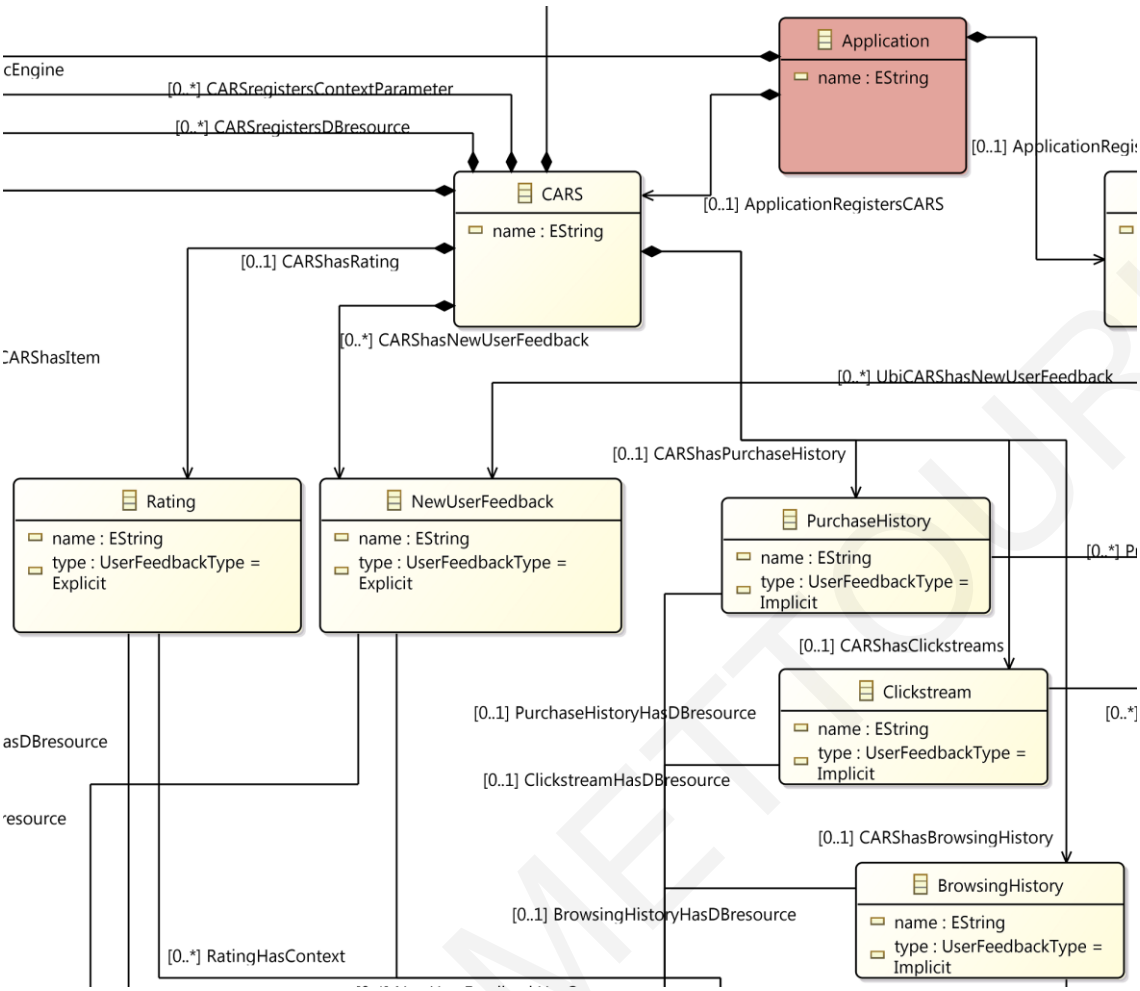

Figure 8: CARS defines Rating as explicit user feedback element and PurchaseHistory, ClickStream and BrowsingHistory as implicit user feedback elements.

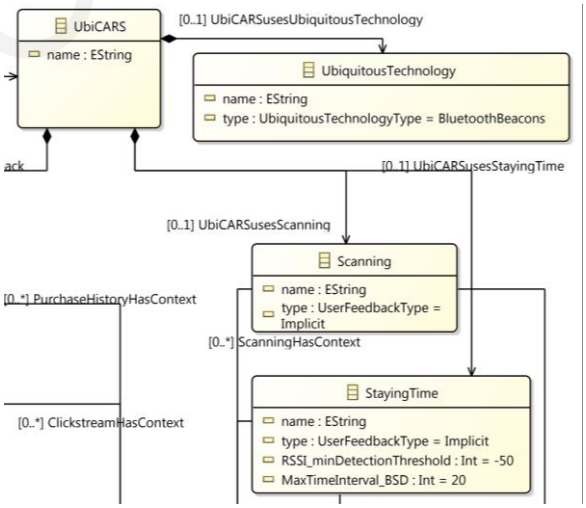

Figure 9: UbiCARS defines StayingTime and Scanning as implicit user feedback elements.

Each of the aforementioned user feedback elements, i.e., `Rating`, `PurchaseHistory`, `ClickStream`, `BrowsingHistory`, `StayingTime`, `Scanning` and

NewUserFeedback uses a DatabaseResource element and may use a ContextParameter (see Figure 10). A DatabaseResource, through parameter resourceValue, specifies information about the database resource where the respective information will be stored and retrieved from. The timestampValue parameter is used to denote whether timestamp is to be used as a time related contextual information, defining thus the exact time the user interaction on the product took place, and use it as context in the recommendation computation. If enabled, the timestamp is automatically embedded in the datasets. If a developer assigns a timestampValue, then, in case this value pre-exists as a database table column, then the system automatically uses it (the developer needs to set the timestamp value to be the same as the corresponding column name); otherwise, the system creates it as a new column to record time.

A ContextParameter captures the context of users while interacting with products. For instance, user location in terms of GPS coordinates, or in terms of any custom defined attributes, such as "store electronics section" or "basement", can be defined. Context is assigned with a Boolean isAvailable to denote whether the corresponding context sensing mechanism is available or needs to be developed. If available, the sensing mechanism can be re-used. A ContextParameter uses a DatabaseResource to denote where the corresponding context will be/is already stored.

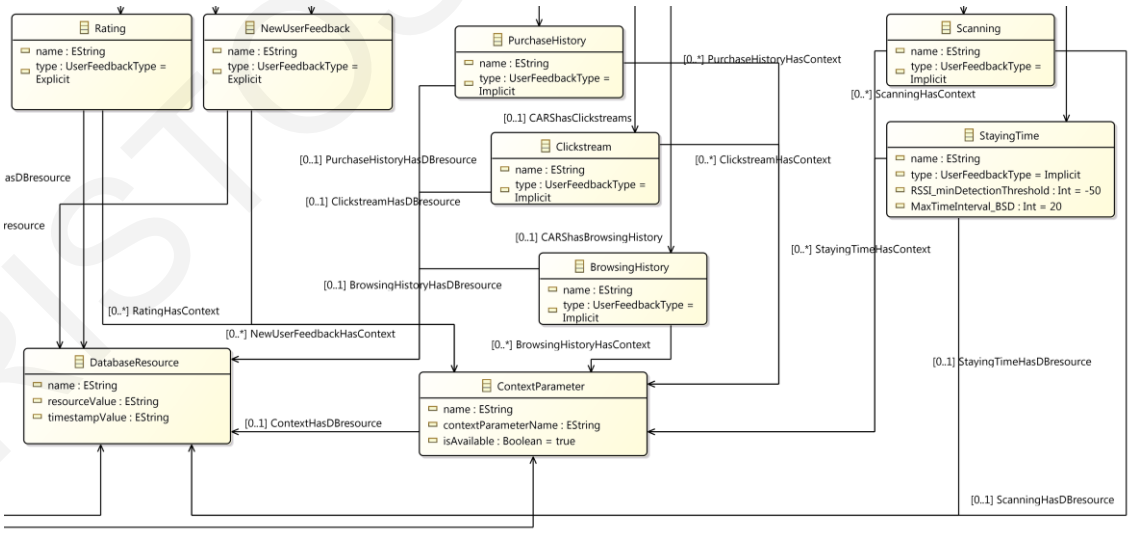

Figure 10: Elements Rating, PurchaseHistory, ClickStream, BrowsingHistory, StayingTime, Scanning and NewUserFeedback use a DatabaseResource and may use a ContextParameter.

The RecommendationEngine element (Figure 11) is responsible for the recommendations computation. The CARSKIT recommendation engine (more information in Section 5.5) is the default engine in the metamodel; however, other recommendation engines can be used by selecting "Other" as the value of the selectedEngine parameter that corresponds to the RecommendationEngineEn enumeration in the metamodel. Please note that, in Figure 7, the parent element of the RecommendationEngine element is the Application element and not CARS, for better projection and availability of the recommendation engine. In specific, this metamodel design decision aims to make the recommendation engine available to be created on canvas from the beginning of the design process, i.e. as the first element to appear on canvas, without the need for a CARS to exist on canvas for it to be created. This better highlights the importance of the recommendation engine.

The recommendation engine uses a RecommendationAlgorithm, which the developer can select from the available algorithms offered by the selected engine. The RecommendatioAlgorithmEn enumeration provides the available recommendation algorithms, which can be selected through the algorithm parameter of the RecommendationAlgorithm element. Provided that the developer has selected CARSKIT to be the recommendation engine, the default algorithm used is context-aware matrix factorisation CAMF_CU, while two more recommendation algorithms are available in the metamodel: CAMF_ICS and CPTF (Tensor Factorisation) [87]. The metamodel can be easily extended with more algorithms that CARSKIT framework offers, by adding more enumerations in RecommendatioAlgorithmEn (see Figure 11). In addition to algorithm selection, relevant algorithmic configurations can also be defined via the algorithmConfigurations parameter of the RecommendationAlgorithm element. Currently, this parameter affects the operation of CARSKIT. In case of use of another recommendation engine, the parameter can be used for the setup of the respective engine.

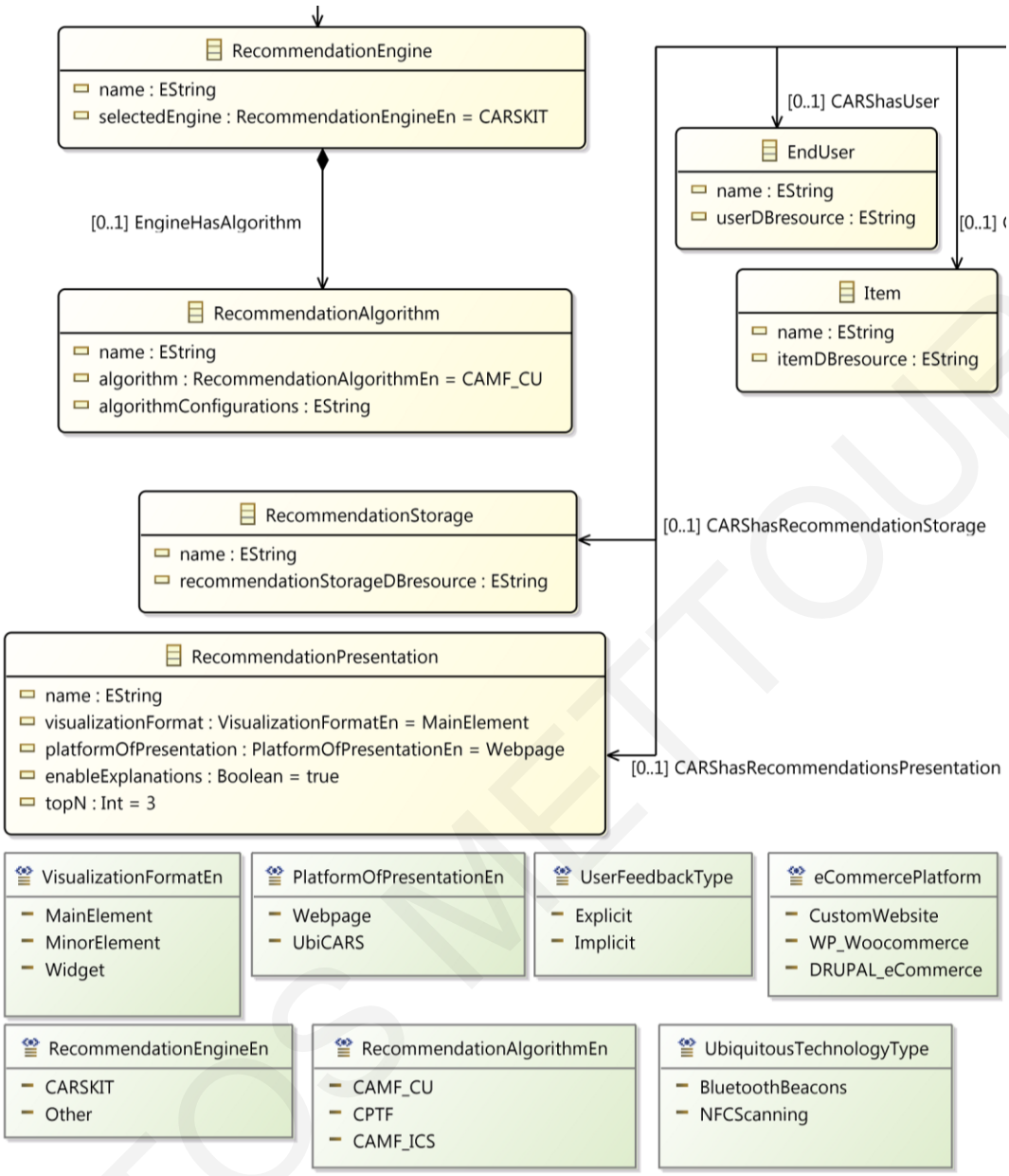

Figure 11: Elements RecommendationEngine, RecommendationAlgorithm, RecommendationStorage and RecommendationPresentation are displayed, together with Enumerations.

While `RecommendationStorage` defines the place in the database, i.e. the database resource where computed recommendations are stored, the `RecommendationPresentation` element denotes, among other, the `platformOfPresentation` of the recommendations to users – whether this would be the e-store via a Webpage or through the smartphone screen via the UbiCARS app (these are defined through the `PlatformOfPresentationEn` enumeration). In addition, the `RecommendationPresentation` element defines the `visualizationFormat` of the recommendations through the `VisualizationFormatEn` enumeration: whether

the recommendations will appear as a main screen element (e.g. as a dedicated to recommendations webpage), a minor one, or as a widget (e.g. in Wordpress).

Another important parameter of `RecommendationPresentation` element is the enablement of explanations for the presented recommendations, via the Boolean parameter `enableExplanations`. Recommendation explanations help users understand the recommendation logic, and can contribute to system transparency, trust, effectiveness, satisfaction, scrutability, persuasiveness, and efficiency, although some of the aforementioned can be contradictive [89, 90, 91, 92]. For example, Amazon.com uses *"Customers Who Bought This Item Also Bought…"*. Recommendation explanations are discussed in Section 2.5. Although the task of providing accurate explanations when complex algorithms are used is difficult, a practitioner can provide a simplistic, generic version of explanations, such as *"These products are suggested to you based on your previous transactions, product ratings and interaction with products in our showroom"*. The `topN` parameter denotes the number of recommendations to be presented (e.g. top-5). It is used by algorithms that solve the top-N recommendation problem, as opposed to those that solve the prediction problem.

5.3 UbiCARS App Specific Elements

The ubiquitous technology to be used by the UbiCARS app is determined by the `UbiquitousTechnology` element, as depicted in Figure 12. Currently, the two supported technologies are `BluetoothBeacons` and `NFCScanning`, as shown by the `UbiquitousTechnologyType` enumeration in Figure 11. It is possible for other technologies to be used as well, such as Wi-Fi [136], or even more innovative ones used for indoor positioning such as *smart floors* [137], although some investment will be required.

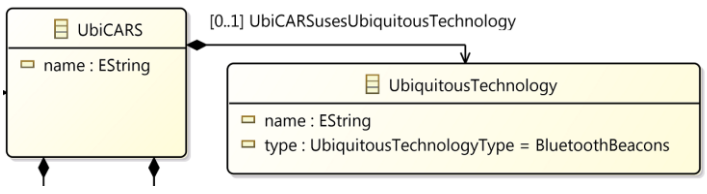

Figure 12: UbiquitousTechnology specifies the ubiquitous technology to be used by the UbiCARS app.

5.3.1 Bluetooth Beacons

Bluetooth beacons enable other Bluetooth devices in close proximity to perform actions. Depending on beacon type, their range may vary from 7m to a few hundreds of meters [138]. Smartphone software can find its relative position to a Bluetooth Beacon in a retail store - retail stores already use beacons for mobile commerce, as beacons can create a more engaging in-store experience for customers [139]. Beacons have also been used for providing users with recommendations [139].

Since products in showrooms are usually positioned relatively close to each other, by placing a Bluetooth beacon on each one of them, overlapping of signal coverage occurs (Figure 13). Hence, customer's staying time in front of a product is calculated by using the Received Signal Strength Indicator (RSSI) of a Bluetooth enabled client (user smartphone) when Bluetooth beacons on products in the proximity are being sensed. No user action is needed other than installing the app and running it.

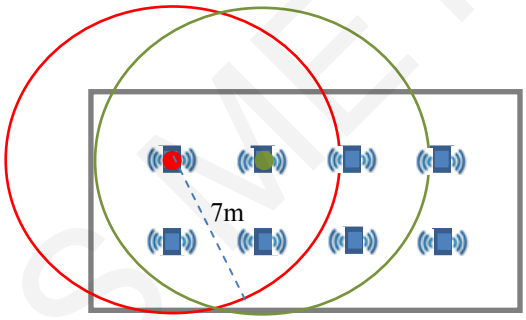

Figure 13: Signal overlapping example: Bluetooth Beacons placed on products in a showroom.

The RSSI acts as a distance indicator between the device and the beacon, as devices closer to beacons register higher RSSI values. Devices scan for beacons continuously, taking about 10-15 seconds between two successive scanning cycles. For each user, based on traces of sensed beacons and corresponding received RSSIs in the available space/showroom, the CARS system calculates an approximation of the distance from the user's device to each Beacon and is thus able to identify the products the user has been staying in front of and for how long (see Listing 4 in Section 5.4). Through the `RSSI_minDetectionThreshold` parameter of the `StayingTime` element (see Figure 9), developers can specify the minimum RSSI value that needs to be sensed by the user's device for the user to be considered that she is close enough to the product and, therefore, that she "has stayed in front of it". After lab experimentation where we have received RSSI values from -30 (corresponding approximately to 2m) down to -120

(corresponding to approximately 8m), we have specified the default value for the `RSSI_minDetectionThreshold` to be -50. Note that sensing RSSI values depends on the sensors used and room specifics, and is also sensitive to obstacles: therefore, experimentation and adjustments will be needed in each specific use case scenario. Moreover, in our experiments, smartphones were mostly held in users' hands; in cases where smartphones were held in pockets and bags (which is a realistic scenario), reduced received signal strength was observed. The system offers flexibility in terms of using the RSSI as a distance indicator between a user's mobile device and a beacon, as lower RSSI values can be configured to be acceptable in order to compensate for reduced signal strength reception for such cases. The downside is that incorrect user detections in front of products may be observed in cases where the user is not standing directly in front of the product but is, instead, near-by the product.

The `MaxTimeInterval_BSD` parameter specifies the maximum time interval between successive detections of a user in front of a product, in order for these detections to be considered in the same "staying time session" in front of that product. Default value is 20, meaning that successive detections of a user device in front of a product beacon when equal to or less than 20 seconds pass between each detection, result in the system adding the total time and attributing it to the same session. Hence, if the user goes away from product X for more than 20 seconds (e.g. to explore other products) and thus stops focusing on product X, but shortly after returns to product X, the system initiates a new "staying time" session for the user and product X.

Using the abovementioned default values, the staying time reasoning algorithm functions as follows (Figure 14): when a user mobile device senses the same beacon for more than once and each successive beacon sensing has a time difference of less than 20 seconds with the previous one, and furthermore at all times the RSSI value is higher than -50, then the system adds the corresponding time difference between the first and the last sensing and assigns the sum as the staying time of the user in front of that product. The above algorithm ensures that, in case a user passes by a product and walks away, no staying time will be assigned to her for the corresponding product.

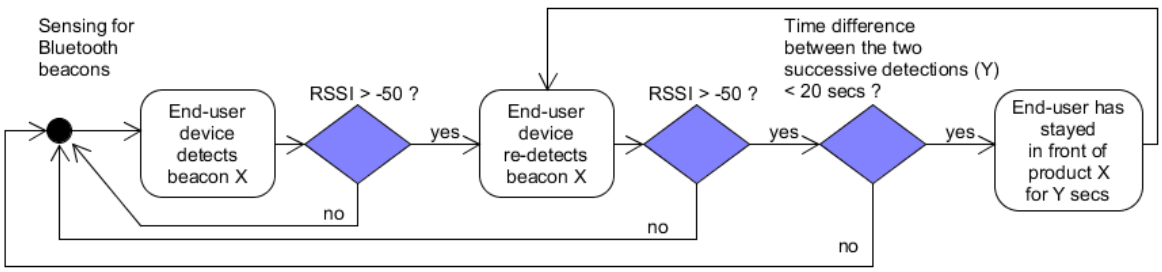

Figure 14: Staying Time Reasoning Algorithm.

The staying time reasoning algorithm is executed by the Logic layer of the framework, after receiving the configuration parameters `RSSI_minDetectionThreshold` and `MaxTimeInterval_BSD` from the Configuration layer.

Privacy Concerns

In terms of privacy, during indoor positioning, location tracking is done by the smartphone software and not by the Bluetooth beacon; however, the data are transferred to the CARS system and stored in the server's database to be exploited by the recommendation engine. Hence, privacy-preserving methods are required to ensure users' privacy. The subject of privacy, although critical for all systems that utilise users' private data, it has not been addressed in the context of this work and is left as future work. For the purposes of using the UbiCARS app, user consent is required, as user's indoor positioning data are utilised by the framework.

5.3.2 NFC Scanning

Among the most often used techniques on identifying products that are of user interest in the physical shopping scenario, are manually typing in a barcode or the product name, automatic barcode recognition, Near Field Communication (NFC) and RFID [121]. NFC is the fastest method and achieves the highest perceived ease of use [121, 140]. These approaches require the user being close to the product, some even to pick it up. NFC requires the user to hold the mobile device in a 5 cm range from the product and it requires on average 3.3 seconds for detecting the product [121]. According to the authors in [121], considering the thousands of products in supermarket shelves, this highly manual interaction is tedious and unfeasible, except when only information for a significantly small number of products is required.

There have been reported examples of stores in the market that have deployed NFC technology to enable customers to easily purchase products or read information about them [141]. Casino supermarket for example has used NFC tags in front of every product on the shop shelves [141]. Customers can bring their smartphones near the tag to view product information or add the product to their mobile app's basket. The mobile app basket allows for physical checkout. Another example is Harvey Nichols that have provided their customers with tablets in-store so that they could interact with products via NFC tags on shelves [141]. In Argos, customers can use their NFC-enabled smartphones in-store to download information about the retailer's products and download the Argos transactional mobile app [141]. In [142], the NFC forum's white paper "NFC Technology: How

Changing Consumer Preferences Create New Opportunities for Retailers" is mentioned, in which data was analysed relating to user preferences and perceptions of NFC in a retail setting. Consumers were particularly excited to use NFC to access timely and relevant information, such as information about related products, store deals and multimedia content. NFC was preferred by a significant margin over QR codes, Bluetooth Beacons, and browsers in all of these use cases. The study also revealed among other that over 75% of respondents who used NFC were very satisfied with their experience, while more than 75% of respondents indicated an interest in viewing additional product information including inventory and availability via NFC.

It is the case, however, that scanning NFC tags requires users to actively use their phone to scan an NFC tag attached on a product. Users can be motivated to scan an NFC tag of a product to find out more information on that product, visit its webpage, be recommended of similar products and read reviews [2, 16, 20, 118, 119, 121].

User's staying time in front of a product is a truly implicit user feedback since it happens without requiring user's active participation and without interfering with user's task (user's consent needs to be granted at a prior stage). Having users scan NFC tags of products, however, needs user participation, similar to user ratings, as well as it interrupts users from their task. Since the motive of a user that scans a product is to read more information about it, when a scan occurs we can be certain that the user has shown some interest on the product, but we cannot be certain that she actually likes it. The same applies for staying in front of a product: the user may as well be looking at something else other than the product in front of her. Nevertheless, implicit feedback is important and can replace explicit feedback techniques where the latter cannot be used.

5.3.3 Deploying Bluetooth Beacons & NFC Tags in a Physical Store

This section discusses deploying the ubiquitous technologies supported by the UbiCARS MDD Framework in a physical store showroom.

The use of Bluetooth beacons assumes a similar product layout to the one depicted by Figure 13: one beacon should be available for each product on display, while products should not be placed directly next to each other. Signal range of Bluetooth beacons need not necessarily be 7m, although beacons of range close to 10m will provide better accuracy results than beacons of higher range (e.g. 100m), as the latter will be detectable by smartphones that may not be in close proximity to the corresponding products.

In case NFC tags are used, one tag for each product on display is required. Product layout within the showroom can be more relaxed and flexible, since NFC requires users being close to the product and holding the mobile device in a 5 cm range from the product; thus, products can be placed next to each other.

Bluetooth beacons require a configuration step by developers in order to map Bluetooth hardware address of beacons to specific product IDs (e.g. "7C:91:22:64:CE:11" maps to product ID "25"), so that the staying time reasoning algorithm can correctly assign users' staying time on products. The mapping is stored in a database table that is created by the UbiCARS configuration process; developers only need to include the mapping data in the table. This can be conducted, either by directly accessing the database to insert the values through a database management tool (e.g. PHPMyAdmin), or by utilising a script offered by the framework where values can be included in a text file.

In terms of using NFC tags, we propose pre-programming each NFC tag as follows: when a mobile device scans the tag of a product, it directly accesses a specific web page for that product, where server-side code registers the user-product interaction. Programming of NFC tags can be easily conducted via using relatively cheap specialised devices ("NFC writers"), or even through smartphones via specific applications.

5.4 Enhancing Recommendation Accuracy and Availability

Assuming a retail business with an e-commerce website and physical showrooms with products, our solution proposes that a UbiCARS app operating in the physical showrooms is used together with a CARS system that operates on the e-store, to enhance the overall recommendation accuracy and availability (in [103] it is shown that users' diverse implicit feedback data can be used to improve recommendation accuracy, whereas in [108], user's implicit feedback data in terms of purchases, together with explicit feedback data in terms of user clicks on products, have also shown to improve recommendation accuracy; see Section 3.1 for both papers). User's implicit feedback data acquired within the physical store complement user's explicit and implicit feedback data in the online scenario to enhance user preferences modelling and improve the overall recommendations for that user. In addition to e-commerce methods for tracking user behaviour (user purchase history, clickstream data and browser history information), we enable tracking of user's ubiquitous behaviour in real-time while in-store, aiming to acquire the ubiquitous user-item

interaction (Figure 15). We propose utilising the "Staying Time in front of a product" and the "NFC tag Scanning of a product".

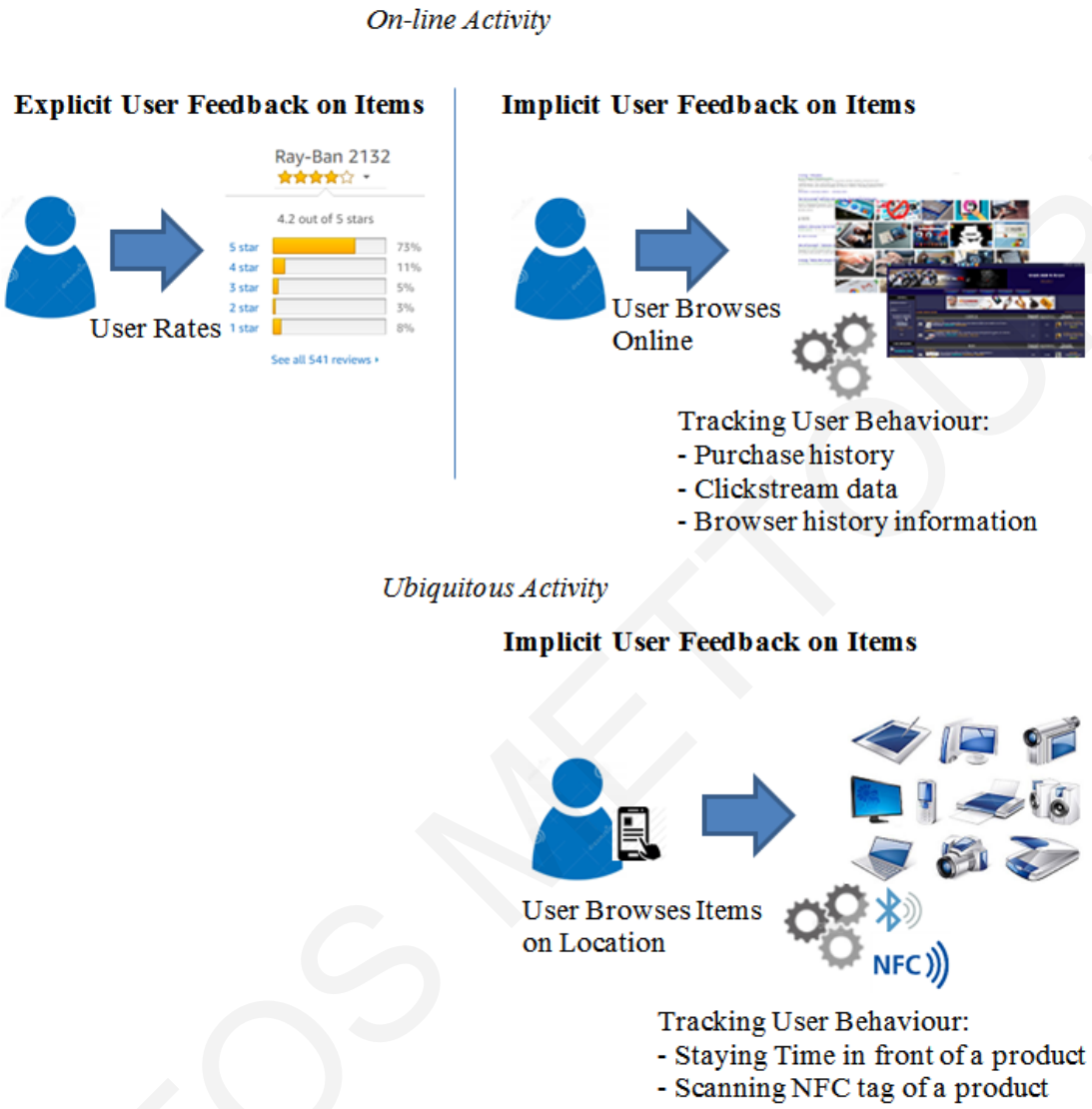

Figure 15: Modelling user preferences in the online and ubiquitous scenarios.

At this point we note that user ratings, as explicit user feedback technique, can occur in a similar manner in both the online scenario (user rates items via a browser) and the ubiquitous scenario (user rates items using his smartphone through the UbiCARS app – although such interaction is not accustomed). In our work, we do not distinguish ratings from the ubiquitous scenario, as, together with the online ratings, they both contribute to the same user ratings dataset. Similarly, while our approach proposes using online user purchase history as an implicit user feedback acquisition technique from the online scenario, user purchases of products in the physical store can also be considered, if these are included together with user purchases in the e-store. In such a setting, one purchase history dataset will be created, including user purchases from both scenarios. Based on the aforementioned, we state that, in our work, we distinguish user feedback acquisition

techniques for the ubiquitous scenario only where the corresponding interaction cannot occur in the online scenario (users' staying time in front of items, users scanning items), as only these techniques offer additional, complementary datasets to the recommendation process.

In case there is a need to distinguish between user purchases and ratings acquired from each of the two scenarios and use all four datasets in the recommendation computation, the UbiCARS Framework can facilitate such a setting through the UbiCARS DSML, by utilising two `NewUserFeedback` elements, one for the ratings and one for the purchase history of the physical store scenario, both linked to the `UbiCARS` element.

UbiCARS Example of Use Revisited

We discuss in detail the example of use described in Section 1.1.

RSs are most helpful in large shops with many product categories where the customer base is likely to be very heterogeneous [2]. We consider an electronics store with a large variety of electronic devices and peripherals that require a level of knowledge from customers in order to purchase the best suited product for their needs and preferences. The store showroom is equipped with Bluetooth beacons placed on products to track user interaction with products via a UbiCARS mobile app.

Nicolas is a regular user of the SmarTech e-store; he has purchased a few products in the past and also rated some of them online. When Nicolas uses the SmarTech e-store, he likes the personalised product recommendations the website offers, as most of the times, the recommended products match his preferences, so he checks them out. Nicolas is now at a SmarTech physical store showroom browsing the products. He is quite interested in technology products and, in particular, in high-end laptops. SmarTech includes only a limited number of products in its showroom, compared to the entire catalogue online. Nicolas approaches a laptop on a shelf that has caught his attention. As he reads a few specifications on the laptop from the small label, he thinks he would like to know more information on its features, as well as to find more products he might be interested in. Nicolas opens the SmarTech UbiCARS app on his smartphone. The app receives signals from the Bluetooth beacons and sends related info to the server which (i) identifies the product Nicolas is in front of and (ii) computes Nicolas *staying time in front of the product*. The SmarTech UbiCARS app recommends to Nicolas a number of product recommendations: two of the recommended products are currently in the showroom, so he checks them out next. Another recommended product is located in another SmarTech showroom downtown; Nicolas thinks he can probably stop by the showroom tomorrow

afternoon after work, as he likes to physically check out the devices before buying. The fourth recommendation is for a product that is available only through the SmarTech e-store; Nicolas can check that product from the convenience of his home PC. Nicolas finds it informative and entertaining to be able to receive recommendations while in the store.

For Nicolas the e-store has recorded a number of ratings on products (explicit feedback) and has tracked his online interaction with products: products browsing history and online purchase history (implicit feedback). From such data, datasets are compiled to be used in the recommendation process, as shown in Listing 1, Listing 2 and Listing 3. Browsing history refers to the number of clicks on the product's name or icon in webpages where many products are listed or recommended, as well as the number of accesses in the product's webpage, while purchase history refers to whether the user has purchased the corresponding product.

The UbiCARS app contributes to the recommendation process by sensing Nicolas feedback data from the physical store, enabling thus the framework to compute the time (in seconds) Nicolas has stayed in front of a product, as displayed in Listing 4. After Nicolas visit to the store, the *Staying time dataset* will be available to the CARS system for use in the next recommendation computation, in addition to the datasets in Listing 1, Listing 2 and Listing 3. Recommendations for Nicolas the next time he will visit one of the SmarTech physical stores or the SmarTech e-store can be more accurate, since more relevant data in terms of Nicolas' feedback will be used during their computation.

In terms of recommendation availability, since each explicit and implicit user feedback technique provides a new dataset for recommendation computation, and thus, an alternative way for computing recommendations, it is stated that the UbiCARS methodology increases recommendation availability. In the abovementioned example of use, recommendations can be computed by using any one of the following:

- The Ratings dataset.
- The Purchase History dataset.
- The Browsing History dataset.
- The Staying Time dataset.

Usage of four datasets instead of one (e.g. the ratings dataset) enables for increased availability, as recommendations can be computed with each one of them. While problems may exist when using implicit user feedback on products [5, 105], it is nevertheless important and can replace or complement explicit feedback techniques.

Listing 1: Ratings Dataset

userID	itemid	rating	Day	Time
1	15	4	Weekday	Morning
5	24	5	Weekend	Noon
5	25	2	Weekday	Afternoon

Listing 2: Browsing History Dataset

userID	itemid	AccessedOnline	Day	Time
5	24	1	Weekday	Noon
4	24	1	Weekend	Morning
1	25	1	Weekend	Evening

Listing 3: Purchasing History Dataset

userID	itemid	purchasedOnline	Day	Time
5	28	1	Weekday	Afternoon
1	15	1	Weekday	Morning
2	64	1	Weekday	Afternoon

Listing 4: Staying Time Dataset

userID	itemid	StayedInFrontOf	Day	Time
1	25	231	Weekday	Afternoon
3	29	38	Weekend	Evening
5	25	178	Weekday	Afternoon

Listing 5: Staying Time Dataset with Context

user ID	item id	StayedIn FrontOf	Day	Time	Context
1	25	231	Weekday	Afternoon	ElectronicsSection: AndroidProducts
3	29	38	Weekend	Evening	ElectronicsSection: Consumables
5	25	178	Weekday	Afternoon	ElectronicsSection: Laptops

5.5 UbiCARS Framework Architecture

In Section 5.1, the UbiCARS app and CARS system were described as a mobile application and server-side system that enable tracking of user interaction with products on-location and on-line respectively, as well as displaying product recommendations to users on-location and through the e-store respectively. Moreover, the CARS system acts on the e-store and conducts all necessary system configurations, while it is also responsible for context-aware recommendation computation by configuring and running the CARSKIT recommendation engine.

The framework architecture is shown in Figure 16. The online store frontend, acting on the client side, is able to track user behaviour on items through the browser software and/or 3rd party software running on the browser. Such behaviour consists of tracking user clicks on parts of e-store webpages, such as clicks on product images and descriptions.

Server side tracking of user behaviour online is achieved by tracking the specific product webpages that the user visits, as well as analysing server log files.

Ubiquitous user-product interaction is tracked through the UbiCARS app, either via Bluetooth beacons (estimating users' staying time in front of an item) or NFC tagging. The framework is possible to be extended with more ubiquitous technologies, as discussed at the beginning of Section 5.3.

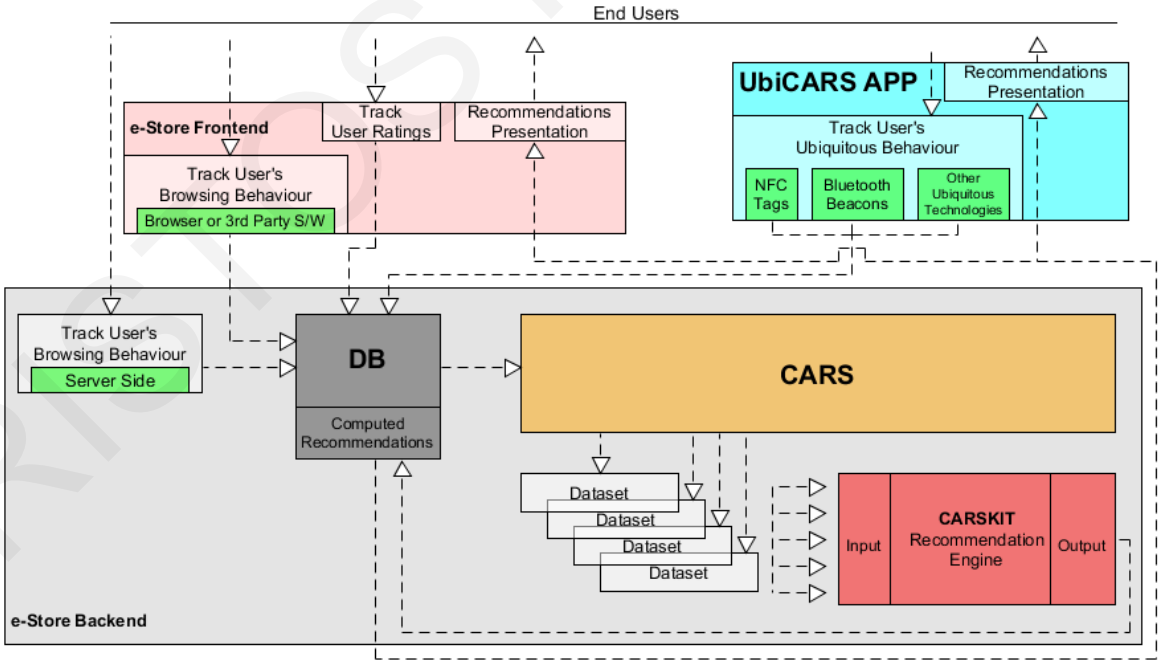

Figure 16: Framework Architecture.

All user feedback data on products are stored in the database (DB). Such data are retrieved by the CARS system to compile the corresponding datasets and feed them to the recommendation engine, i.e. CARSKIT. Computed recommendations are stored in the

database. Recommendations are presented through the e-store front-end and the UbiCARS app.

CARSKIT [87] is a Java-based, open source Context-aware Recommendation Engine that offers state-of-the-art context-aware recommendation algorithms. CARSKIT has been selected among other recommendation frameworks due to the many and efficient context-aware recommendation algorithms it offers, its ease of use, and the flexibility with which it can be fine-tuned to work with multidimensional datasets. It is important to note that, among the frameworks discussed in Section 3.3, CARSKIT is the only one that offers context-aware recommendation algorithms.

Figure 17 depicts the activity diagram of the framework with a clear representation of the two user roles: the developer and the user/customer. The framework tracks user activities (as a customer) as shown in the figure, as well as the context of such activities, provided that the developer has included in her model the corresponding `ContextParameter` model elements. For example, consider the need to track users' location in the physical store while tracking their staying time in front of products to enable context-aware computation of recommendations. A `ContextParameter` model element for location is linked to the `StayingTime` model element, while a `DatabaseResource` element is also created and linked to the `ContextParameter` element, to specify where the context information will be stored and retrieved from. With the above design, the framework becomes aware that the staying time dataset is context-aware in terms of location information. Implementation of context sensing mechanisms (context plugins) for users' location is undertaken by the developer; however, the entire infrastructure in terms of context data storage, the inclusion of context in the datasets, the compilation of the context-aware datasets and the computation of context-aware recommendations is undertaken by the framework. The implementation required by the developer includes sensing of context data and storing them in a specific database table that has been created by the framework.

Cross-model reuse of context-aware plugins is provided, in the sense that any developed plugin can be reused across models by developers. Such plugins are represented in the model through the `ContextParameter` element via the parameter `isAvailable`, which indicates whether the corresponding context sensing mechanism is already available and hence can be used, or whether it needs to be developed.

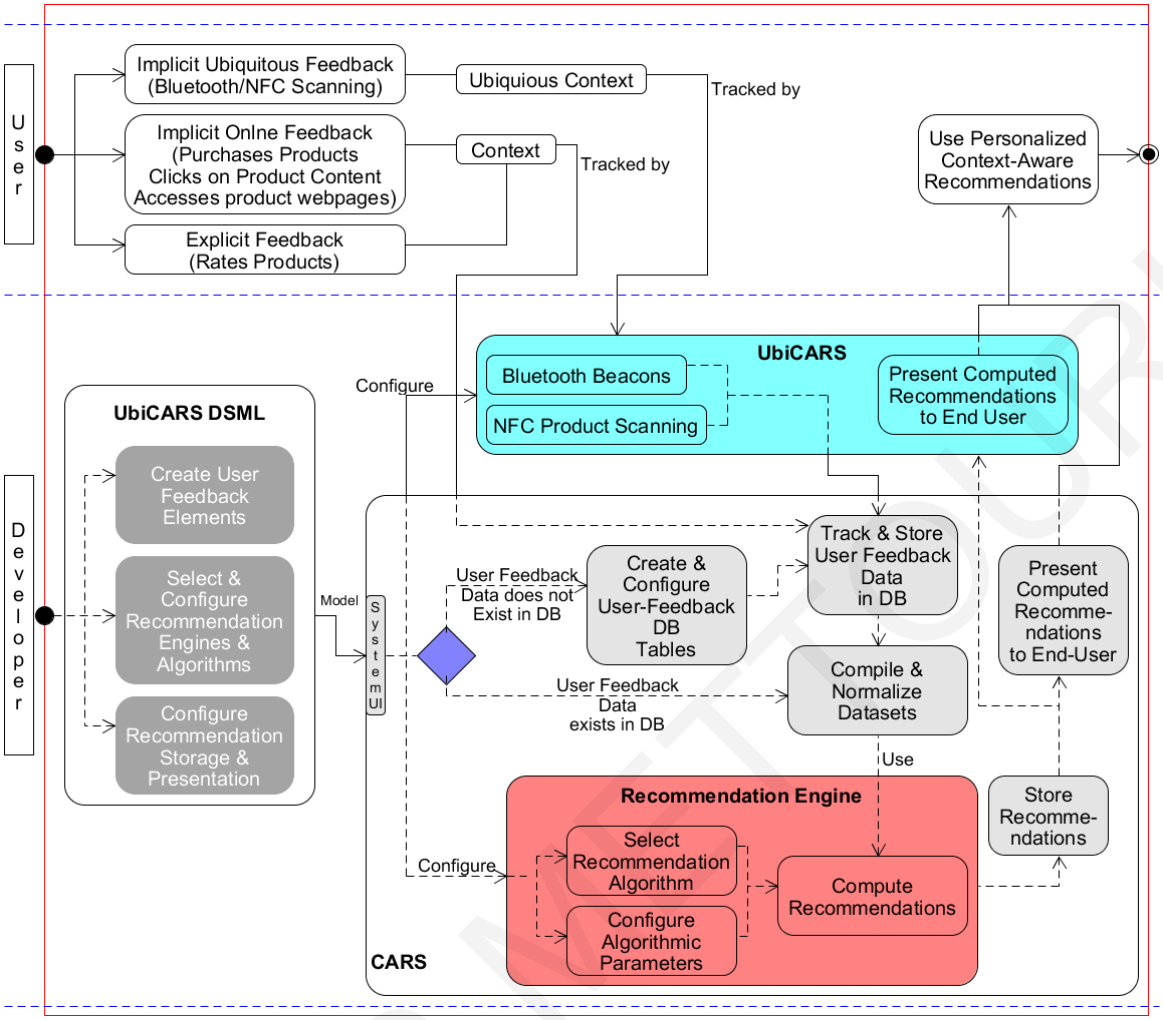

Figure 17: Activity diagram for the two user roles.

From the developer's perspective, interaction with the system is made mainly through the UbiCARS DSML, where the developer can create elements and configure engines and plugins. Once the model is created, through the system UI, the framework is fed with this model as input to initiate configurations that include, not only creation of the necessary information infrastructure, but also configuration of functional modules on the e-store (plugins) to acquire user feedback data through the e-store, as well as configuration of the UbiCARS app in terms of the ubiquitous technology to be used to acquire user feedback data from the physical store. The UbiCARS app is dynamically configured by retrieving configuration settings from the database. These settings were included in the database by the framework after model parsing. Configurations of other ubiquitous parameters, i.e. `RSSI_minDetectionThreshold` and `MaxTimeInterval_BSD` are conducted server-side by the Logic layer of the framework, as described in Section 5.3.1 .

Configuration is also conducted on the recommendation engine and algorithm. Once recommendations are produced, the framework presents those to the end-user (customer) through appropriate interfaces in the e-store and UbiCARS app.

Figure 18 depicts the parsing and configuration steps followed by the UbiCARS Framework. Input to this process is the model designed via the UbiCARS modelling editor using the UbiCARS DSML. Configuration steps from the start, down to the dotted line, regard configurations conducted on the e-store platform by the CARS system in terms of database resources from user feedback elements and their context elements (if any). The dotted line signifies the point during execution where the framework inspects whether the respective data from user-product interaction exist in the platform. If such data do not exist, execution steps below the dotted line cannot be realised (this refers to the cold start problem, an inherent problem of RSs which dictates that, for RSs to be able to produce meaningful recommendations, user interaction with items first needs to take place). In case data exist, the framework proceeds with those to compile the corresponding datasets, compute recommendations, and present them to users. For example, if user ratings on products exist, but users did not visit the physical store, and therefore, user staying time data do not exist, the framework will proceed to compile the ratings dataset and use that for the computation of recommendations. The developer may at any time initiate compilation of any dataset, provided that the aforementioned requirements apply.

In case context database resources are defined in the model but context data and resources do not yet exist in the database, the context database resources need to be created. In this case, context data are not yet available, and thus, non-context-aware execution is initiated (see Figure 18). For example, while context parameters for user location tracking may have been defined in the model during design, context plugins for implementing tracking of user location may have not yet been developed. However, during following system iterations where context resources and the respective context data will have been included in the database, context-aware datasets are compiled and context-aware recommendation computation is initiated.

Chapter 5 presented the UbiCARS MDD Framework in terms of its methodology, the DSML and the framework architecture. In Chapter 6, instance models and a demonstrator of the framework are presented, as well as system implementation and evaluation setup. Chapter 6 proceeds then with description of the evaluation methodology and evaluation results.

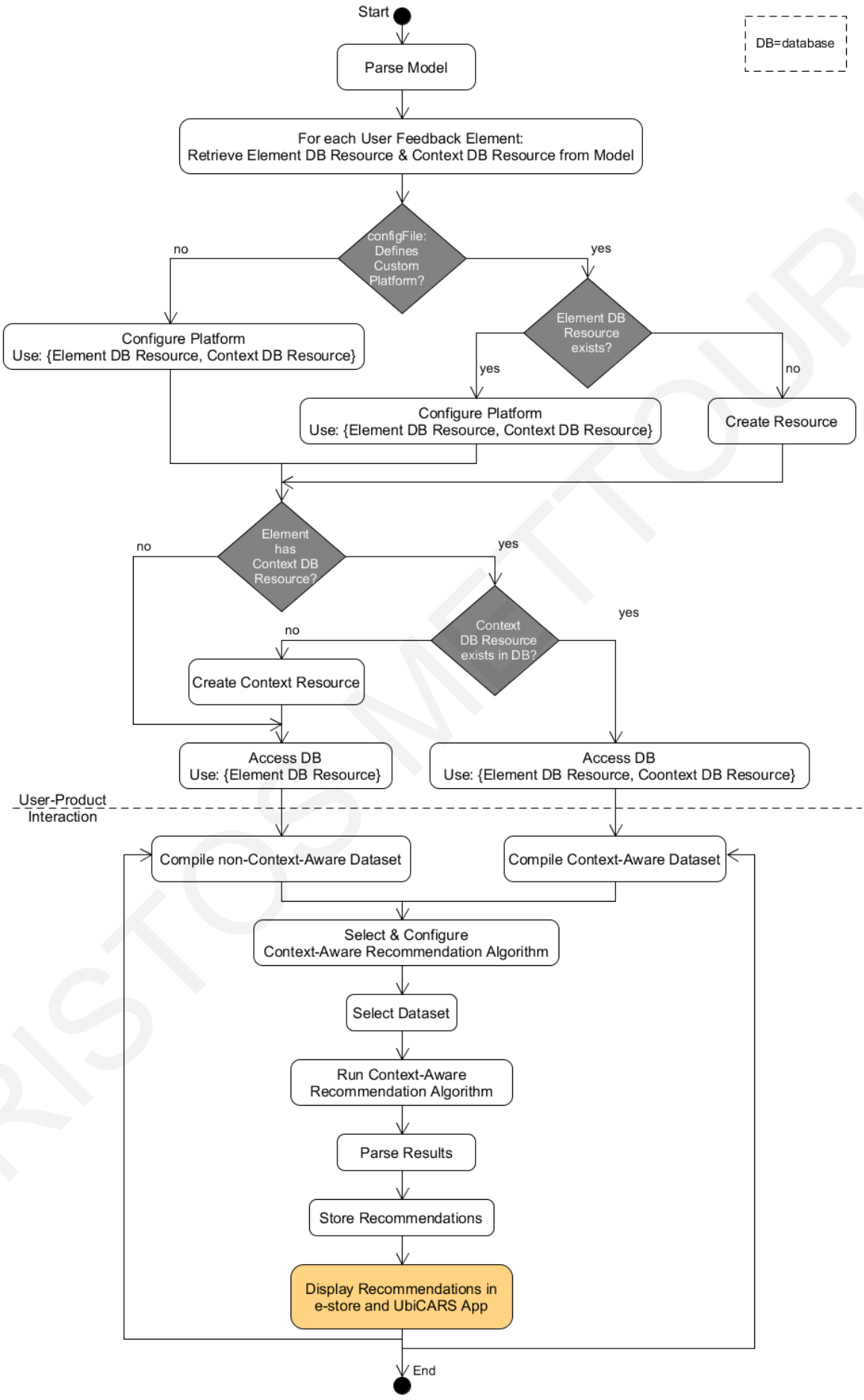

Figure 18: UbiCARS Framework configuration steps.

Chapter **6**

UbiCARS Framework Demonstrator & Evaluation

6.1 UbiCARS Framework Demonstrator

The UbiCARS MDD framework provides practitioners with a toolset to model and configure a UbiCARS app and a CARS system, as these were described in Section 5.1. A *UbiCARS model* can be seen in Figure 19. To avoid confusion in terms of the UbiCARS terminology, we state that the UbiCARS model is the model produced as outcome of usage of the UbiCARS MDD framework that includes the elements of CARS system and UbiCARS app.

The modelling toolset is comprised of the DSML along with its modelling editor. Figure 20 shows the editor, its toolbox and a part of a model shown within the editor. The available tools are placed in the toolbox named "MyTools" situated to the right hand side of the editor. To add an Element in the editor a user needs to click on the respective item in the toolbox and then click in the white space in the editor, or drug-and-drop the element from the respective item toolbox to the canvas. Through the properties view (shown in Figure 21), information can be added/edited about the selected element.

As the full model instance in Figure 19 shows, there are 4 types of Elements:

- Dark Blue: CARS and UbiCARS are the starting points.
- Light Blue: main elements of a model, including all user feedback elements, as well as the ubiquitous technology, recommendation engine, algorithm, storage and presentation elements.
- Yellow: Database Resource elements.
- Orange: Context elements.

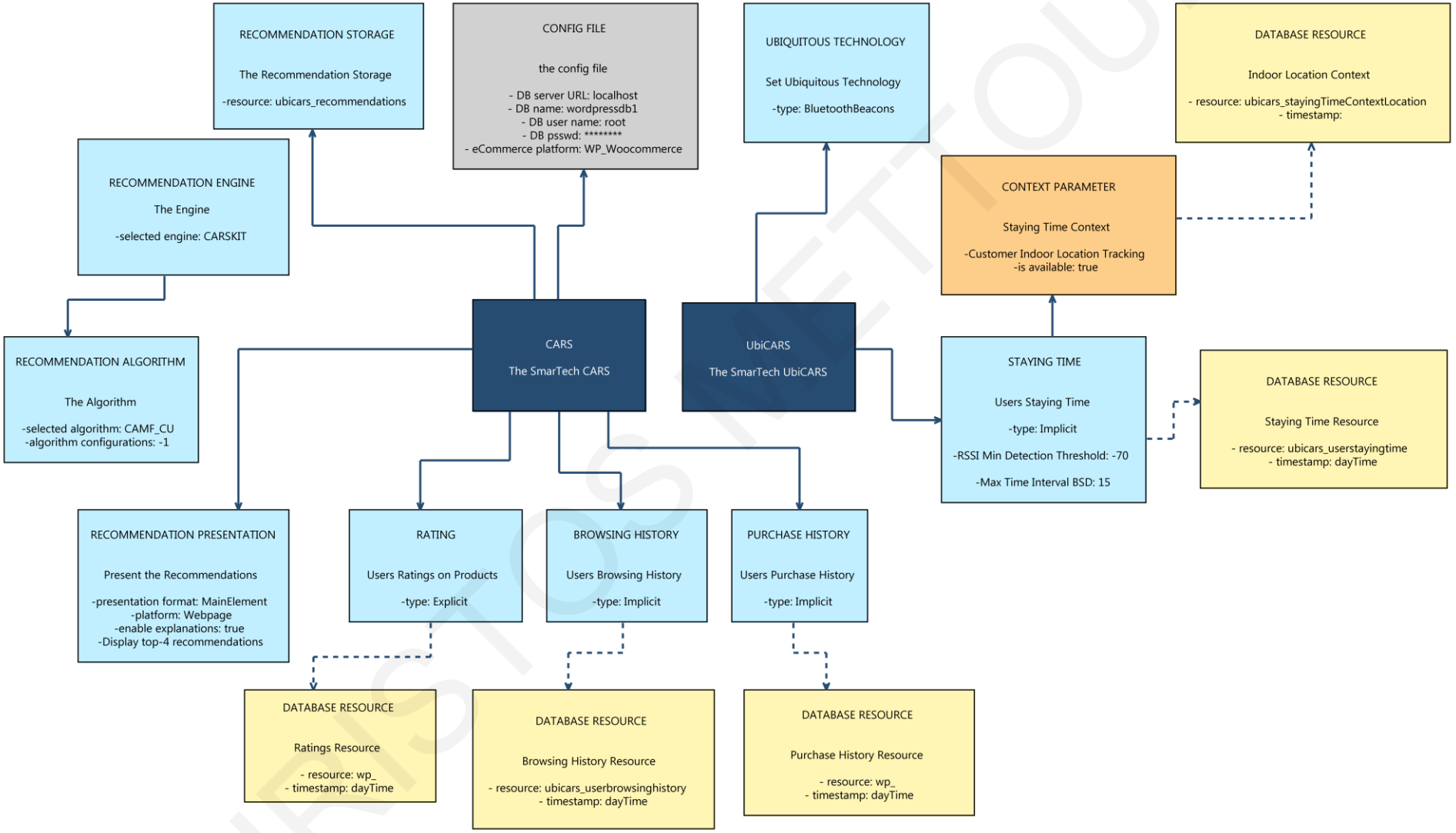

Figure 19: UbiCARS Model Instance.

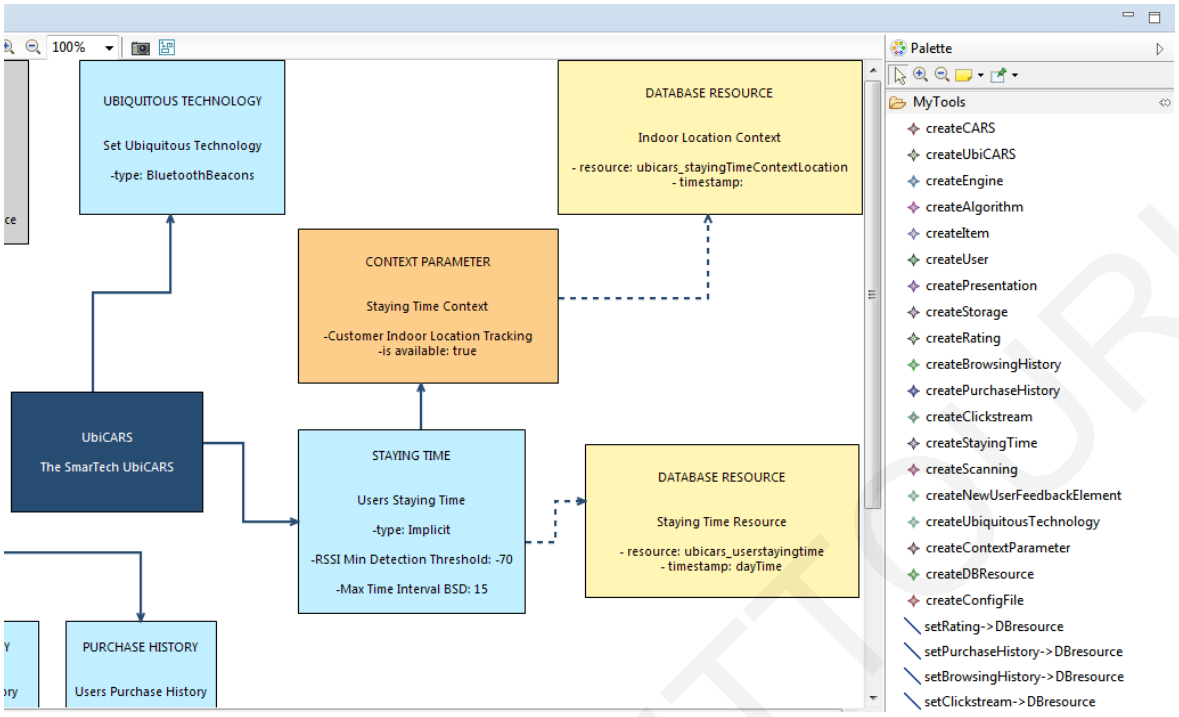

Figure 20: UbiCARS Modelling Editor, Toolbox and Model Instance.

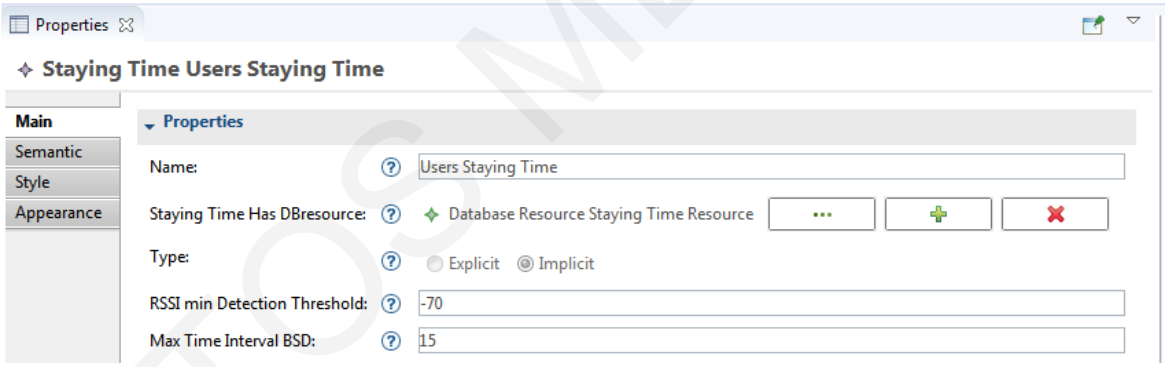

Figure 21: Properties View of Staying Time Element within the UbiCARS Modelling Editor.

Regarding the blue elements (dark and light), users cannot add more than one element of the same type, e.g. two CARS or two Recommendation Engine elements. Moreover, for the light blue elements, it applies that they have a one-to-one correspondence with their parent elements (e.g. there is only one CARS element for Rating element and vice-versa); hence, they are at all times linked together. Therefore, when a light blue element is added on the canvas, it is automatically linked to its parent element, with the exception of recommendation engine element that does not have a parent (it is independent).

The value of an element's parameter can be edited from the "Properties" window, as depicted in Figure 21 for the STAYING TIME element. Legal characters are only letters

and numbers, unless stated otherwise, such as for RSSI min Detection Threshold and max Time Interval BSD parameters in the same figure that require numbers.

To link an element *ElementA* to another element *ElementB*, users need to click on the respective edge in the toolbox that names the two types of elements as follows: setElementA->ElementB. Then, they need to click first on ElementA and then on ElementB. The other way around is not possible. As stated before, elements in a one-to-one correspondence with other elements are already linked together during element creation. Users only need to specifically link together:

- User feedback main elements with database resource elements.
- User feedback main elements with context elements.
- Context elements with database resource elements.

To delete an element or an edge, users click on the element or edge and then press delete on their keyboard.

The model elements displayed in Figure 20 define the `SmarTech UbiCARS` app that uses `BluetoothBeacons` technology and a `STAYING TIME` element as implicit user feedback on products, which tracks customer-product interaction in the physical store and engineers it into a dataset for the CARS system to use in the recommendation process. The `Staying Time Context` adds `Customer Indoor Location Tracking` as a context parameter to each customer-product interaction (e.g., such context could constitute the store department where the interaction takes place). Listing 5 shows a snippet of such a dataset for staying time with context.

Note that datasets where user feedback values are not scaled as numbers from 1 to 5 (here the `StayedInFronfOf` in Listing 5), need to be properly scaled before used in the recommendation engine. The number of seconds can be scaled from 1 to 5 indicating five levels: 1 corresponding to minimum staying time and 5 to maximum (these scales need to be defined by the developer, depending on the use case scenario).

System implementation considers two of the most well-known, open source e-commerce platforms, WordPress WooCommerce [143] and Drupal Commerce [144], as well as custom platforms. Practitioners may specify their platform of choice in the model to drive system configuration via the `platformOfUse` parameter of the `configFile` element (see Figure 19), which uses the enumeration `eCommercePlatform` (see Figure 11). Possible values for `eCommercePlatform` are `WP_Woocommerce`, `DRUPAL_eCommerce` and `CustomWebsite`. The framework provides code snippets and functional modules that can be installed on the corresponding platform and provide the

following functionality: retrieve explicit user feedback in terms of user ratings on products, retrieve implicit user feedback in terms of user purchase history and browsing history (product webpages accessed) on the website, as well as display computed recommendations accordingly, as specified in the model. Integration with the UbiCARS mobile app is provided.

For testing and evaluation purposes, we have set up one example for each e-store platform (WooCommerce and Drupal Commerce), in which we have included electronic products, as these can be found in a real e-store. Since the cold start problem affects system execution, in the sense that, after system configuration has completed, the CARS system is not able to produce recommendations until customers have interacted with the electronic and physical stores, lab personnel has interacted with the e-stores to produce user feedback data. This interaction meant to simulate regular user activity on an e-store, such as browsing products, purchasing products and rating them. To simulate a physical store showroom, we have used a similar layout to the one in Figure 13 to represent a mobile smartphone showroom. The products were Bluetooth enabled Android and iOS smartphones "for sale", eliminating thus, in this case, the need for Bluetooth beacons. Beacons are expected to be more accurate than smartphones; however, when Bluetooth is embedded in the products for sale, budget (for beacons) can be saved.

It should be noted that, in case customer interaction pre-exists in a platform prior to system configuration, the system integrates and uses it, if relevant. For instance, in case customers have already rated products in WooCommerce before the system takes effect, the latter will consider such data during configuration and use it during the recommendation computation process.

By utilising data from online and physical user interaction with products within the lab, the framework was able to produce 4 datasets: ratings, purchasing history, browsing history and staying time. Each dataset was able to produce recommendations. Figure 22 shows recommendations for a user on WooCommerce platform. The left part of the figure depicts computed recommendations using the browsing history dataset, while the right part of the figure depicts recommendations for the same user using the staying time dataset.

Regarding custom user feedback, practitioners may specify the feedback data according to their needs, but the scoring scale still needs to be a number from 1 to 5. For instance, custom feedback could constitute the number of times a customer has walked passed a product: while the acquired value can be a large number, e.g. 112 times, the dataset needs to be scaled to values 1-5.

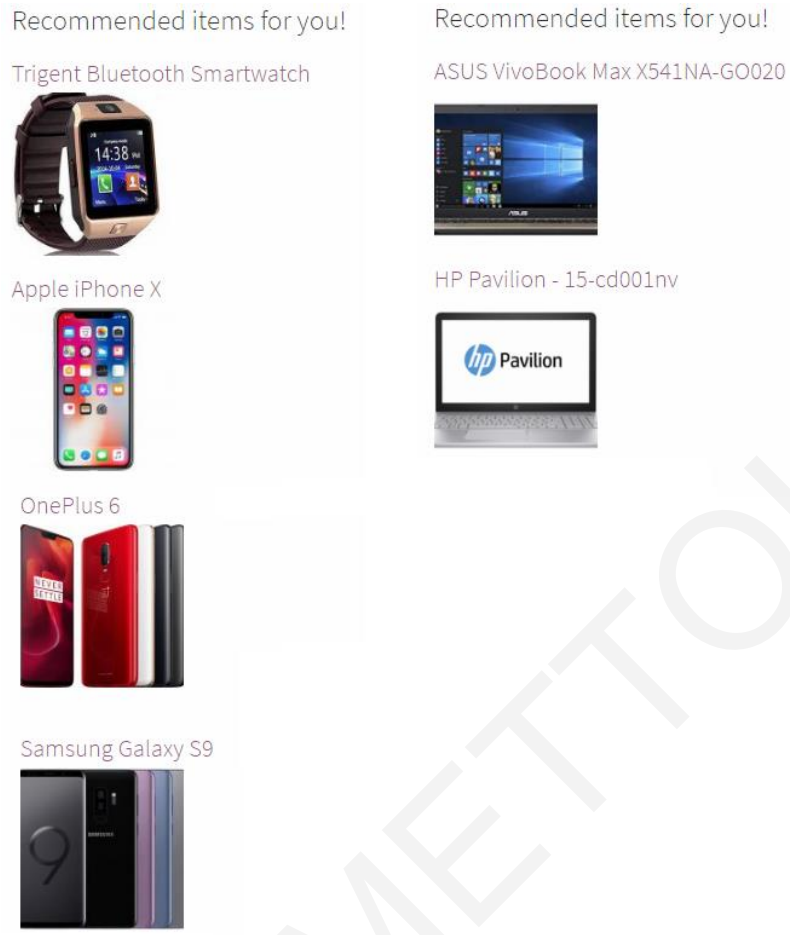

Figure 22: Recommendations for user Chris on WooCommerce platform: on the left are shown recommendations using the browsing history dataset; on the right are shown recommendations using the staying time dataset.

6.2 Evaluation

The evaluation of the framework was conducted via the survey method by using a task-oriented questionnaire. The aim was to use the UbiCARS MDD Framework to develop a full RS on an e-store to the point where recommendations are presented. The evaluation was conducted by involving only one participant at a time, with the author (as the researcher responsible) observing the entire process with each participant. Estimated duration for an evaluation session was 60 minutes.

As discussed in Section 1.3, the primary target group of the framework are e-store developers and practitioners with no expertise in RSs. Secondary target groups are recommender system experts and model-driven development practitioners/experts. The aim of involving the secondary target groups is to acquire expert evaluations in terms of

the Recommender Systems domain and the MDD domain. Professional developers, as the primary target group of the evaluation, were invited by the author to participate. Businesses active in e-commerce, as well as Information Technology (IT) companies that develop e-stores for e-commerce businesses have been contacted.

Through the questionnaire, participants were given a set of tasks to complete using the framework and, at the same time, they were asked to report their findings and respond to various questions about their experience with it. For the purposes of this evaluation, participants were asked to assume working for a retail business selling electronic products that has an e-store and a physical store.

6.2.1 Evaluation Methodology

Before conducting evaluation sessions with participants, we have conducted a pilot study with an expert researcher in the area of Human Computer Interaction (HCI). During the pilot, the researcher had conducted a full evaluation session with the framework, during which 13 errors on tasks description have been identified. The errors were both minor, such as to rephrase a sentence, and major, such as to rewrite a task or break a task into subtasks. All errors have been corrected before evaluation sessions begun.

Prior to the evaluation session, participants were given a brief, two page developer's manual on the modelling editor, its toolbox and the modelling elements. They were also provided with a short video on the MDD Framework and three introductory videos to watch: a video on the topic of recommender systems and the purpose of the UbiCARS MDD Framework, a video on the modelling editor and a video on CARSKIT so that to become familiar with its usage and potential. Before starting each evaluation session, we had introduced the UbiCARS MDD framework to the participant, explained the aim and the steps of the evaluation session process, as well as answered any questions the participant had. It had been made clear to participants that they could stop the process at any moment. They were also instructed that they could ask the author anything related to the process, the instructions or the system, such as a clarification, or report a problem; however, the author as observer would, by no means, interrupt the process himself. In case participants asked for a clarification or reported a problem, the author would answer the question and, if possible, fix the issue, so that the evaluation could proceed; at the same time, the author would record the issue in a report. After the end of an evaluation session, any problems or ambiguities with the instructions or the system UI would be addressed by the author before proceeding with further evaluation sessions.

The survey was split into three parts: the first one constituted the modelling part, the second one was the configuration and validation part, while the third one was named "final questionnaire". The modelling part and the configuration/validation part included all the tasks that participants were asked to complete. The final questionnaire (third part of the evaluation survey) did not include any tasks, only questions to respond to.

Modelling

During the modelling part, participants were requested to use the modelling editor and its toolbox to add and edit elements on the canvas and thus create a ubiquitous context-aware recommender system application model. The process consisted of 8 tasks (tasks 1-8), for each of which the questionnaire included a description of its purpose, the steps that participants should follow in detail and two 7-Likert scale agreement questions: whether participants agree on being able to understand the task, and whether they agree the task being difficult for them to accomplish (1 being strongly disagree and 7 being strongly agree). By successfully completing these 8 tasks, a full UbiCARS model was constructed that could successfully be deployed on the system UI. Each task description in the questionnaire was followed by the question *"Please record your time: I started at..."*, so that participants would record the time just before starting working on the task. The time was recorded in the format *hh:mm*, e.g. 10:42. After task completion and before proceeding to answering questions, a *"Please record your time: I stopped at..."* question instructed participants to also record the time right after finishing with the task, using the same time format. As the aim was to record the exact time duration participants spent on the actual task in order to complete it, the time-recording questions were placed only after the task description text so that the time needed to read the task was excluded. In this manner, the *task duration* per task per participant was recorded within the questionnaire.

One additional modelling task, namely task 14, was also included in the questionnaire. Task 14 was executed only after the next part of the evaluation survey, the configuration and validation part was completed. Task 14 required from participants, after configuring the e-store, to go back to the modelling editor to alter the model by adding context, so as to also include customer's location while staying in front of products in the physical store, to enable context-aware computation of recommendations.

Configuration and Validation of Results

The configuration and validation part was consisted of five tasks during which participants were instructed on how to upload their model on the system UI, conduct e-store configuration, four dataset production (ratings, purchasing history, browsing history and staying time, see Section 6.1), recommendation computation with any of those datasets and finally recommendation presentation through the e-store. The aim of these tasks was to show analytically the steps from the time the model is uploaded on the system, up to the point where recommendations were available. As these tasks were short in duration and simple to accomplish, needing no considerable effort from participants (the tasks mainly included clicking buttons on the system UI and observing results), participants were not asked to keep track of their time, neither to state whether the task was difficult for them to accomplish. Instead, participants were asked to record whether they were able to understand the produced results.

Final Questionnaire

The final questionnaire did not include any tasks for participants to work on. Instead, it included demographic questions, questions related to user experience with the modelling editor and the system UI, as well as questions related to the usefulness of the framework and its ease of use. In specific, similarly to [145], we have used questions from the Technology Acceptance Model - TAM satisfaction questionnaire [146, 147], as well as questions from the User Experience Questionnaire - UEQ [148, 149], and questions specifically designed for the evaluation of the UbiCARS framework. For the questionnaire, 7-point Likert scale agreement questions were used. In our analysis, we use the following notation for Likert scale agreement questions:

1: Strongly Disagree
2: Disagree
3: Slightly Disagree
4: Neutral
5: Slightly Agree
6: Agree
7: Strongly Agree

TAM Satisfaction Questionnaire

Questions related to Technology Acceptance Model aim to observe how acceptable the technology is to users. In his initial paper [146], Davis defined new scales for *perceived usefulness* and *perceived ease of use*, which are hypothesised to be fundamental determinants of user acceptance. Perceived usefulness is defined as the tendency of users to use or not use an application to the extent they believe it will help them perform their job better [146]. Perceived ease of use refers to whether the application is easy or hard to use. It is essential that, not only the system is useful in terms of enabling users in doing their job better, but, at the same time, to be easy to use, so that the benefits of usefulness are not out-weighted by the too much effort users need to put in using the application [146].

Hans van der Heijden [147] states that the TAM model has been refined and extended many times and that, it has been confirmed by research works that perceived usefulness is the strongest predictor of user acceptance across a diverse area of research settings, at the expense of perceived ease of use and perceived enjoyment. His study [147] supports the hypotheses that, for a hedonic information system, perceived enjoyment and perceived ease of use are stronger determinants of intention to use than perceived usefulness. Therefore, which is more important, i.e. to provide hedonic or utilitarian value, can be determined by system purpose.

In [145], the evaluation method and results are described of an empirical evaluation of model-driven engineering and its infusion in industry in the context of industrial participants in an EU research project (MODELPLEX). The author uses an extended version of TAM for collecting developers' perceptions regarding MDE and tools, as well as intensions for future usage.

For our evaluation purposes, we have adopted Davis' TAM items for perceived usefulness and ease of use. Table 5 depicts how the TAM items from literature have been adapted for the UbiCARS Framework evaluation survey.

Table 5: TAM items from literature adapted for the UbiCARS Framework evaluation.

TAM Items	**TAM adapted for UbiCARS Framework evaluation survey (7-point Likert scale agreement questions)**
Perceived Usefulness	
Work More Quickly	Using the UbiCARS Framework would enable me to develop Recommender Systems *more quickly*
Job Performance	Using the UbiCARS Framework *would improve my performance* in developing Recommender Systems
Increase Productivity	Using the UbiCARS Framework *would increase my productivity* in developing Recommender Systems
Effectiveness	Using the UbiCARS Framework would *enhance my effectiveness* in developing Recommender Systems
Makes Job Easier	Using the UbiCARS Framework would *make it easier* to develop Recommender Systems
Useful	I would find the UbiCARS Framework *useful* in developing Recommender Systems
Perceived Ease of Use	
Easy to Learn	Learning to operate the UbiCARS Framework would be *easy* for me
Controllable	I would find it *easy* to get the UbiCARS Framework to *do what I want it to do*
Clear & Understandable	My interaction with the UbiCARS Framework would be *clear and understandable*
Flexible	I would find the UbiCARS Framework to be *flexible* to interact with
Easy to Become Skilful	It would be easy for me to become *skilful* at using the UbiCARS Framework
Easy	I would find the UbiCARS Framework *easy to use*

UEQ Questionnaire

Initially including 229 items and later reduced to 80 and finally to 26, the User Experience Questionnaire was developed by usability experts with the aim to evaluate user experience and usability with products (i.e. software products) [148]. The UEQ questionnaire allows

users to quickly assess their interaction with the product, as well as their experience with it. UEQ includes usability aspects, as well as aspects related to effectiveness and efficiency that are also in accordance with respective ISO standards [149]. Hedonic quality and user satisfaction with the product are also considered [149]. To take its final form, the UEQ has been used and tested in several studies that focused on the quality of interactive products, such as software packages, online-collaboration software and business software.

A number of the 26 items of UEQ are already included in the TAM questionnaire. In our evaluation, we have selected a small number of items from the UEQ questionnaire that are complementary to the TAM questionnaire to avoid repetition, and also to keep the survey as short as possible. UEQ items have the form of a semantic differential [148], meaning that each item is represented by two terms with opposite meanings. In terms of the UEQ, 4 items were included in the survey questionnaire.

Other Questions

The questionnaire included one question related to the Research Questions defined in Section 1.2, two questions investigating participants' future use intention for the UbiCARS Framework, as well as three questions on participants' mental effort and feelings towards the Framework and the MDD Methodology.

Comparison Questions Between the UbiCARS Framework and CARSKIT

As other model-driven development approaches for the development of recommender systems do not yet exist, a direct comparison of the UbiCARS Framework against such tools was not possible. However, by using other recommendation engines and frameworks as the baseline, it is possible to compare effort and time needed to develop recommender systems against the model-driven approach of the UbiCARS Framework. Due to the extended time and effort needed to manually (using coding) develop RSs using existent recommendation engines/frameworks in the context of an evaluation session, as well as due to that the possibility participants in the study have any familiarity with any of these engines/frameworks is reduced (meaning that even more development time will be needed), we had decided to simulate the comparison process, rather than having participants actually conducting it. The simulation involved making the participants familiar with the baseline recommendation engine so that they could estimate the time and effort needed for them to develop a recommender system using it, instead of doing the actually work.

We have selected to use CARSKIT recommendation engine as the baseline recommendation engine for the simulation, as it is easy to use, and it was already set up within our framework. The simulation involved a demonstration of how CARSKIT works as a stand-alone engine, i.e. out of the context of our framework, including observing its configuration files, datasets and results. A demonstration video for CARSKIT was also made for interested participants to watch prior to the evaluation session, as well as during the session at their will.

We acknowledge that, ideally, a direct comparison between the UbiCARS Framework and the CARSKIT recommendation engine should be conducted, either utilising one group of users conducting sessions with both of them and compare results, or by having two groups, one experimental group where participants use the UbiCARS Framework to develop recommender systems, and one control group where participants use the CARSKIT recommender engine to develop recommender systems, and then compare the results of the two groups. In such settings, the comparison would involve measuring effort in terms of lines of code and database queries, as well as the time taken to develop a full recommender system with each one of the two tools. We leave such an evaluation process as future work.

Two comparison questions between the UbiCARS Framework and the CARSKIT recommendation engine were included in the questionnaire.

6.2.2 Summary of Evaluation Results

In this section, a summary of the evaluation results is provided. For a detailed presentation and discussion of the results, please see Section 6.2.3.

The overall results of the Evaluation of the UbiCARS MDD Framework with participants were very positive:

- A total of 20 participants conducted the evaluation session for a total of 19.5 hours. While 15 of them were developers by profession (developers group), from the remaining five participants (non-developers group), four participants had at least four years of accumulated programming experience.
- 80% of participants had little to none experience in RSs and algorithms.
- 70% of participants had experience in e-store development/administration.

- All participants were able to successfully complete all tasks. The modelling editor and system UI were available for use at all times as none of the evaluation sessions was interrupted or suspended due to technical problems with the editor or system UI.
- Participants could complete the modelling tasks of the evaluation on average in less than half an hour (29.6 minutes), and the entire process to develop a full RS on an e-store in less than an hour (58.5 minutes). On the contrary, based on participants' understanding on how the Java Recommendation Engine CARSKIT works, they stated that they would be able to develop a full RS for an e-store by directly using the Recommendation Engine (CARSKIT), WITHOUT using the UbiCARS Modelling Framework in a few days or weeks. It is observed that there is a big difference between the time needed to develop a RS with the UbiCARS Framework and the estimated time needed to manually develop a RS, even with a recommendation engine such as CARSKIT provided.
- Non-developers seemed to be more familiar with the modelling tasks than developers, as they had achieved less times than developers in many of them (differences were very small though).
- All participants agree to have understood all modelling tasks. On a 7-point Likert scale (1 strongly disagree to 7 strongly agree), the average task comprehension is 6.82. Higher levels of task comprehension by non-developers were observed, in comparison to developers (again, differences were very small).
- Participants disagree that the modelling tasks were difficult for them to accomplish. On a 7-point Likert scale (1 strongly disagree to 7 strongly agree), the average disagreement on task difficulty is 1.22. It is observed that, for non-developers, there was less difficulty in the modelling tasks than developers, although the results for developers were also very positive.
- Participants strongly agree to have understood the produced results by the system UI during the configuration and validation part of the evaluation, that included e-store configuration, dataset production, recommendation computation and recommendation presentation through the e-store. On a 7-point Likert scale (1 strongly disagree to 7 strongly agree), the average results comprehension is 6.7.

- On participants' perceived **usefulness**, evaluation results on average indicate that participants (7-point Likert scale: 1 strongly disagree to 7 strongly agree):

  Strongly agree that (mean values are 6.5 or above):
  - Using the UbiCARS Framework would enable them to develop recommender systems more quickly (mean = 6.75).
  - Using the UbiCARS Framework would improve their performance in developing recommender systems (mean = 6.6).
  - Using the UbiCARS Framework would increase their productivity in developing recommender systems (mean = 6.75).
  - Using the UbiCARS Framework would make it easier to develop recommender systems (mean = 6.7).
  - They would find the UbiCARS Framework useful in developing recommender systems (mean = 6.8).

  Agree that (mean values between 5.5 and 6.49):
  - Using the UbiCARS Framework would enhance their effectiveness in developing recommender systems (mean = 6.45).

- On participants' perceived **ease of use**, evaluation results on average indicate that participants (7-point Likert scale: 1 strongly disagree to 7 strongly agree):

  Strongly agree that (mean values are 6.5 or above):
  - They would find the UbiCARS Framework easy to use (mean = 6.5).

  Agree that (mean values between 5.5 and 6.49):
  - Learning to operate the UbiCARS Framework would be easy for them (mean = 6.3).
  - They would find it easy to get the UbiCARS Framework to do what they want it to do (mean = 5.95).
  - Their interaction with the UbiCARS Framework would be clear and understandable (mean = 6.3).
  - They would find the UbiCARS Framework to be flexible to interact with (mean = 6.05).
  - It would be easy for them to become skilful at using the UbiCARS Framework (mean = 6.2).

- It is observed that, although results for both the perceived usefulness and perceived ease-of-use of the framework are very positive, participants seem to value higher the usefulness of the framework. Across all questions, mean value for perceived usefulness is 6.68, while mean value for perceived ease-of-use is 6.22.
- Participants *strongly agree* in that using the UbiCARS Framework for developing recommender systems would reduce their effort in terms of lines of code and database queries needed.
- Participants *disagree* that it would be easier for them to develop recommender systems by using the Java Recommendation Engine CARSKIT instead of the UbiCARS Modelling Framework.
- Participants *agree* that they would revisit the UbiCARS Framework within a week's time if it was available for use.
- Participants *agree* that they would revisit the UbiCARS Framework regularly if it was available for use.
- Participants *agree* that it does not require a lot of mental effort to interact with the UbiCARS Framework.
- Participants *strongly agree* that using a model-driven development approach through the UbiCARS Framework to develop recommender systems is a good idea.
- Participants *strongly agree* that they feel positive toward the UbiCARS Framework.
- In terms of the User Experience Questionnaire, on average, participants found the framework to be: Fairly Enjoyable; Fairly Exciting; Fairly Pleasing; and Fairly Interesting.

Framework Revisits

To measure the *learnability* of the UbiCARS Framework's DSML and modelling editor, i.e. how quickly the DSML and editor allow users to become familiar with them and enable them to make good use of their features and capabilities, we had two participants of the evaluation revisiting the framework a second time after the main evaluation, through new evaluation sessions. In the revisiting evaluation sessions, the aim was to re-conduct the modelling part of the evaluation; however, participants this time did not have in their disposal a detailed description of tasks, only an outline of what they should achieve with

the editor. A detailed description of the evaluation of the learnability of the UbiCARS DSML is provided in Appendix A.

- Learnability for the UbiCARS MDD Framework for the first participant is found to be 82 minutes (the total interaction time for the participant with the framework).
- Due to errors in the model produced as outcome by the second participant, learnability for the second participant could not be determined from the evaluation session and is estimated to be more than 64 minutes (the total interaction time for the participant with the framework). It is highly probable though, to be only 30-60 minutes more, which is the time needed for one more session with the modelling editor.
- One participant *agrees* and one *slightly agrees* that revisiting the UbiCARS Framework to develop a recommender system was easy for them.
- Both participants maintained their opinion from the first evaluation session in that they *strongly agree* that using a model-driven development approach via the UbiCARS Editor to develop recommender systems is a good idea.

Remote Evaluation – Full UEQ Questionnaire

Attempts were made to facilitate a remote evaluation of the UbiCARS Framework. In the remote evaluation sessions, participants were requested to use the UbiCARS Modelling Editor to create a Recommender System application model. Feeding the model to the system through the System UI was out of the scope of the remote evaluation. Three participants conducted the remote evaluation session. The remote evaluation is described in Appendix B.

- Results from five participants (three from remote evaluation sessions and two from framework revisit evaluation sessions) that have responded to the full UEQ questionnaire indicate very positive results (see Figure 59), especially for the following items for which mean values were above 2.5 out of 3, ordered highest first: *"My interaction with the UbiCARS Framework was":* Easy; Good; Practical; Easy to learn; Understandable; Secure; Efficient; Friendly; Meets expectations;
- In terms of the UEQ scales (see Figure 60), the results were extremely positive. From the most highly rated UEQ scales to the least, we can report that:
    - It was easy to get familiar with the framework and learn how to use it - Perspicuity.

- Participants were able to solve their tasks without unnecessary effort - Efficiency.
- Participants liked the framework - Attractiveness.
- Participant felt being in control of the interaction - Dependability.
- It was exciting and motivating to use the framework - Stimulation.
- The framework was innovative, creative and interesting - Novelty.

In this section, a summary of the UbiCARS MDD Framework evaluation results was presented. Section 6.2.3 provides a detailed presentation of the survey evaluation results (task-oriented questionnaire). For the framework revisits and the remote evaluation, the reader is referred to Appendices Appendix **A** and Appendix **B**.

6.2.3 Detailed Presentation of Evaluation Results

It is important to note that all participants were able to complete all tasks. Two participants decided to entirely skip task 14 (the last task of the evaluation), before reading the task description. The modelling editor and system UI were available for use at all times. No evaluation session was interrupted or suspended due to technical problems with the editor or system UI. Any problems reported by participants, as well as notes made by the author are reported in sub-section "Evaluation Report and User Comments" of this section.

Demographics

A total of 20 users (n=20) participated in the evaluation session for a total of 19.5 hours. Although our framework evaluation aimed for a higher number of participants, nevertheless, we state that our results are significant, firstly because these 20 participants fall well within our target group, hence they are perceived as experts for the purposes of our evaluation, and secondly, because they have been involved with the framework a significant amount of time, completing tasks and answering questions, and therefore their feedback can be perceived as valuable and important.

For the remaining of the analysis we will use "n" in figures and tables to denote the number of participants in the respective context. The gender distribution of the participants was 15 males and 5 females. Half of the participants belonged in the 35-44 age group, 40% in the 25-34 age group and the remaining 10% in the 18-24 age group, with the most common age being 37 (mode=37) and average age being 32.1 years old.

From the 20 participants, 15 were developers by profession. From the remaining five participants, four stated to have at least four years of programming experience. Thus, from the total number of participants, only one person, an e-commerce store administrator, had no programming experience at all. All 19 remaining participants (95%) had at least four years of accumulated programming experience, 10.52 years on average and 200 years in total. The maximum accumulated programming experience for a participant was 25 years.

In the following analysis, we distinguish between the (professional) developers group with a total of 15 participants (n=15), the non-developers group with a total of five participants (n=5), and the total participants group (n=20). From the non-developers group, two participants were model-driven development practitioners/experts, one participated as an e-commerce store administrator, one as a PhD Student in Computer Science department of the University of Cyprus, and another participant as a researcher.

Figure 23 depicts participants' experience for the total number of participants in (i) developing and/or technically administrating e-stores, (ii) in using model-driven development tools, UML or other modelling tools, (iii) in developing recommender systems, and (iv) their expertise in recommendation algorithms. 5% of participants were experts in developing and/or technically administrating e-stores and 5% in using model-driven development tools, UML or other modelling tools. An additional 30% and 15% of participants respectively had advanced experience in these two areas. In developing recommender systems, 75% of participants had little to none experience, while 80% of participants had little to none experience in recommendation algorithms. On the contrary, 70% of participants had experience in developing and/or technically administrating e-stores. For the developers group, the respective percentage was 73.3.

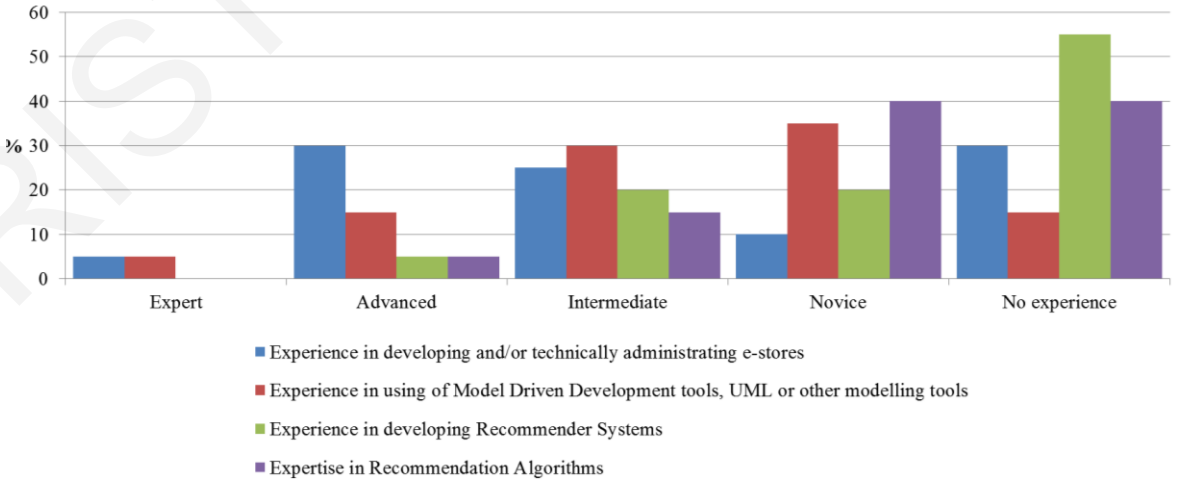

Figure 23: Total participants' experience & expertise.

From the above discussion it is observed that the majority of participants were non-experts in developing RSs and algorithms. Moreover, the majority of participants had knowledge in e-store development and administration.

Modelling Part

As previously mentioned, the modelling part consisted of 8 tasks (tasks 1-8), followed by another task (task 14) which was completed after the configuration/validation part was completed. Each task was comprised of a number of subtasks, which were the actual activities participants had to perform. For the initial tasks where participants had no prior experience with the editor, the subtasks were designed to be simple activities; however, as the evaluation progressed, subtasks' complexity increased, building also upon previous knowledge of participants. For example, while task 1.1 that aimed to create a CARS element on canvas, explicitly informed participants what to do: *"From the toolbox, click on the createCARS element and then click on the empty canvas to create a CARS"*, task 1.2, that aimed to create a UbiCARS element, simply stated *"Create on canvas a UbiCARS"*. Table 6 summarises the aim of each task and the number of their subtasks.

Table 6: Task description for the Modelling Part of the evaluation.

Task #	Number of subtasks	Aim of Task
Task 1	3	To familiarise participants with the modelling editor and its toolbox by creating a few simple model elements for CARS and UbiCARS.
Task 2	6	To add elements in the model that are relevant to the computation of recommendations, i.e. a recommendation engine and an algorithm.
Task 3	5	To add elements related to tracking of customer's browsing history on products in the e-store, i.e. record customer accesses in product webpages.
Task 4	4	To add elements related to customers' "staying time in front of a product" as user feedback on products from the physical store.
Task 5	2	To add elements for enabling the UbiCARS app to utilise customers' staying time in front of a product via Bluetooth beacons.

Task 6	2	To configure the Staying Time element of the UbiCARS app in terms of (i) parameter RSSI min Detection Threshold and (ii) the parameter Max Time Interval BSD.
Task 7	7	To complete the model by adding on canvas more user-product interaction elements and database resources, as well as appropriately link them together.
Task 8	2	To add on canvas a Configuration File element
Task 14	4	To go back to the modelling editor to alter the model so as to also include customer's location while staying in front of products in the physical store, to enable context-aware computation of recommendations.

Tasks Duration

For the total number of participants, mean value for the duration of tasks of the modelling part (tasks 1-8) was 29.6 minutes. Figure 24 depicts the maximum, minimum, median and mean task duration for the total of participants (n=20). Task 14 was only completed by 18 participants. On average ("Mean" in the figure), the most time consuming task was task 14 at 6.6 minutes. This may be due to that the notion of context, especially in the recommender systems domain, is inherently not easy to comprehend for participants, thus, they probably needed a few more minutes to understand the task objective and what to do. The least time consuming task was task 5 at 1.3 minutes. The median follows a similar distribution to the mean, indicating that the task duration values among participants are evenly distributed between the lowest and highest values.

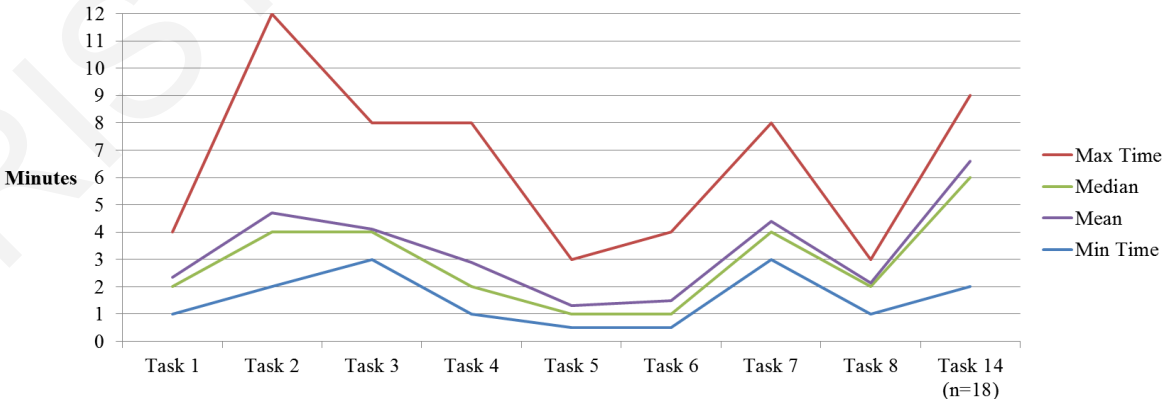

Figure 24: Tasks duration for all participants (n=20).

The "Min Time" graph in Figure 24 shows the minimum task duration across all participants. This is the ideal scenario: it proves that it is possible for users to achieve such

low timings. It is also true that, through repeated usage over time, users' time per task could be reduced even further. The maximum time however shows the worst scenario: the maximum time per task participants have achieved. It was observed that the maximum times per task, as indicated in the graph, were recorded by different participants. The same applies for the minimum times. It is also the case that, participants that have registered the maximum time in one task, may have achieved a time close to the minimum time in other tasks. Regarding the 12-minute maximum peak of task 2 in Figure 24, this was recorded in only one session. The next maximum value for task 2 was six minutes. Note that the two MDD practitioners/experts in the non-developers group did not achieve the minimum times in all tasks that are shown in Figure 24. This may be due to that experts probably took more time to explore the editor and its tools, than novice users did.

We should note at this point that, by having participants noting their starting and ending times for the tasks in the time format *hh:mm* (see Section 6.2.1), and then by calculating the total duration of a task through the time difference between the noted times, the result lies within accuracy of minutes, meaning that seconds are not considered (the accuracy lies between ± 1 minute). Table 7 presents examples of the ideal scenario, the worst scenario and a random one.

Table 7: Timing error margin scenarios for the modelling tasks.

	Real Starting Time	**Noted Starting Time**	**Real Ending Time**	**Noted Ending Time**	**Task Duration in minutes**	**Error in seconds**
Ideal Scenario	10:42.12	10:42	10:45.12	10:45	3	0
Random scenario	10:42.12	10:42	10:45.45	10:45	3	33
Worst Scenario	10:42.00	10:42	10:45.59	10:45	3	59

On this issue, we state that, for the purposes of this survey where new Model-Driven tools and DSML are being used in tasks to design models for applications, such tasks are expected to be somewhat time consuming and thus the expected error margin is not significant. Taking the mean error of 30 seconds per task, the propagated error for all 9 tasks would be 4.5 minutes. In addition, when the starting and ending times noted for a task fall within the same minute in time, then, the time difference is calculated to be zero.

This was observed in the minimum task duration dataset ("Min Time" graph in Figure 24) for tasks 5 and 6. To avoid a zero time record for the task duration which would not be accurate, for those tasks, we have used the average error margin, i.e. noted that the minimum duration for these tasks is 0.5 minutes.

Figure 25 shows the distribution with standard deviation (SD or "σ") for the mean time per task for all participants. The mean deviation is 1.4 minutes. More deviation in minutes are observed in tasks 2, 14 and 4 that require adding elements related to recommendation computation, context sensing and ubiquitous user feedback acquisition. We regard these tasks as the most conceptually challenging tasks. It is observed that task 2 that is related with adding recommendation engine and algorithm related elements in the model, has the highest σ, while also from Figure 24 it is shown that task 2 is on average the second most time consuming task at 4.7 minutes. These results may indicate the absence of RSs expertise from participants. Regarding task 7, although it is on average the third most time consuming task at 4.4 minutes (see Figure 24), it is not considered among the most challenging tasks, as it required from participants to add elements that are similar to other elements they had previously added in task 3. The extended duration of task 7 is mainly attributed to its many subtasks, rather than to its level of difficulty.

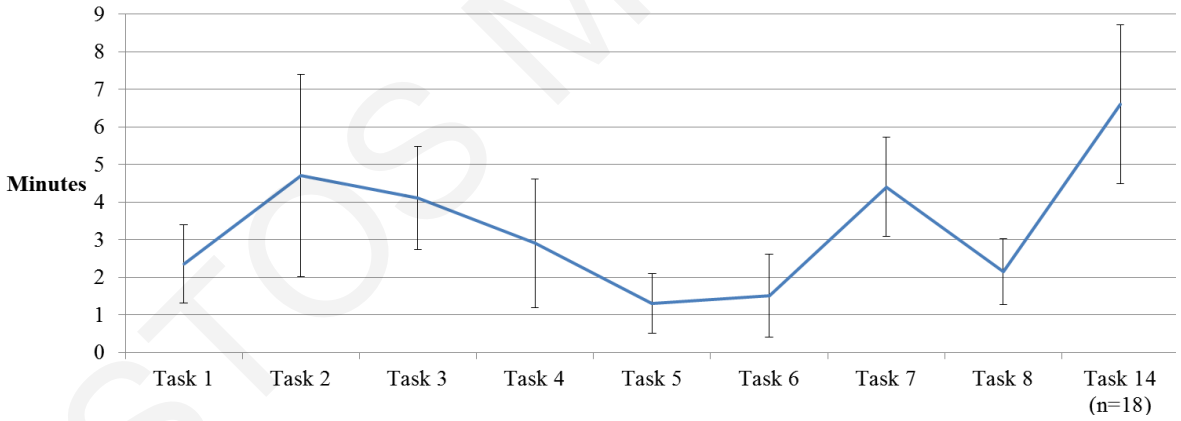

Figure 25: Mean time per task with Standard Deviation – σ per task.

Figure 26 depicts the mean time per task for each of the two groups, developers and non-developers, as well as for all participants. It is shown that the average times are identical for tasks 4, 5, 6 and 14, while they are noticeable different for tasks 2 and 3, with non-developers recording better times. Small differences exist for tasks 1, 7 and 8, with developers achieving better times only for task 8. These results may suggest that non-developers are more familiar with modelling tasks than developers. Indeed, two of the five participants in the non-developers group (40%) are MDD practitioners/experts.

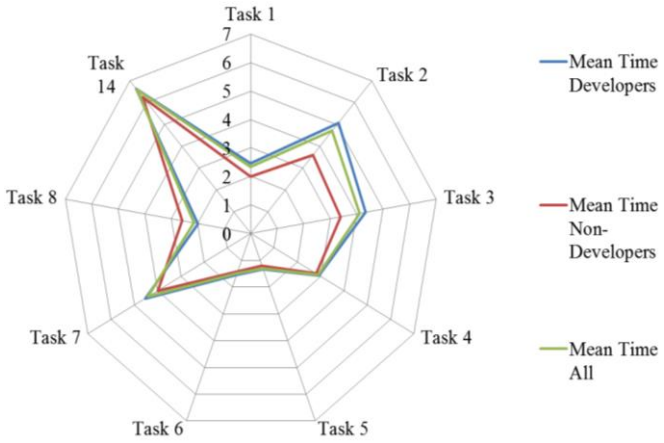

Figure 26: Average duration per task in minutes: developers, non-developers and all.

Figure 27 shows three time distributions: the distribution of the total duration of all tasks of the evaluation survey (i.e., both modelling and configuration tasks), the distribution of the total duration of tasks of the modelling part (tasks 1-8), and the distribution of the duration of the most time-consuming task, i.e. task 14. The presented distributions correspond to all participants. Two participants have not completed task 14; interestingly however, even with skipping a task, these two participants have not scored the best times for any of the depicted distributions.

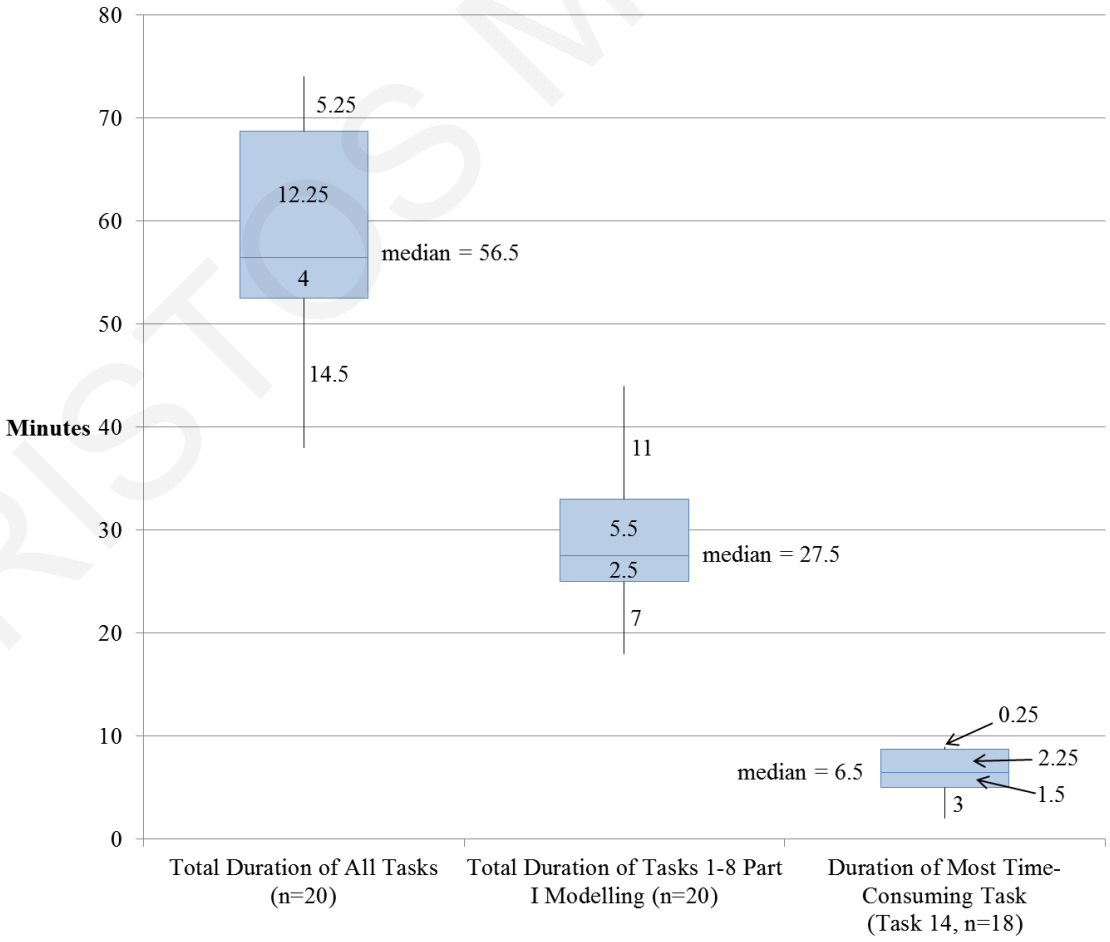

Figure 27: Time distributions on tasks for all participants.

The numbers in the figure show the quartile ranges in minutes. The first quartile is the lower one in each box plot, while the fourth quartile is the upper one. The medians (blue horizontal lines) are explicitly mentioned.

With regard to the distribution of the total duration of all tasks (left box plot in Figure 27), the median (dataset's middle value) is 56.5 minutes, i.e. less than one hour. The minimum value is 38 minutes, while the maximum is 74 minutes. The box, that represents the middle 50% of the values, is not symmetrical: the second quartile shows significantly less variation than the third quartile. The largest variation is observed in the first quartile that is expanding for 14.5 minutes. Therefore, although the minimum time noted is 38 minutes, not many of the participants in this quartile have achieved similar low timings. The forth (upper) quartile has the second smaller variation, ranging from the maximum value of 74 to its lowest one 68.74.

Observing the distribution of the box plot representing the total duration of tasks 1-8 of the modelling part (middle box plot in Figure 27), we notice less variation in the three lower quartiles than the corresponding variation in the quartiles of the box plot to the left. It is also depicted that 50% of the participants have achieved timings close to the median: from 25 to 33 minutes. The minimum time achieved is 18 minutes which is quite low, on average 2.25 minutes per task, while the maximum time achieved is 44 minutes (on average 5.5 minutes per task). Moreover, as the forth quartile shows large variation, it is estimated that not many participants have achieved a timing close to the maximum value.

In overall, we observe that participants took considerably less time to complete the modelling tasks, than the time taken to complete all tasks of the evaluation survey (modelling tasks + configuration/validation tasks. By achieving timings close to 30 minutes or lower, 25% of participants have managed to successfully complete the modelling tasks, while a further 50% achieved even better timings, ranging from 18 minutes to 27.5 minutes. It is also shown that participants achieving the best results in the overall process (lower end of first quartile of left box plot) have comparable timings to participants achieving the worst results in the modelling tasks (upper end of the forth quartile of middle box plot).

Regarding the duration of the Most Time-Consuming Task, which was task 14 (n=18), the variation of the box plot is the lowest among the three box plots. The maximum time taken value is close to the third quartile, such that the forth quartile is not depicted in the graph (it is between 8.75 and 9 minutes). The minimum time taken for this task was 2 minutes, followed by the second minimum time which was 4 minutes. Furthermore, 25%

of the participants have achieved times between 4 and 5 minutes (5 is the upper value of the first quartile).

At this point, we note the following mean values for the total number of participants (not depicted in Figure 27): mean value for the total duration of all tasks was 58.5 minutes, mean value for the duration of tasks of the modelling part (tasks 1-8) was 29.6 minutes.

Participants Perceived Task Comprehension

After completing each task, participants were asked to express their agreement with the statement *"I was able to understand the task"* by using a 7-point Likert scale: 1 (strongly disagree) to 7 (strongly agree). Table 8 shows the mean and standard deviation for each task, while Figure 28 shows in percentages the level of comprehension of each completed task by all participants. Note that, for task 14, one participant did not respond this question, reducing the total responses for the task to 17 (since two participants did not conduct task 14 at all).

Table 8: Mean and Standard Deviation on task comprehension by all participants.

Task #	Mean	Standard Deviation
Task 1	6.9	0.30
Task 2	6.75	0.64
Task 3	6.4	0.75
Task 4	6.9	0.31
Task 5	6.95	0.22
Task 6	6.95	0.59
Task 7	6.8	0.41
Task 8	7	0
Task 14	6.71	0.69

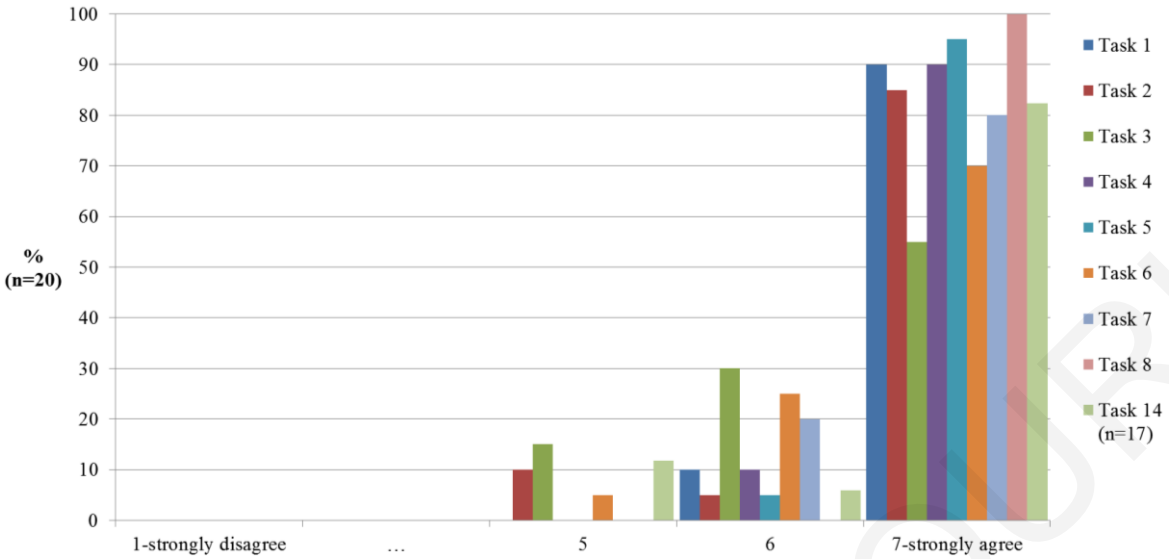

Figure 28: Agreement distribution of all participants to the statement *"I was able to understand the task"*.

From the results it is evident that, in overall, all participants agree to have understood all tasks. Most participants strongly agree that they were able to understand the task at hand. Mean values show that participants tend to strongly agree in having understood all tasks (see Table 8). Only for task 3, participants show a tendency towards agreement (point 6 in scale) in having understood the task. Standard deviation values are low, indicating that participants' responses are concentrated over the mean values. According to Figure 28, tasks 3, 14, 2 and 6 are the least understood tasks. For these tasks, 15%, 11.77%, 10% and 5% of participants respectively have stated that they slightly agree in having understood the tasks (point 5 in scale).

Figure 29 and Figure 30 depict in percentage the level of comprehension of tasks by non-developers and developers respectively. It is observed that, with the exception of tasks 6 and 14, a higher percentage of non-developers have understood the modelling tasks and their outcomes, than developers. As with tasks duration where non-developers have recorded lower mean values than developers for almost half of the tasks, higher levels of tasks comprehension observed here indicate that non-developers are more familiar and able to better understand the modelling tasks than developers. As a reference, we note that 40% of non-developers are experts or have advanced experience in MDD, whereas for developers, none is expert and 13% have advanced experience in MDD. It is the case however that, even if no real expertise in MDD exists in the developers group, developers have indeed understood all tasks.

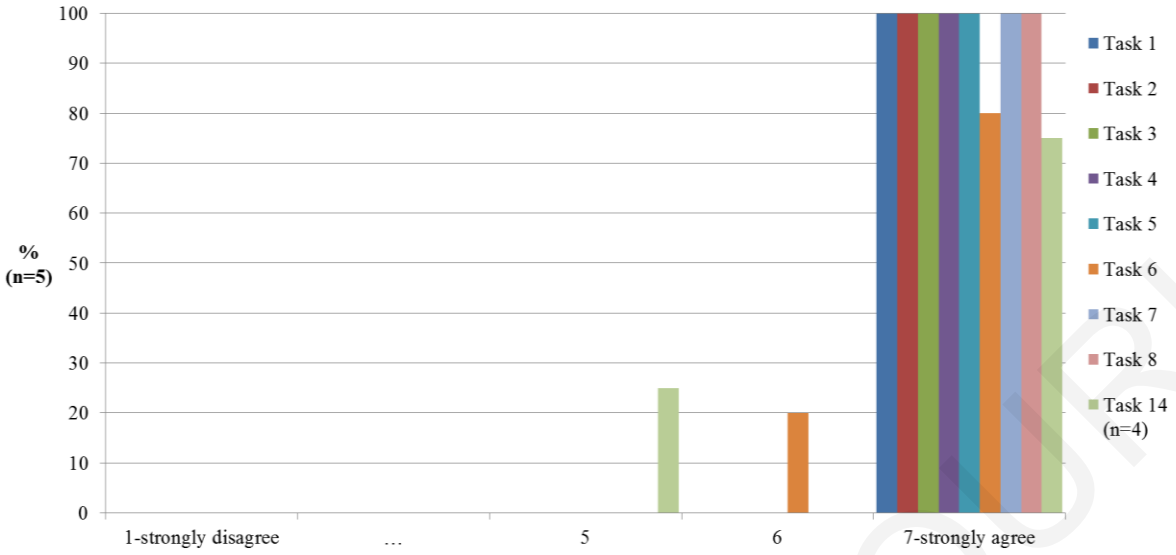

Figure 29: Agreement distribution of non-developers to the statement *"I was able to understand the task"*.

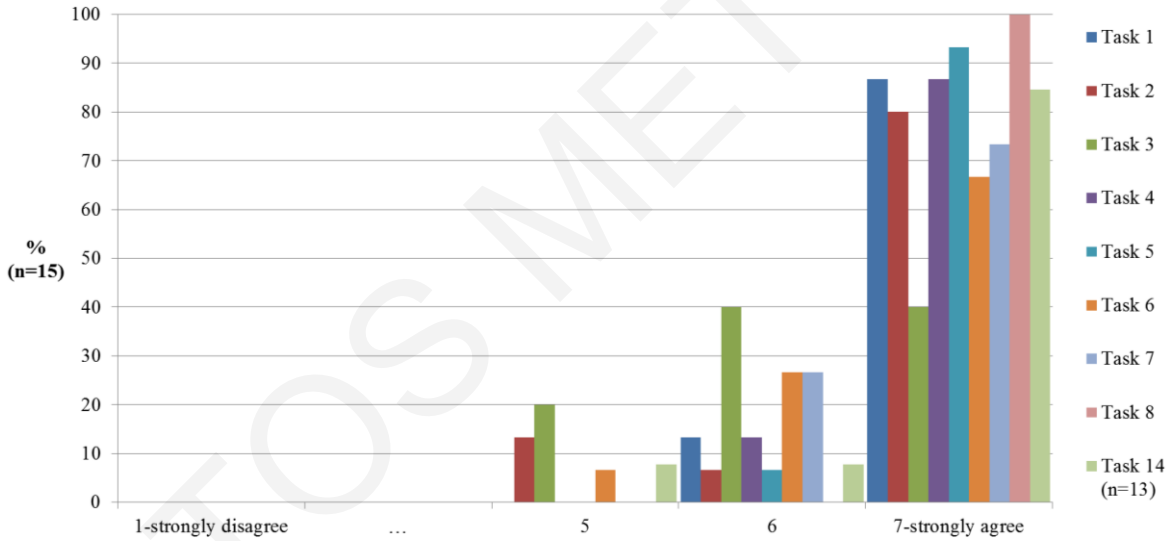

Figure 30: Agreement distribution of developers to the statement *"I was able to understand the task"*.

Participants' Perceived Task Difficulty

After completing each task, participants were also asked to express their agreement with the statement *"This task was difficult for me to accomplish"*, again by using a 7-point Likert scale: 1 (strongly disagree) to 7 (strongly agree). Table 9 shows the mean and standard deviation of participants' responses for each task. Figure 31 shows the percentage of participants' perceived difficulty for each completed task.

Table 9: Mean and Standard Deviation on participants' perceived difficulty per task.

Task #	Mean	Standard Deviation
Task 1	1.25	1.12
Task 2	1.35	1.14
Task 3	1.45	0.69
Task 4	1.15	0.37
Task 5	1	0
Task 6	1.15	0.37
Task 7	1.1	0.31
Task 8	1	0
Task 14	1.56	1.46

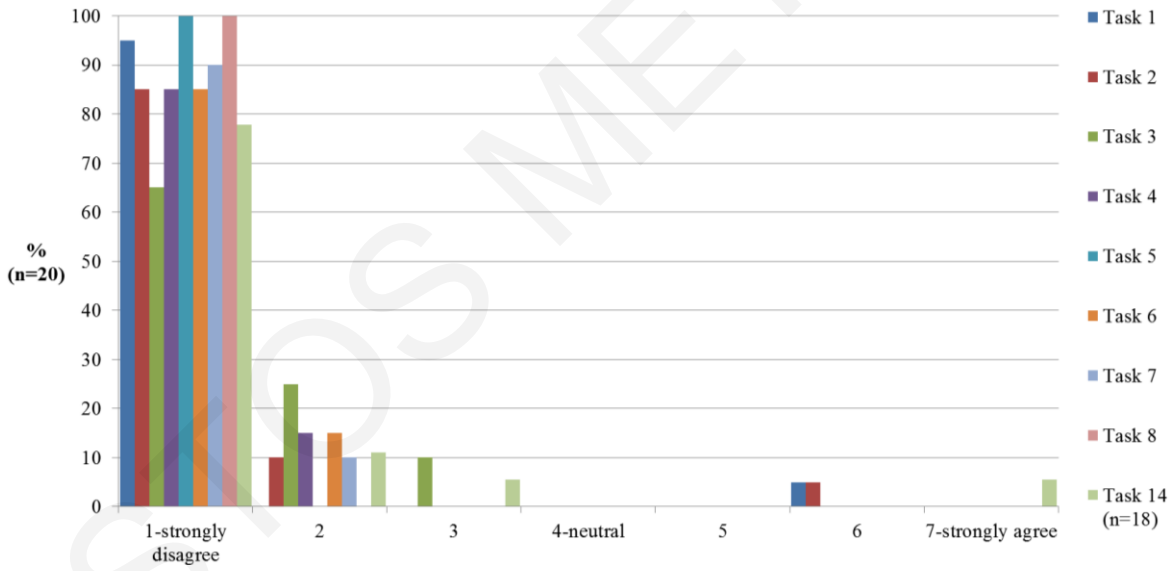

Figure 31: Agreement distribution of all participants to the statement: *"This task was difficult for me to accomplish"*.

In overall, the results are very positive. On average, participants strongly disagree that the tasks were difficult for them to accomplish. For tasks 5 and 8, all participants strongly disagree in being difficult. From Table 9 we observe that all other tasks are balanced between strongly disagree and disagree in being difficult. All of them but one have a tendency towards strong disagreement. Task 14 is the only task that, on average, shows a small tendency towards disagreement.

Standard deviation in Table 9 for tasks 1, 2 and 14 is not as low as the corresponding values in Table 8, a result that could indicate that participants' responses in these tasks are

more spread out over a wider range of values. However, by observing Figure 31 we can see that this is not the case. Tasks 1 and 2 have 95% of responses under strong disagreement and disagreement, while at the same time, they have exactly one response each (5% of the total sample) under agreement. Similarly, task 14 has about 95% of responses on the positive side (strong disagreement, disagreement and slight disagreement), with 77.78% lying on strong disagreement, but with one response under strong agreement. Furthermore, the agreements for tasks 1 and 2 were both provided by the same participant, who, for the remaining tasks, has responded with disagreement for tasks 3 and 4, and strong disagreement for all others, including task 14. It is therefore the case that, either the participant found the modelling process difficult at first, but then quickly became familiar with it and found it not difficult for the remaining tasks, or, the participant made an error in the first two tasks. Similarly, the strong agreement response for task 14 (far right column in Figure 31) was made by another participant who has stated strong disagreement in all other tasks. The possibility therefore that these participants have made an error holds true. Note that the responses to this question express positive opinion in the opposite manner than the responses to the previous question ("I was able to understand the task"), since a full positive response requires strong disagreement from the participant, rather than strong agreement. Indeed, during evaluation sessions a few participants had made such errors, which had later noticed by themselves, but only after completing a number of tasks. In such cases, participants were advised to correct their responses before finishing the survey to reflect their true opinion on the questions. The survey questionnaire allowed participants to revisit previous responses and alter them as they think best.

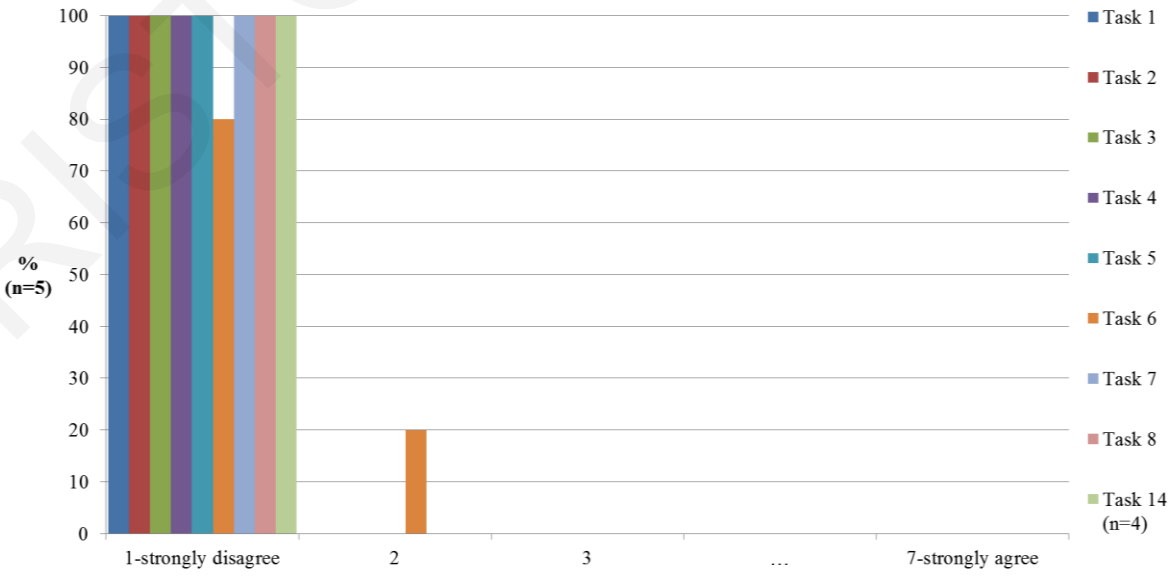

Figure 32: Agreement distribution of non-developers to the statement: *"This task was difficult for me to accomplish"*.

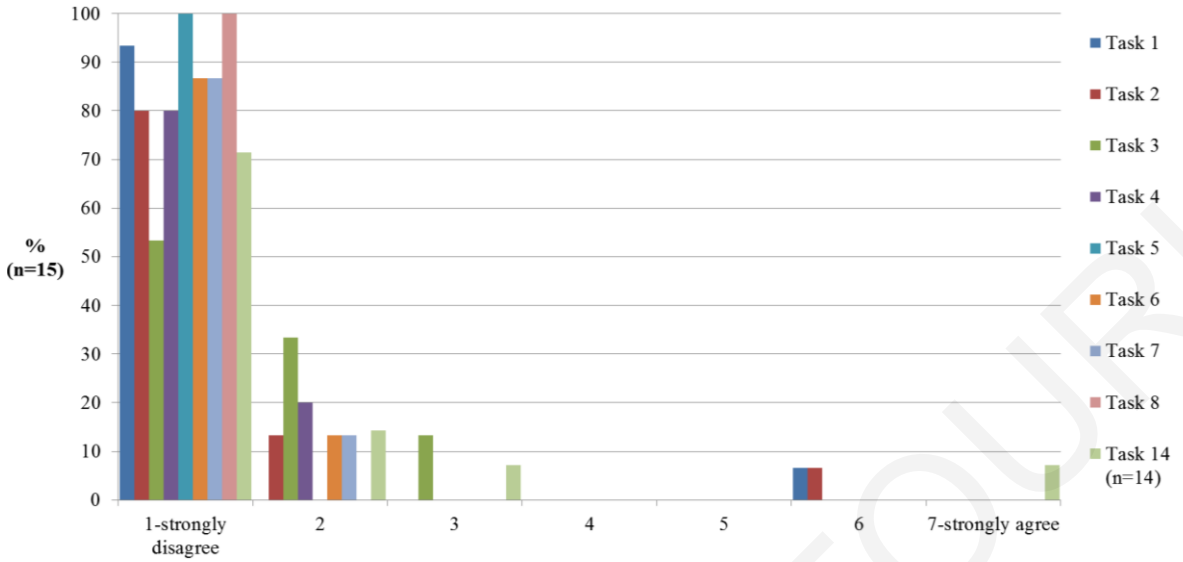

Figure 33: Agreement distribution of developers to the statement: *"This task was difficult for me to accomplish"*.

Figure 32 and Figure 33 depict the percentage of perceived difficulty for each task by non-developers and developers respectively. It is observed that, for non-developers, there was less difficulty in the modelling tasks than developers, although the results for developers are also very positive. It is depicted that Figure 31 and Figure 33 have a similar distribution of values in the graphs, which is explained by the fact that Figure 32 is evenly distributed at the maximum points.

Configuration and Validation Part

During the five tasks of this part of the evaluation, participants were instructed to upload their model on the system UI, conduct e-store configuration, dataset production, recommendation computation and recommendation presentation through the e-store. The tasks were short in duration and simple to accomplish, mainly requesting from participants to use the system UI and observe the presented results. Table 10 summarises the aim of each task (there were not any subtasks).

After completing each task, participants were asked to express their agreement with the statement *"I was able to understand the produced results"*, by using a 7-point Likert scale: 1 (strongly disagree) to 7 (strongly agree). Table 11 shows the mean and standard deviation of participants' responses for each task. Figure 34 shows the percentage of participants' perceived comprehension of results for each completed task.

Table 10: Task Description for the Configuration and Validation Part of the evaluation.

Task #	Aim of Task: Use the System UI to:
Task 9	Parse the model: the necessary configurations on the e-store and its database are automatically created.
Task 10	Fill the new database tables with data.
Task 11	Compile the four datasets (see Section 6.2.1) and then compute recommendations with any of those datasets.
Task 12	Store recommendations.
Task 13	Present Recommendations.
Task 14	Execute the altered model that includes also a context parameter to enable context-aware computation of recommendations.

Table 11: Mean and Standard Deviation for participants' perceived comprehension of results.

Task #	Mean	Standard Deviation
Task 9	6.9	0.31
Task 10	6.55	1.00
Task 11	6.4	0.94
Task 12	6.9	0.31
Task 13	6.95	0.22
Task 14	6.5	0.51

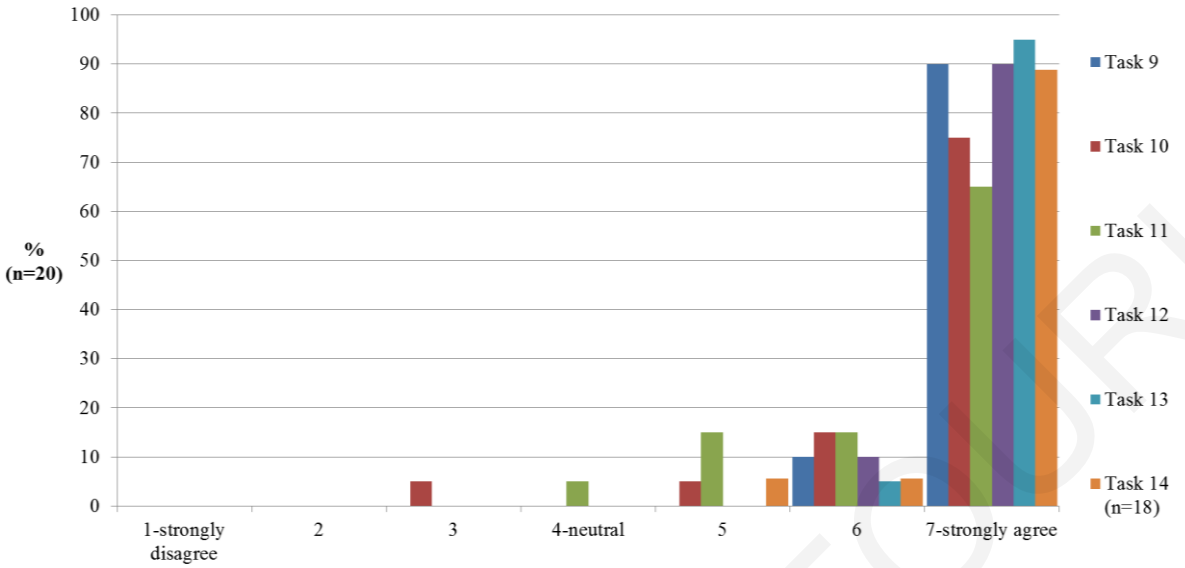

Figure 34: Agreement distribution of participants to the statement: *"I was able to understand the produced results"*.

As task 14 also involved using the system UI for the abovementioned purposes after altering the model in the modelling editor, it included this question as well. Hence, task 14 is part of the analysis below.

On average (Table 11), participants strongly agree in that they were able to understand the produced by the system UI results for tasks 9, 12 and 13. For tasks 10, 11 and 15, on average, participants balance between strongly agreeing and agreeing in understanding the results. Overall, the results from the responses are positive. Standard deviation shows small variation of responses in all but two tasks, 10 and 11. As one may observe from Figure 34, task 10 involves one participant slightly disagreeing in understanding the results, and another one slightly agreeing. As task 10 includes filling the database with data, which is a step that in real scenarios it would not need be realised, it is acceptable for a small number of users to display a degree of uncertainty about the process. Concerning task 11, one participant responded neutrally, while three more responded slightly agreeing. The participant who had responded neutrally, who was also the first participant to conduct the evaluation session, reported not being able to understand on how to proceed due to ambiguity in the instructions and/or problematic user interface design for Task 11 in the system UI. Hence the low scoring. To proceed with the task, we had to solve the ambiguity for the participant; then, the participant was able to fully understand the process and results. The instructions and the interface design for Task 11 have been corrected for evaluations to follow.

Please note that, the corresponding graphs for developers and non-developers do not differ significantly from the corresponding graph for all participants presented in Figure 34. Therefore, they have not been included in the thesis.

Final Questionnaire

The final questionnaire was the last part of the evaluation. It required from participants demographic information, followed by 21 questions about their experience with the UbiCARS MDD framework. In 19 of these questions participants were asked to express their agreement with a statement by using a 7-point Likert scale: 1 (strongly disagree) to 7 (strongly agree). The remaining two questions were consisted of one multiple choice question and one question related to the UEQ questionnaire.

In the following, an analysis is conducted on the participants' responses to each question.

Results on Perceived Usefulness

Question: *Using the UbiCARS Framework would enable me to develop Recommender Systems more quickly.*

Mean and Standard Deviation are: mean = 6.75, σ = 0.64. On average, users tend to strongly agree to the statement. Standard deviation indicates that participants' responses are a bit spread out over a wider range of values. From Figure 35 it is depicted that 85% of participants (17 participants) strongly agree. All participants agree to the statement.

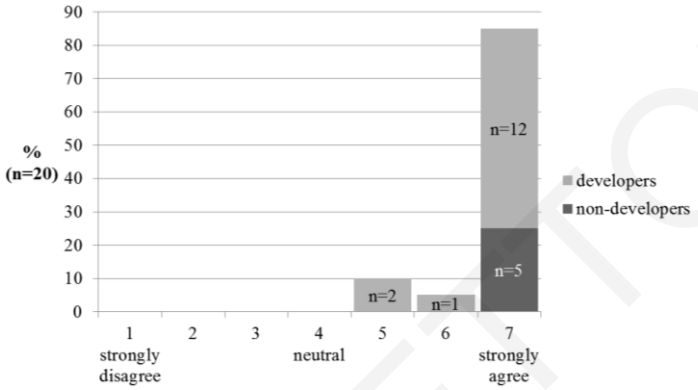

Figure 35: Agreement Distribution of Participants to the Statement: *"Using the UbiCARS Framework would enable me to develop Recommender Systems more quickly"*.

Question: *Using the UbiCARS Framework would improve my performance in developing Recommender Systems.*

Mean and Standard Deviation are: mean = 6.6, σ = 0.68. Based on the mean value, we note that participants tend to strongly agree to the statement. Standard deviation indicates that participants' responses can be spread out over a wider range of values**.** Figure 36 shows that all 20 participants agree. 70% strongly agrees to the statement.

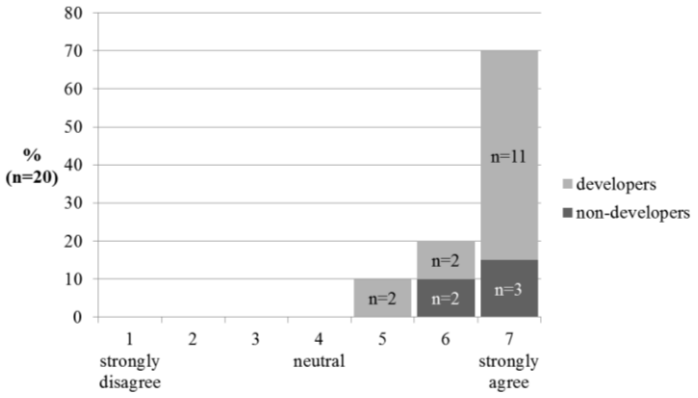

Figure 36: Agreement distribution of participants to the statement: *"Using the UbiCARS Framework would improve my performance in developing Recommender Systems"*.

Question: *Using the UbiCARS Framework would increase my productivity in developing recommender systems.*

Mean and Standard Deviation are: mean = 6.75, σ = 0.55. To this statement participants on average responded positively, leaning towards strong agreement. Standard deviation is relatively low and indicates concentration around values 6 and 7. Figure 37 depicts that 80% of participants strongly agree.

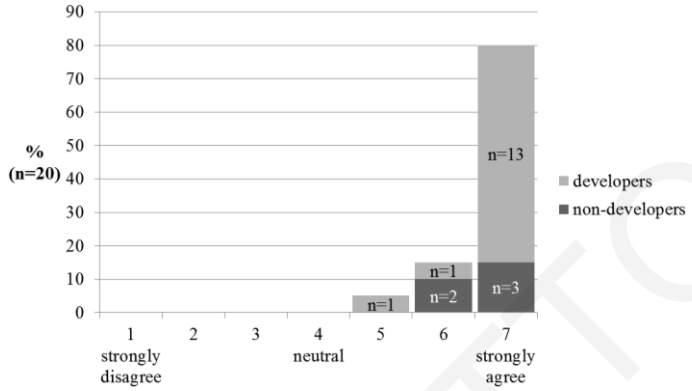

Figure 37: Agreement distribution of participants to the statement: *"Using the UbiCARS Framework would increase my productivity in developing recommender systems"*.

Question: *Using the UbiCARS Framework would enhance my effectiveness in developing recommender systems.*

Mean and Standard Deviation are: mean = 6.45, σ = 0.88. Participants on average balance between agreement and strong agreement. Standard deviation is moderate, indicating a small spread of responses towards lower values as well. Figure 38 shows indeed that one participant (a developer) feels neutral about the statement. The remaining 95% of participants agree to the statement, with 65% in strong agreement (92% of which are developers).

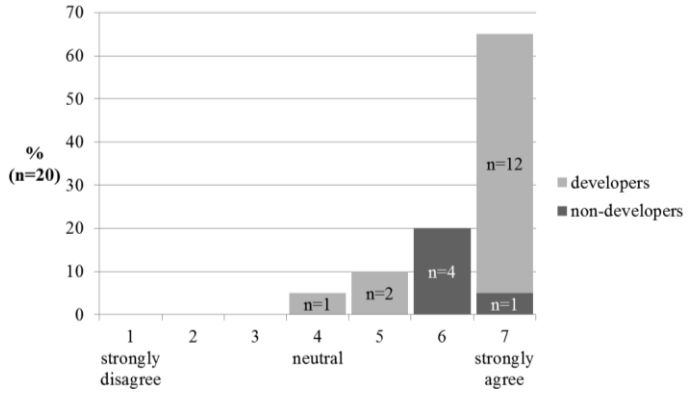

Figure 38: Agreement distribution of participants to the statement: *"Using the UbiCARS Framework would enhance my effectiveness in developing recommender systems"*.

Question: *Using the UbiCARS Framework would make it easier to develop recommender systems.*

Mean and Standard Deviation are: mean = 6.7, σ = 0.57. Participants tend to strongly agree to the statement. Standard deviation is also low, indicating a concentration of values towards strong agreement and agreement (points 7 and 6 on scale). Figure 39 confirms participants' agreement to the statement.

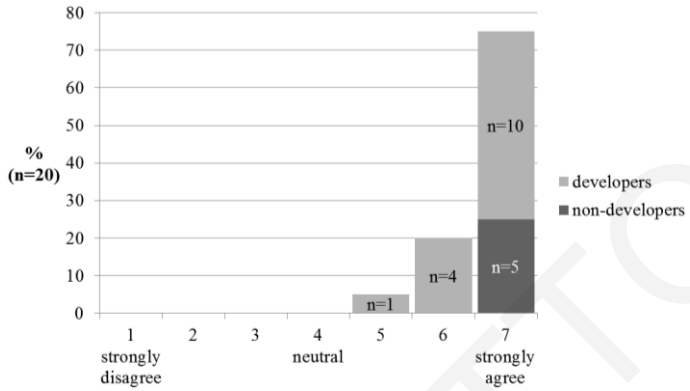

Figure 39: Agreement distribution of participants to the statement: *"Using the UbiCARS Framework would make it easier to develop recommender systems"*.

Question: *I would find the UbiCARS Framework useful in developing recommender systems.*

Mean and Standard Deviation are: mean = 6.8, σ = 0.41. On average, participants tend to strongly agree to the statement. Standard deviation is also not high, something confirmed by Figure 40.

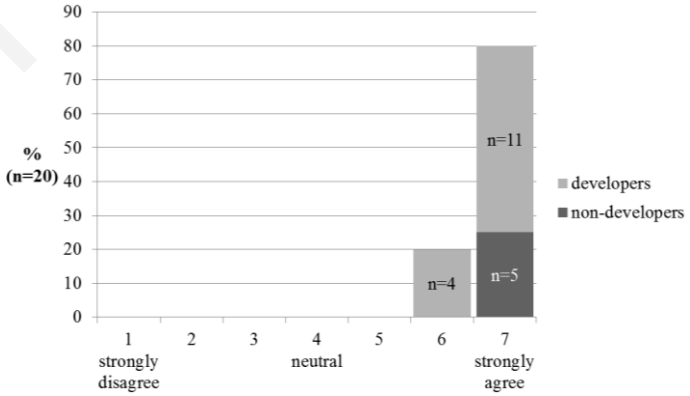

Figure 40: Agreement distribution of participants to the statement *"I would find the UbiCARS Framework useful in developing recommender systems"*.

Results on Perceived Ease of Use

Question: *Learning to operate the UbiCARS Framework would be easy for me.*

Mean and Standard Deviation are: mean = 6.3, σ = 0.92. While the average of participants' responses is positive, standard deviation indicates a wider spread of values for responses. Figure 41 confirms that participants have responded using the entire spectrum of positive values, as well as the neutral one (a non-developer).

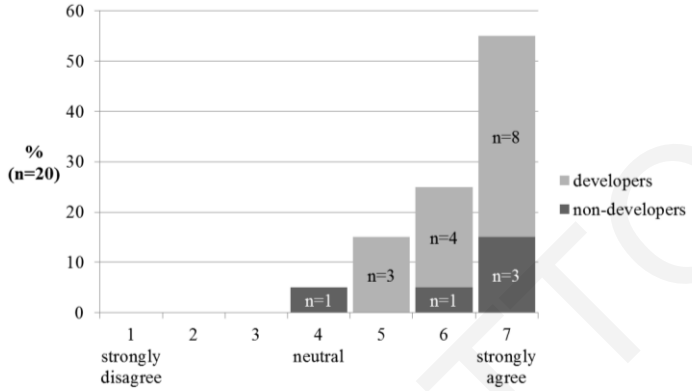

Figure 41: Agreement distribution of participants to the statement *"Learning to operate the UbiCARS Framework would be easy for me".*

Question: *I would find it easy to get the UbiCARS Framework to do what I want it to do.*

Mean and Standard Deviation are: mean = 5.95, σ = 0.94. In overall, participants agree to the statement. Figure 42 depicts that 10% of participants stated neutral response. The participants that strongly support the statement are all developers.

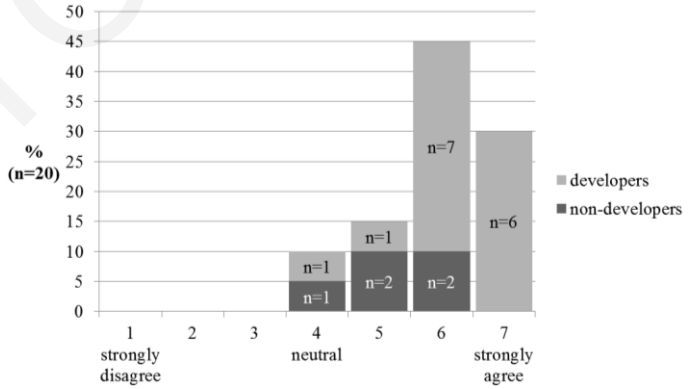

Figure 42: Agreement distribution of participants to the statement *"I would find it easy to get the UbiCARS Framework to do what I want it to do".*

Question: *My interaction with the UbiCARS Framework would be clear and understandable.*

Mean and Standard Deviation are: mean = 6.3, σ = 0.92. Standard deviation shows variation among participants' responses. Figure 43 shows one neutral response.

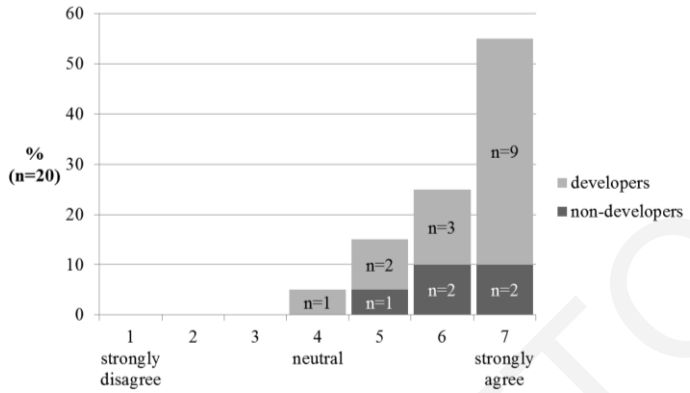

Figure 43: Agreement distribution of participants to the statement *"My interaction with the UbiCARS Framework would be clear and understandable"*.

Question: *I would find the UbiCARS Framework to be flexible to interact with.*

Mean and Standard Deviation are: mean = 6.05, σ = 0.94. On average, participants agree on finding the UbiCARS Framework to be flexible to interact with. Standard deviation indicates variation, which can be observed in Figure 44, where 10% of participants responded neutrally, all developers.

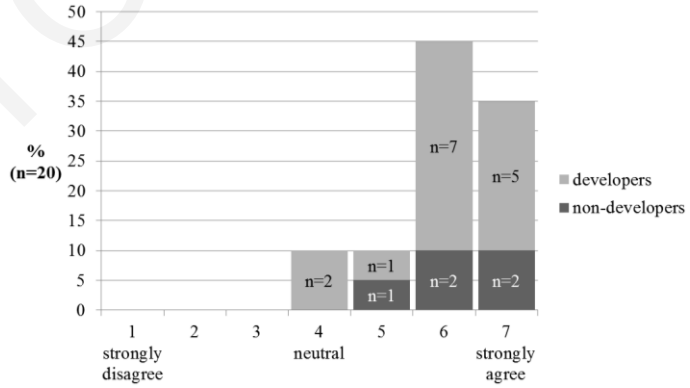

Figure 44: Agreement distribution of participants to the statement *"I would find the UbiCARS Framework to be flexible to interact with"*.

Question: *It would be easy for me to become skilful at using the UbiCARS Framework.*

Mean and Standard Deviation are: mean = 6.2, σ = 1.15. While on average participants' responses indicate agreement, standard deviation indicates some variation. From Figure 45 we observe that, while the majority strongly agrees to the statement, there is one developer response on slight disagreement and another one on neutral.

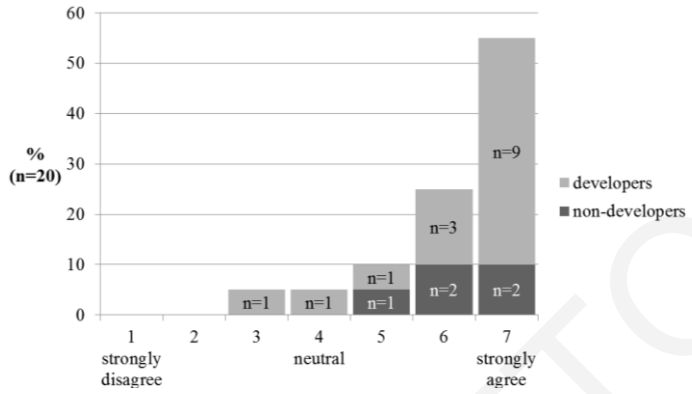

Figure 45: Agreement distribution of participants to the statement *"It would be easy for me to become skilful at using the UbiCARS Framework"*.

Question: *I would find the UbiCARS Framework easy to use.*

Mean and Standard Deviation are: mean = 6.5, σ = 0.76. Participants' responses on this statement were highly positive. From Figure 46 it is depicted that none of the participants expressed neutral or negative opinion.

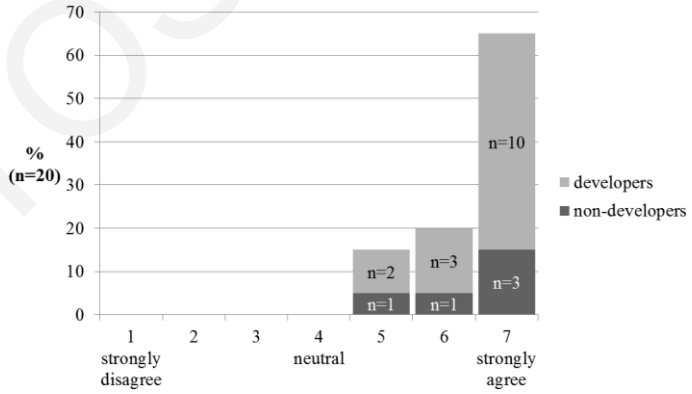

Figure 46: Agreement distribution of participants to the statement *"I would find the UbiCARS Framework easy to use"*.

Results on Participants' Perceived Effort While Interacting with the Framework

Question: *I believe that using the UbiCARS Framework for developing Recommender Systems would reduce my effort in terms of lines of code and database queries needed.*

Mean and Standard are: mean = 6.7, σ = 0.73. Participants on average strongly agree to the statement. Figure 47 depicts all participants lying on the positive values, with the exception of one developer that has stated a neutral response.

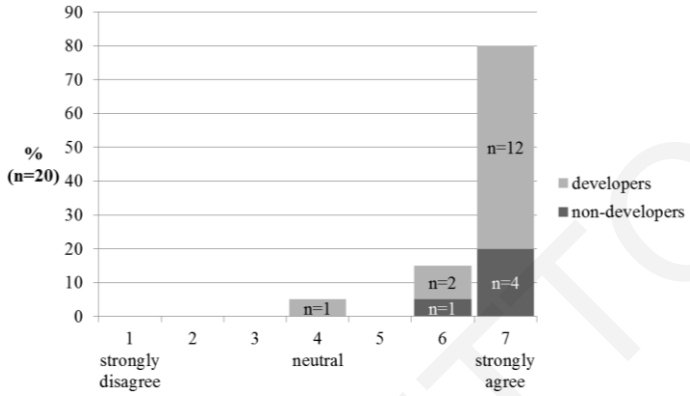

Figure 47: Agreement distribution of participants to the statement *"I believe that using the UbiCARS Framework for developing Recommender Systems would reduce my effort in terms of lines of code and database queries needed"*.

Results on the Comparison Between the UbiCARS Framework and CARSKIT

Question: *Based on my current understanding on how the Java Recommendation Engine CARSKIT works, I believe that I would be able to develop a full Recommender System for an e-store by directly using the Recommendation Engine (CARSKIT), WITHOUT using the UbiCARS Modelling Framework in…*

Figure 48 shows the distribution of participants' responses to the question. 85% of participants responded within a few days or more. 10% responded they would estimate needing a few months. Three participants, corresponding to 15% of the group, responded they would be able to do it within a day. Interestingly, one of them is not a developer and is in fact the only participant in the survey who had stated of having no programming experience at all. As this person is, according to his declared capacity, an ecommerce store administrator, it may be the case that such a task would be undertaken by his developer(s). Thus, only two of the participants (10%) estimated that they could by themselves develop a full recommender system for an e-store without using the UbiCARS MDD Framework within a day.

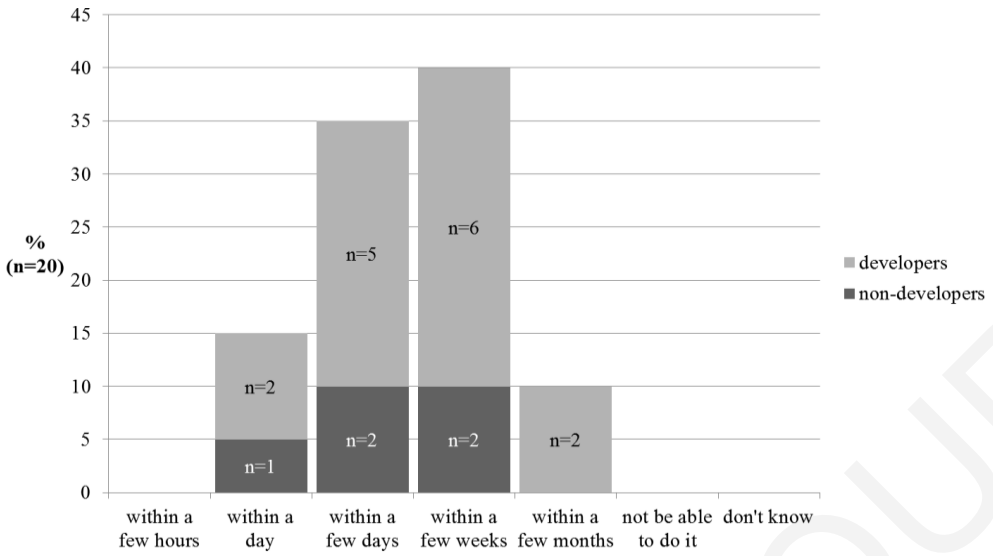

Figure 48: Distribution of participants' responses to the question: *"Based on my current understanding on how the Java Recommendation Engine CARSKIT works, I believe that I would be able to develop a full Recommender System for an e-store by directly using the Recommendation Engine (CARSKIT), WITHOUT using the UbiCARS Modelling Framework"*.

Question: *I believe that it would be easier for me to develop Recommender Systems by using the Java Recommendation Engine CARSKIT instead of the UbiCARS Modelling Framework.*

Mean and Standard Deviation are: mean = 1.68, σ = 1.00. On average, participants' responses lie between strong disagreement and disagreement. It is therefore the case that the model-driven development method of the UbiCARS Modelling Framework is perceived to be easier for developing recommender systems, than using the CARSKIT recommendation engine in a manual manner, by writing code and building the data models. Figure 49 depicts that 5.2% (one developer) responded neutrally. Note that only 19 participants have responded to this question.

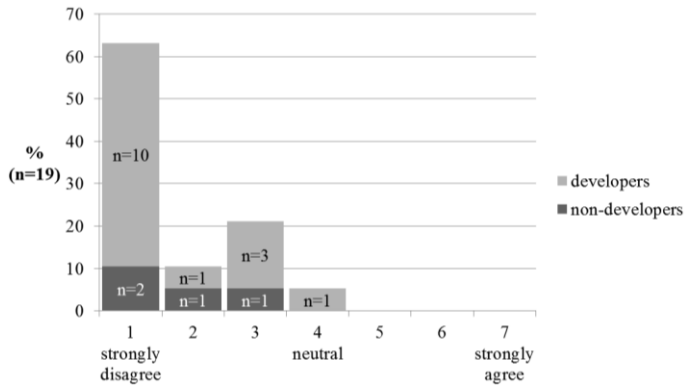

Figure 49: Agreement distribution of participants to the statement *"I believe that it would be easier for me to develop Recommender Systems by using the Java Recommendation Engine CARSKIT instead of the UbiCARS Modelling Framework"*.

Results on Participants' Future Use Intention

Question: *Do you believe that you would revisit the UbiCARS Framework within a week's time if it was available for you to use?*

Mean and Standard Deviation are: mean = 5.95, σ = 1.36. On majority, participants' responses are positive. Standard deviation indicates a wide spread of values in responses, something that is shown in Figure 50: one developer responded by strong disagreeing. The developer, in overall, responded positively to very positively to other questions. A reason for this divergence from other responses may be the fact that this person does not develop recommender systems in her daily work routine; therefore, she would not need to actually use the framework even if it had been available.

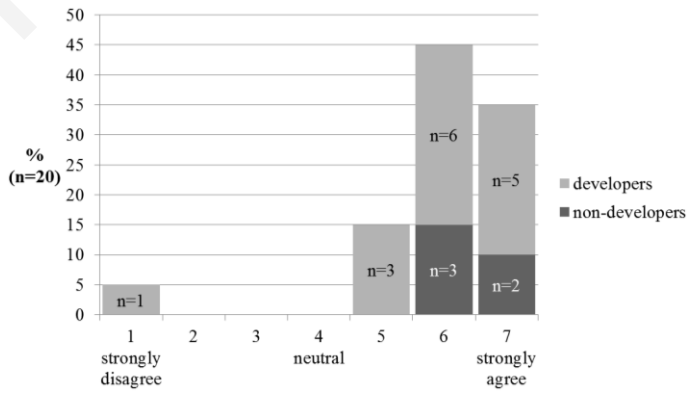

Figure 50: Agreement distribution of participants to the statement *"Do you believe that you would revisit the UbiCARS Framework within a week's time if it was available for you to use?"*

Question: *Do you believe that you would revisit the UbiCARS Framework regularly if it was available for you to use?*

Mean and Standard Deviation are: mean = 5.65, σ = 1.30. On average, participants agree see also Figure 51). Mean value lies between slight agreement and agreement. As with the previous question, one developer strongly disagrees, who is in fact the same person. We believe that the same reasons for her decision also apply here.

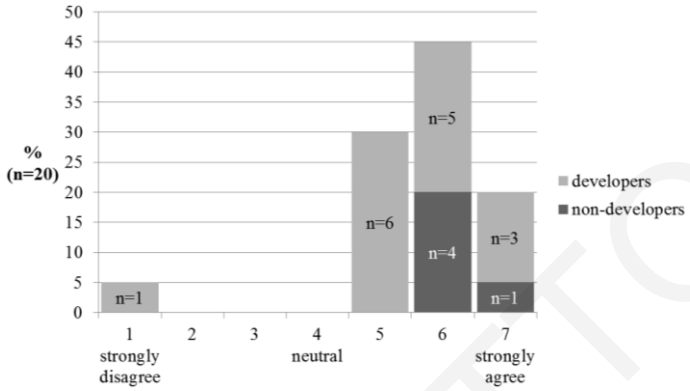

Figure 51: Agreement distribution of participants to the statement *"Do you believe that you would revisit the UbiCARS Framework regularly if it was available for you to use?"*

Results on Participants' Mental Effort and Feelings Towards the Framework and the MDD Methodology

Question: *Do you believe that it does not require a lot of mental effort to interact with the UbiCARS Framework?*

Mean and Standard Deviation are: mean = 6.2, σ = 0.95. Figure 52 depicts that half of the participants strongly agree.

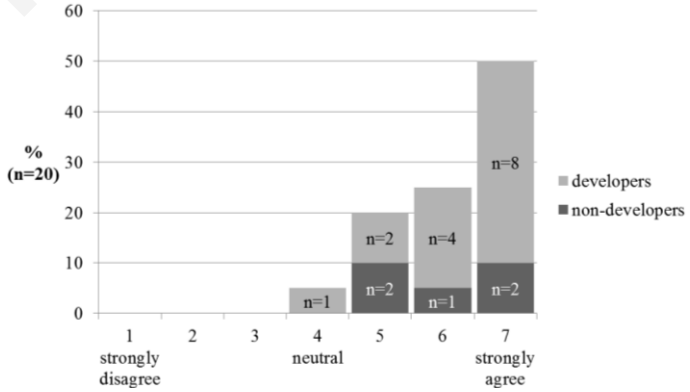

Figure 52: Agreement distribution of participants to the statement *"Do you believe that it does not require a lot of mental effort to interact with the UbiCARS Framework?"*

Question: *Do you believe that using a Model-Driven Development approach through the UbiCARS Framework to develop Recommender Systems is a good idea?*

Mean and Standard Deviation are: mean = 6.75, σ = 0.44. Strongly positive were the responses of all non-developers, as well as of 66.7% of developers (see Figure 53). Mean value leans towards strong agreement with low standard deviation.

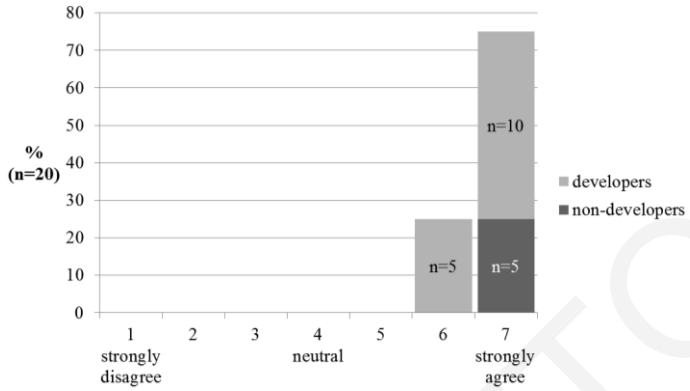

Figure 53: Agreement distribution of participants to the statement *"Do you believe that using a Model-Driven Development approach through the UbiCARS Framework to develop Recommender Systems is a good idea?"*

Question: *Do you feel positive toward the UbiCARS Framework?*

Mean and Standard Deviation are: mean = 6.8, σ = 0.41. It is beyond doubt that participants feel positive towards the framework. Mean is very close to strong agreement and standard deviation is low, indicating that values are concentrated around 6.8. Figure 54 shows that 80% of the participants strongly agree to the statement.

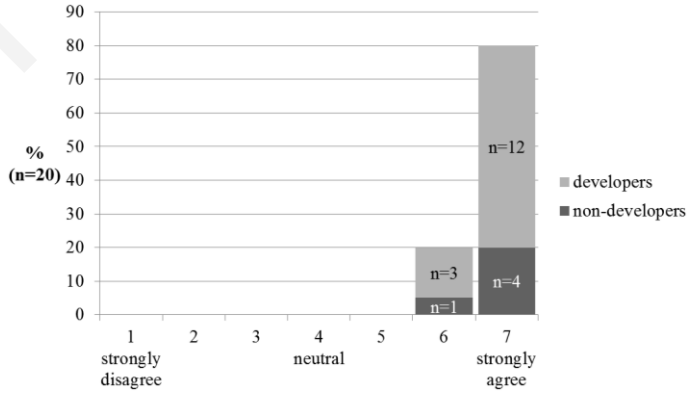

Figure 54: Agreement distribution of participants to the statement *"Do you feel positive toward the UbiCARS Framework?"*

Results on Participants' Interaction with the UbiCARS Framework from the UEQ Questionnaire

Figure 55 depicts the mean values of all participants' responses to the question: *"My interaction with the UbiCARS Framework was…"*. The question is in the semantic differential form (see sub-section "UEQ questionnaire" in Section 6.2.1), where each item is represented by two terms with opposite meanings. The scale is from -3 to +3 (see Figure 55). For the intermediate values 2 and -2 we use the word "fairly" and for 1 and -1 the words "slightly". Table 12 shows the standard deviation for the four items.

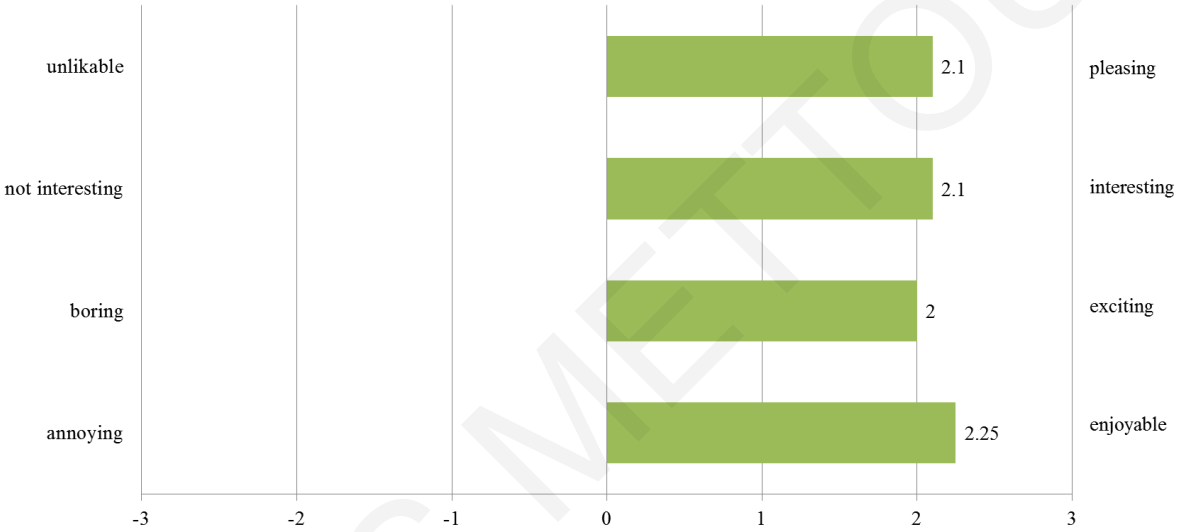

Figure 55: Mean values on total participants' agreement (n=20) with the UEQ items: unlikable/pleasing; not interesting/interesting; boring/exciting; annoying/enjoyable (semantic differential presentation).

Table 12: Standard Deviation for UEQ items for all participants.

Item	**σ**
pleasing	1.29
interesting	1.83
exciting	0.79
enjoyable	0.85

The figure and table indicate that, on average, participants found the framework to be fairly enjoyable, fairly exciting, fairly pleasing and fairly interesting. Mean values are 2 or higher and standard deviation for enjoyable and exciting is 0.85 or lower. With regard to pleasing and interesting, while mean values are high, indicating positive responses, the values for

standard deviation are high as well, especially on the item "interesting". Figure 56 and Figure 57 show the distribution of participants' responses on the two items "pleasing" and "interesting". Figure 56 shows that one developer stated that his/her interaction with the UbiCARS Framework was fairly unlikable, while another one stated that it was neutral. The majority of the remaining 18 participants stated having a pleasing interaction with the framework, including the total number of non-developers.

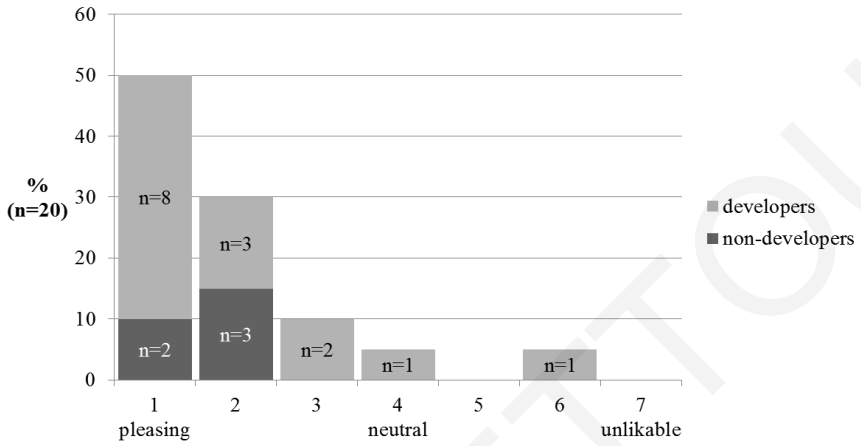

Figure 56: Distribution of participant responses on the item *Pleasing/Unlikable.*

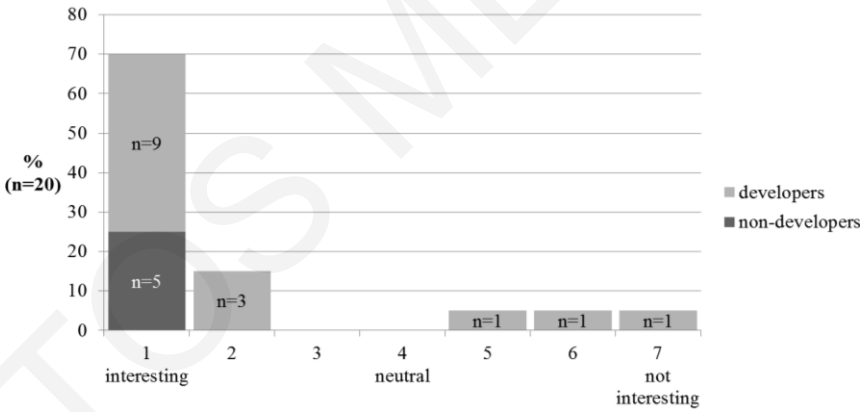

Figure 57: Distribution of participant responses on the item *Interesting/Not Interesting.*

Regarding the item "interesting", Figure 57 depicts that 85% of participants, including 80% of developers, responded positively. The remaining 15% of the participants, all developers, were on the negative scale (not interesting).

Figure 58 shows the mean values of the question for the groups of developers and non-developers. It is shown that developers have found the framework to be fairly enjoyable, fairly exciting, fairly pleasing and fairly interesting. Non-developers have found the framework to be interesting, pleasing, enjoyable and fairly exciting.

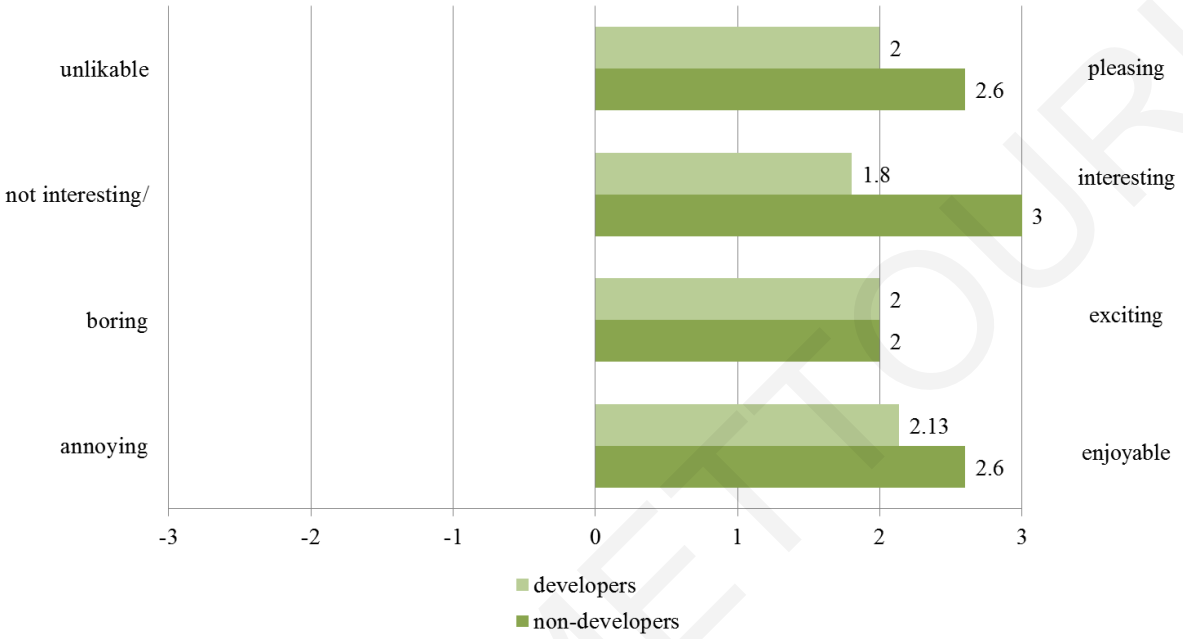

Figure 58: Mean values on developers' (n=15) and non-developers' (n=5) agreement with the UEQ items: unlikable/pleasing; not interesting/interesting; boring/exciting; annoying/enjoyable (semantic differential presentation).

Evaluation Report and User Comments

During the evaluation sessions we have been recording issues, problems and any ambiguities with the questionnaire instructions in a report.

Table 13 lists these issues per session. In specific, the table lists participants' comments on the description of the tasks in the questionnaire, as well as any errors or bugs identified while interacting with the system UI. Concerning participants' interaction with the modelling editor, no bugs or errors have been reported. Nevertheless, participants had comments on the editor which have been recorded and are discussed below.

Table 13: Summative report on participants' feedback during the evaluation sessions (sessions with zeros in all columns have been omitted).

Issue #	Session #	Comments on Modelling Editor	Errors in System UI	Comments on Tasks Description
1	**1**	0	1	10
2	**2**	2	0	1
3	**3**	0	1	2
4	**4**	0	1	2
5	**5**	0	0	3
6	**10**	0	1	2
7	**13**	1	0	1
8	**14**	1	0	0
9	**16**	0	1	0
10	**18**	1	0	0
11	**19**	0	0	2
12	**20**	1	0	0
TOTAL	**12 Sessions**	**5**	**5**	**23**

Comments on the Modelling Editor

An MDD expert suggested that the editor should be able to provide assistive information, as soon as an element is dropped on the canvas. For example, by double clicking on the newly created element, information could be displayed on what this element is about and what each property means, specifically for the Recommender Systems domain. Moreover, the expert did not like that a (small) number of element properties, while available, they were left empty during the evaluation sessions. He suggested these properties be disabled or removed if not required, as it is not apparent for the user if they should be completed or whether they would cause any errors if left empty. As a response, we state that these properties were left empty only for the purposes of the evaluation sessions so that these do not have too much duration, as well as make the process simple by only having participants completing a minimum number of element properties. In normal usage of the framework by practitioners, these properties would have been filled, or not appeared on the canvas. Being empty does not cause any problems. During the evaluation they were not removed as it was perceived by the author important that participants are able to observe all

available functionality within the modelling editor, even if not used during the evaluation sessions. However, the expert's comment is valid in that it is indeed necessary to inform participants that leaving these parameters empty causes no errors to the execution of the system.

On another subject, the same expert mentioned that it would be best if the names of the elements that users add on the editor canvas did not include any spaces, as is the case with variables in programming. On this subject, a developer would prefer having in parenthesis the exact full name of the tool, as this was mentioned in the tasks description (e.g. having `createStorageElement` instead of `createStorage`). Another MDD expert mentioned that the names of the tools in the editor toolbox used for creating elements need not mention the word "create", e.g., instead of "`createCARS`", the tool for creating CARS should be renamed to "`CARS`".

A developer mentioned renaming the toolbox item "`createEngine`" to "`createRecommendationEngine`", so that it is clearer and it better matches with the task description. The same participant suggested including a different icon for the tool that is used for linking context elements with database resource elements, so that it can be distinguished from the rest of the linking tools that are used for linking other elements between them.

A participant (PhD Student in Computer Science) suggested including search functionality for easy access of tools in the toolbox.

A researcher would like to be able to copy-paste and duplicate elements in the editor canvas. A developer expressed the same desire regarding the copy-paste functionality.

Errors in System UI

After completing the modelling part of the evaluation, participants used the system UI to parse their model and configure the e-store. In a number of cases where the model included errors, the parsing failed. Errors in the model resulted from faulty instructions (in earlier sessions) or misinterpretation of instructions by participants, or even user errors (e.g. in one occasion the participant misread the instructions and typed an incorrect database resource value in an element that caused the parser to fail, since such a resource did not actually exist in the database). In cases of user error, we did not attribute the error to the participant; rather, we looked deeper into the problem and attributed such errors to issues with the instructions or the framework (e.g., in the abovementioned example we have

rephrased the instructions to minimise the possibility the particular user error is reproduced in future evaluations).

In another case, a participant did not notice she had to click on the "compute recommendations" link to initiate computation of recommendations. As such, she could not proceed with the session. We pointed out the link. This problem did not occur with other participants. As a result of this error, the link was redesigned so as to be more noticeable.

Another participant (developer) did not realise that timestamps in database resource elements are used as resources in the database infrastructure and, despite that in the instructions it was mentioned that, for timestamps, spaces between words are not allowed, he added spaces, causing the parser to fail. The system UI has been adapted to appropriately format timestamps to cope for cases where users do not provide values according to the requirements.

Another developer had omitted to include any values for the `RSSI_minDetectionThreshold` and `MaxTimeInterval_BSD` parameters of the `StayingTime` element, although he was instructed to do so in a task, causing the system UI parser to fail. The bug has been fixed so that, when values for these parameters are omitted, the system utilises a set of default values in order to proceed with execution.

Comments on Tasks Description

From Table 13 it is observed that, during the first session, a total of 10 comments were made by the participant regarding problems with the tasks description. It is important to note that these problems were not identified during the pilot study. The problems were in majority minor comments concerning expressions used in tasks description. For example, a comment was to mention in the instructions that the links between elements in the editor are draggable, or, that all database tables with the prefix "wp_" are default tables of Wordpress. In sessions following session one, the number of comments on tasks description was noticeably reduced, as problems had been already corrected.

General Comments

A developer mentioned that the framework had a very nice and clean interface. He mentioned it was nice to work with and that he would like to use it in the future.

Another developer would like to know more information about how to install the framework's plugins on another pre-existing Wordpress e-store. He expressed his confidence that this would be of great interest to many developers.

The participant with the e-commerce store administrator capacity stated that, to him, it was obvious that the tool was useful. He said that, for himself, this type of evaluation session may not be very relevant because he is not a developer and has no development experience. Furthermore, to use the framework in a real scenario he would have his developers do the work for him. However, he noted that, even as not a developer, he was able to conduct the session with success, even if he had to request some information from the author on technical matters regarding the database. In addition, the participant expressed the confidence that, even if recommender system tools and plugins had existed for Wordpress or other platforms, these would not be as intelligent as this tool (i.e. the UbiCARS framework). While closing the session, the participant stated that he would be interested in testing the framework on their company's e-store with real customer data, when the e-store is launched. Furthermore, he was interested in setting up a meeting with his lead developer to test the framework in real settings.

A developer mentioned that, while the user interface of the system UI needed attention, he believed that the framework can have commercial value with some modifications, but he did not specify more on the subject. The participant liked very much the modelling editor and the modelling process offered by the framework.

A researcher said that the tool was useful and that he liked it.

The PhD Student in Computer Science mentioned that the framework was very easy to use and that it saves a lot of time and effort when someone wants to create a recommender system. He said he would definitely use it again if he had the chance. He also expressed the opinion that the recommendations explanation subject would be interesting for further research (as future work).

A developer stated that the framework appears to be a very helpful tool for developers, but he cannot be 100% confident on how helpful, as he has never used CARSKIT in order to be in the position to compare the two tools.

Another developer, after mentioning that she has extended knowledge on e-store development and databases, she stated being very confident in saying that, by using the UbiCARS framework, the development of such applications (i.e. recommender systems) is conducted much faster than by coding. She was impressed in that the process only took her about an hour to develop and test a full recommender system. She explicitly mentioned that it would be difficult to build the entire infrastructure manually without the UbiCARS

Framework. It is interesting to note here that the same participant was one of the two developers who in the question *"Based on my current understanding on how the CARSKIT works, I believe that I would be able to develop a full Recommender System for an e-store without using the UbiCARS Framework by directly using CARSKIT"* responded "*within a day*". It is therefore very probable that the participant unintentionally responded to the latter question incorrectly, thinking that the question was actually asking for an estimation on the time required to develop a recommender system by using the UbiCARS Framework (this was also the case in two more occasions, where participants later corrected their response in realisation of the error).

One of the two MDD experts reported that the modelling environment is very easy to use and that all element names are self-explainable. On the task descriptions, the same expert reported that they make it easy for the user to understand recommender systems. The expert had concerns though, on whether the framework would be able to compile datasets from pre-existent data on customer interaction with products in an e-store, in cases where these data were produced through custom processes. Our response is that, in case custom processes have been used by a developer/practitioner, it is possible that the format of the data stored is not compatible with the infrastructure the UbiCARS Framework builds. In such cases, developer supervision of the process and also intervention may be required.

6.3 Answering to the Research Questions

RQ1: Does the UbiCARS MDD Framework reduce the development time (expedite the development) of state-of-the-art Ubiquitous Context-Aware Recommender Systems for commerce for developers that are non-experts in RS?

From the results of the evaluation session, it can be stated that:

- The UbiCARS MDD Framework enables the development of state-of-the-art UbiCARS for commerce by developers that are non-experts in RSs on average in 58.5 minutes. This time is significantly less than the perceived time needed using the baseline recommendation engine, for which participants estimated needing a few days to a few weeks to use it and to manually develop the required infrastructure and data models. However, the time reported for the UbiCARS

Framework (58.6 minutes) applies in cases where no additional explicit or implicit user feedback techniques need to be developed, as was the case during the UbiCARS MDD Framework evaluation sessions. In cases where additional techniques need to be developed, the UbiCARS DSML facilitates such techniques via the explicit and implicit `NewUserFeedback` elements (see Section 5.2.2), while the UbiCARS Framework enables inclusion of the new elements in the UbiCARS infrastructure that is deployed on e-stores; nevertheless, the developer will need to contribute in the process by manually developing the modules that will acquire the user-product interaction data from e-stores and/or the physical store. This part of implementation is unavoidable and corresponds only to a small part of the implementation needed, had the UbiCARS Framework not been used.

- The UbiCARS MDD Framework is considered by evaluation participants to:
  - Enable them to develop recommender systems *more quickly*: this statement refers to development time.
  - *Increase their productivity* in developing recommender systems: this statement refers to finishing tasks related to recommender systems development more quickly or at a more rapid rate.

RQ2: Does the UbiCARS MDD Framework reduce development complexity in developing state-of-the-art Ubiquitous Context-Aware Recommender Systems for commerce for developers that are non-experts in RS by:

1. Reducing the lines of code and DB queries developers need to write?
2. Supporting software maintainability and software evolution in terms of future enhancements?

From the results of the evaluation session, it can be stated that:

- All participants, while being non-experts in RSs, were able to complete all tasks successfully to develop a state-of-the-art UbiCARS for commerce in less than one hour on average, without the need to write code or DB queries. Development complexity was reduced by abstracting the technical details, as well as abstracting recommender systems domain specifics.
- In terms of software maintainability and evolution: Task 14 of main evaluation was designed so as to require from participants, after completing e-store configuration through the system UI, to re-execute the whole process, i.e., to go back to the

modelling editor to enhance the model by adding context, in order to also include customer's location while staying in front of products in the physical store, to enable context-aware computation of recommendations. The task was successfully completed by all participants in 6.6 minutes on average, while the resulting enhanced model was able to be successfully parsed by the system UI and re-configure the e-store to handle context. This task acts as proof that non-RSs expert users are able to alter their initial model to add (or remove) elements and reconfigure their e-stores in minutes; hence, the UbiCARS MDD Framework supports software maintainability and evolution.

RQ3a: Does the UbiCARS MDD Framework enhance recommendation accuracy when used for the development of Ubiquitous Context-Aware Recommender Systems for commerce by tracking and utilising the user behaviour in both physical and online settings?

The UbiCARS MDD Framework enables the combination of explicit and implicit user feedback techniques from the e-stores and physical stores and the compilation of (at least) five datasets as follows: the Ratings dataset, the Purchase History dataset, the Browsing History dataset, the Staying Time dataset and the Scanning dataset. From each of these datasets, recommendations are computed. Moreover, combinations of datasets into one larger one are possible; however, proper scaling of user feedback values is required. As the common practise in RSs is to utilise user ratings on products for recommendation computation (i.e. the ratings dataset), it is argued that the additional datasets offered and utilised by the UbiCARS Framework in the recommendation computation process have the *potential* to increase recommendation accuracy, as additional user feedback information for the same users is used in the recommendation process. We acknowledge that the abovementioned argumentation does not prove that recommendation accuracy is improved; hence, the research question is not positively answered. However, there is a strong indication that the research question can be positively answered. One way to answer the question is by engaging real users as customers to interact with products and then receive the recommendations, which they can either follow (recommendations were perceived as useful) or not (recommendations were not perceived as useful). However, this kind of evaluation is out of the scope of this thesis.

RQ3b: Does the UbiCARS MDD Framework increase the availability of recommendations when used for the development of Ubiquitous Context-Aware Recommender Systems for commerce by tracking and utilising the user behaviour in both physical and online settings?

As the UbiCARS MDD Framework enables the compilation of datasets from different user feedback acquisition techniques, and with the potential to facilitate more user feedback acquisition techniques (and thus even more datasets), from each of which recommendations can be computed, we argue that recommendation availability is increased in comparison to the common practise in RSs to utilise one dataset, i.e. user ratings on products. During the evaluation sessions, participants were able to produce all four datasets, as discussed in Sections 6.1 and 6.2.1, and feed the recommendation engine with the dataset of their choice. Each dataset offered different recommendations to each user.

In real settings, a developer might be interested to do the same for a number of reasons:

- In case customers do not provide adequate feedback for a dataset to be of acceptable quality, another dataset can be used, or a combination of them.
- Address the cold start problem: for RSs to be able to produce meaningful recommendations, user interaction with items first needs to take place. In case a dataset is not of acceptable size, another one can be used instead, or a combination of them.
- For experimentation reasons: experiment with different datasets and observe which recommendations users prefer.

Chapter 7

Discussion and Future Work

7.1 Discussion

The absence of works that use model-driven development methods for the design, development and deployment of Ubiquitous Context-Aware Recommender Systems (UbiCARS) that specifically aim for technical abstraction, reduction of development complexity, and development expedition of the aforementioned systems, has motivated the work presented in this thesis, together with the lack of methodologies that combine explicit and implicit user feedback techniques from both the physical commerce scenario and the electronic commerce scenario, aiming to enhance recommendation accuracy and availability in these scenarios. This thesis proposes the UbiCARS MDD Framework to address the abovementioned limitations.

In particular, after discussing the research background in Chapter 2, we have presented related work in Chapter 3 in terms of explicit and implicit user feedback acquisition techniques for the online scenario and the physical scenario, available recommendation frameworks and engines, as well as literature works on using modelling and other software engineering techniques to tackle RS development complexity. The chapter completed with a discussion about prior works of the author on aiding the development of CARS, that act as antecedent work to the research presented in this thesis. The UbiCARS framework was designed, among other, to address limitations of these works. Chapter 4 introduced and defined the Ubiquitous Context-Aware Recommender Systems – UbiCARS [18], which, in this work, enable user feedback acquisition and computation of context-aware recommendations for both the physical store (ubiquitous scenario) and the e-store (online scenario).

Chapter 5 presented the UbiCARS MDD Framework [36]. In particular, the UbiCARS MDD Methodology and the UbiCARS DSML were discussed, as well as the methodology with which the framework combines explicit and implicit user feedback techniques from e-commerce and physical commerce. The DSML is designed to offer high level of abstraction from the technical details and domain specifics, which, as it was proved by our evaluation, was a key factor for reducing the development complexity and expediting the development of ubiquitous context-aware recommender systems by developers who had no recommender systems experience. In the final section of the chapter, the UbiCARS Framework architecture was presented.

The UbiCARS Framework demonstrator was presented in the first section of Chapter 6. The UbiCARS modelling editor, its tools, and an instance model were depicted. System implementation was discussed, including testing and evaluation setup. The second part of Chapter 6 described the methodology for the evaluation of the UbiCARS MDD Framework and the results. The results were overall very positive. In about one hour on average, participants that were not experts in the Recommender Systems domain were able to successfully utilise the capabilities of the UbiCARS DSML, the modelling editor and the system UI, to design, configure and deploy UbiCARS on example e-stores. Participants stated to have understood the produced results. The UbiCARS MDD Framework was perceived by participants to be, among other, useful, quick, to improve developers' performance, to increase developers' productivity, to reduce developers' effort and to be easy to use. Participants agree that it does not require a lot of mental effort to interact with the framework, and, furthermore, stated feeling positive toward the UbiCARS Framework. Using the framework to develop UbiCARS was considered by participants to take considerable less amount of time than using the baseline recommendation engine that requires from developers to manually develop the infrastructure and the data models.

Other important findings of the evaluation include that the UbiCARS MDD Framework supports software maintainability and evolution in terms of future enhancements of the model and re-configuration of the e-stores, as well as that participants strongly agree that using a model-driven development approach through the UbiCARS Framework to develop recommender systems is a good idea.

The positive evaluation results show the potential of the UbiCARS MDD Framework. Participants appreciated the abstraction from technical details and Recommender Systems domain specifics that the framework offers, and liked the fact that they did not need to write code to develop UbiCARS, neither to learn details about recommender systems and algorithms. The expedited development of UbiCARS offered by

the framework, in comparison to any manual method that requires coding and comprehension of Recommender Systems domain specifics, was a clear benefit mentioned by participants.

Use of the UbiCARS MDD Framework by RSs Experts

While the framework is beneficial for developers that are non-RSs experts, it can potentially be used effectively by RSs experts as well, to easily and expeditiously configure and deploy UbiCARS on e-stores. Experts are enabled to quickly test the accuracy of various recommendation algorithms on existing e-stores with many users and data, by firstly integrating these algorithms with the framework. The integration is done through the modelling editor via the parameter "other" of the `RecommendationEngineEn` enumeration. The framework will directly use the new recommendation algorithm, provided that the framework is directed to execute the particular algorithm, and that the algorithm is configured to use the specific datasets compiled by the framework.

The possibility to effectively switch between integrated algorithms through the UbiCARS DSML and modelling editor is also provided. Moreover, RSs experts can develop their own algorithms and integrate them directly within the UbiCARS Framework, as discussed above. Another possibility would be the RSs experts to include their algorithm within CARSKIT recommendation engine and provide an abstraction for the algorithm in the modelling layer of the framework by adding the new algorithm in the `RecommendatioAlgorithmEn` enumeration (see Figure 11), so that it is available for use to the UbiCARS models.

7.2 Future Work

It is important to investigate whether recommendation algorithm specific details could be included in the DSML through new elements and parameters, while two critical requirements are met:

- The abstraction of the DSML is maintained: such elements should facilitate a cross-recommendation engine design.
- The complexity of the DSML is not increased: when algorithmic details are abstracted in the DSML, the complexity of the DSML increases, as complex

Recommender Systems domain specific parameters are now available to be used by designers, who need to be assisted and facilitated in using them appropriately. Too complex DSML result in designers not using them, or, in the best case scenario, not using their entire potential.

There is much research on the subject on recommendation explanations (see Section 2.5), as these can contribute to many factors, among other to system transparency, trust, effectiveness and satisfaction [89, 90, 91, 92]. In this thesis work, recommendation explanations have been included in the DSML and can drive enablement or disablement of simple explanations on e-stores. However, explanations can be facilitated to a greater extend within the DSML, such as by abstracting the different explanation techniques, making them available in the model.

In Section 2.6.1 the New EU regulation on the General Data Protection Regulation (GDPR) was discussed, as well as the impact it may have on recommender systems. In terms of future work, we propose the extension of the UbiCARS DSML with GDPR compliance enabling elements, aiming to guide the design and development of GDPR compliant UbiCARS. Since UbiCARS utilise user behaviour data from the e-store and the physical store, such data are inherently sensitive, e.g. user location, and need to be handled in specific processes.

In terms of privacy concerns, it is argued that privacy-preserving methods need to be utilised by the framework. As Al-Rubaie and Chang note in [150], ML tasks consider three roles: the input party (data owners, i.e. users), the computation party (i.e. where ML tasks are executed) and the results' party (where recommendations are displayed). As they note, privacy is preserved only when all three roles are assumed by the same entity; in all other cases where these roles are distributed between two or more entities, privacy-preserving methods need to be applied. Threats exist, not only while transferring data from user devices to the computational party, which can be conducted over secure channels, but most importantly, while the data are stored in the computational party, either as raw data, or as feature vectors extracted from raw data [150]. Even when feature vectors are used instead of raw data, threats appear in the form of reconstruction attacks, where users' private data can be reconstructed from these feature vectors [150]. Based on the ML algorithm used, other risks may appear as well, such as [150] model inversion attacks, membership inference attacks, and de-anonymisation (although removing any personal identifiers of

users in the dataset is a good practise for protecting users' privacy, successful attempts have been made to identify the users from anonymised datasets). In the case of the UbiCARS Framework, while the computation party and the results party are realised by the same entity (the recommendation engine of the framework), these are functioning remotely from users (the input party). Therefore, privacy-preserving methods are needed in respect to the following functionality:

- Data transfers from the UbiCARS mobile app to the CARS server-side system need to be conducted over secured channels;
- Data need to be encrypted and/or anonymised before they reside on the server.

Moreover, specific privacy-preserving techniques need to be applied, in accordance with the particular ML method used for recommendation computation, which also dictates whether the computation party and the results party will be executed by the same entity (this may not always be the case). Privacy-preserving ML techniques for recommender systems have already been presented in the literature that propose training a machine learning model (including Matrix Factorisation models) over acquired user feedback data in a privacy-preserving manner, and then apply the model to generate personalised recommendations [151]. The aforementioned will be considered as future work.

Enhancements to the modelling editor have been proposed by participants in terms of providing assistive information during element creation on canvas. Information should concern the newly created element and its relationships with other elements in the DSML, as well as the role of the element in the recommendation process and how it can be utilised. Such information could increase further the system learnability, while it will make the framework more easy to use.

Future evaluation activities include a more detailed investigation of the framework's usability aspects in terms of the modelling editor. In particular, by utilising eye tracking devices and software, we aim to track participants' eye focus and eye movement while using the editor, to infer possible usability problems. Heat maps and saccade pathways can be created from eye tracking to find the exact spots on the editor that participants have been focusing on and for how long, as well as to track the routes of eye movements of participants [152]. Such data indicate where the attention of participants is on the editor, and whether participants are constantly searching for things such as tools and elements, in which case usability improvements will be needed.

Application in Other Domains

The potential of applying the UbiCARS MDD Framework and methodology in domains other than commerce is discussed in this section. In order for this to be feasible, there must be a need for utilisation of ubiquitous user feedback acquisition techniques, which call for the application of UbiCARS. Commerce is one of the most important domains of application of UbiCARS, as activities of users in physical retail stores are conducted very often in their daily lives and have prominence importance, as discussed in this thesis.

Another potential domain where the UbiCARS Framework can be applied, provided that the DSML is altered to reflect the particular domain, is *Tourism*. In tourism, Point-of-Interest (POI) recommender systems recommend POIs to users based on their preferences. POI recommender systems can be utilised in a variety of mobile and web applications for tourism, touristic websites, travel agency systems, etc. A potential application of the UbiCARS Framework in tourism would enable the acquisition of ubiquitous user feedback data from the physical site (archaeological sites, museums, etc.) by means of users' total staying time near a POI, as well as users' number of visits to a POI. These data could be used together with online user feedback data from the tourism website (user ratings on POIs, browsing history for POIs). The UbiCARS app for tourism would enable the ubiquitous interaction of users with POIs on location.

References

1. Ricci, F., Rokach, L., Shapira, B., Kantor, P. B.: Recommender Systems Handbook. 1st ed. New York, NY, USA: Springer-Verlag New York, Inc. (2010).
2. Walter, F. E., Battiston, S., Yildirim, M., Schweitzer, F.: Moving recommender systems from on-line commerce to retail stores. Inf Syst E-Bus Manage, v.10, pp.367–393 (2012).
3. Schafer, J.B., Konstan, J.A., Riedl, J.: E-Commerce Recommendation Applications. Data Mining and Knowledge Discovery vol. 5, pp. 115-153 (2001) DOI https://doi.org/10.1023/A:1009804230409.
4. Bell, R., Koren, Y., Volinsky, C.: Matrix Factorization Techniques for Recommender Systems. In Computer, vol. 42, pp. 30-37 (2009). DOI 10.1109/MC.2009.263.
5. Hu, Y., Koren, Y., Volinsky, C.: Collaborative Filtering for Implicit Feedback Datasets. In Proc. of the 8th IEEE International Conference on Data Mining, pp. 263–272 (2008).
6. Google Cloud Machine Learning (ML), https://cloud.google.com/ml-engine, Last Accessed: 25/11/2018.
7. SLI Systems Recommender, https://www.sli-systems.com/, Last Accessed: 25/11/2018.
8. Azure Machine Learning Studio, https://azure.microsoft.com/en-us/services/machine-learning-studio/, Last Accessed: 25/11/2018.
9. Amazon Machine Learning, https://aws.amazon.com/machine-learning/, Last Accessed: 25/11/2018.
10. SuggestGrid, https://www.suggestgrid.com, Last Accessed: 10/05/2017.
11. Yusp, https://www.yusp.com/, Last Accessed: 25/11/2018.
12. Aldrich, S. E.: Recommender Systems in Commercial Use. In AI Magazine, vol. 32, no. 3, pp. 28–34 (2011).

13. Fang, B., Liao, S., Xu, K., Cheng, H., Zhu, C., Chen, H.: A novel mobile recommender system for indoor shopping. Expert Systems with Applications. Volume 39, Issue 15, pp. 11992-12000 (2012).
14. Statista, "Retail e-commerce sales worldwide from 2014 to 2021 (in billion U.S. dollars)", https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/, Last Accessed: 25/11/2018.
15. International Council of Shopping Centers-ICSC, "Shopping Centers: America's First and Foremost Marketplace", https://www.icsc.org/uploads/research/general/America-Marketplace.pdf, Last Accessed: 25/11/2018.
16. Chen, C.-C., Huang, T.-C., Park, J. J., Yen, N. Y.: Real-time smartphone sensing and recommendations towards context-awareness shopping. Multimedia Systems, vol. 21, no. 1, pp. 61–72 (2015).
17. Want, R., Pering, T.: System challenges for ubiquitous & pervasive computing. In Proc. The 27th international conference on Software engineering, ACM 9-14 (2005).
18. Mettouris, C., Papadopoulos, G.A: Ubiquitous recommender systems. Computing, Springer, V. 96 (3), pp. 223-257 (2014).
19. Adomavicius, G., Tuzhilin, A.: Context-Aware Recommender Systems. In: Ricci F., Rokach L., Shapira B., Kantor P. (eds) Recommender Systems Handbook. Springer, Boston, MA, pp. 217-253 (2011).
20. So, W. T., Yada, K.: A Framework of Recommendation System Based on In-store Behavior. in Proceedings of the 4th Multidisciplinary International Social Networks Conference, New York, NY, USA, p. 33:1–33:4 (2017).
21. Hoyos, J.R., García-Molina, J., Botía, J.A.: A domain-specific language for context modeling in context-aware systems. J. Syst. Softw. Volume 86, Issue 11, pp. 2890-2905 (2013) DOI http://dx.doi.org/10.1016/j.jss.2013.07.008.
22. Ou, S., Georgalas, N., Azmoodeh, M., Yang, K., Sun, X.: A Model Driven Integration Architecture for Ontology-Based Context Modelling and Context-Aware Application Development. In: Rensink A., Warmer J. (eds) Model Driven Architecture – Foundations and Applications. ECMDA-FA 2006. Lecture Notes in Computer Science, vol 4066. Springer, Berlin, Heidelberg (2006).
23. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley, Boston, United States. (2008).
24. Easyrec, easyrec.org/, Last Accessed: 25/11/2018.
25. Lenskit, https:lenskit.org/, Last Accessed: 25/11/2018.
26. Librec, www.librec.net, Last Accessed: 25/11/2018.

27. Apache Mahout, http://mahout.apache.org/, Last Accessed 25/11/2018.
28. Duine, http://www.duineframework.org/, Last Accessed 25/11/2018.
29. GraphLab Create (Turi create intelligence), https://turi.com/index.html, Last Accessed 25/11/2018.
30. MyMediaLite Recommender System Library, http://mymedialite.net/, Last Accessed 25/11/2018.
31. Apache Prediction IO, https://prediction.io/, Last Accessed 25/11/2018.
32. Inzunza, S., Juárez-Ramírez, R., Jiménez, S.: User Modeling Framework for Context-Aware Recommender Systems. In Recent Advances in Information Systems and Technologies, pp. 899–908 (2017).
33. Hussein, T., Linder, T., Gaulke, W., Ziegler, J.: Hybreed: A software framework for developing context-aware hybrid recommender systems. User Model User-Adap Inter, vol. 24, no. 1–2, pp. 121–174 (2014).
34. Portugal, I., Alencar, P., Cowan. D.: The use of machine learning algorithms in recommender systems: A systematic review. Expert Systems with Applications, Volume 97, pp 205-227 (2018).
35. Rojas, G., Domínguez, F., Salvatori, S.: Recommender Systems on the Web: A Model-Driven Approach. In Proceedings of the 10th International Conference EC-Web 2009. Linz, Austria. Springer Berlin Heidelberg, pp. 252–263 (2009).
36. Mettouris, C., Achilleos, A.P., Kapitsaki, G.M., Papadopoulos, G.A.: The UbiCARS Model-Driven Framework: Automating Development of Recommender Systems for Commerce. In: Kameas A., Stathis K. (eds) Ambient Intelligence. AmI 2018. Lecture Notes in Computer Science, vol 11249. Springer, Cham, pp. 37-53 (2018), DOI https://doi.org/10.1007/978-3-030-03062-9_3.
37. Mettouris, C., Papadopoulos, G.A.: CARS Context Modelling. In Proceedings of the 9th International Conference on Knowledge, Information and Creativity Support Systems. KICSS 2014. pp. 60-71 (2014).
38. Mettouris, C., Papadopoulos, G.A.: Using Appropriate Context Models for CARS Context Modelling. In Knowledge, Information and Creativity Support Systems, Series "Advs in Intelligent Syst., Computing", Springer, Vol. 416, ISBN 978-3-319-27477-5 (2015).
39. Mettouris, C., Papadopoulos, G.A.: Contextual Modelling in Context-Aware Recommender Systems: a generic approach. In: Haller, A., Huang, G., Huang, Z., Paik, H.-Y., Sheng, Q.Z. (eds.) WISE 2011 and 2012 Combined Workshops. LNCS, vol. 7652, pp. 41-52. Springer, Heidelberg (2013).

40. Mettouris, C., Achilleos, A.P., Papadopoulos, G.A.: A Context Modelling System and Learning Tool for Context-Aware Recommender Systems. In Scaling up Learning for Sustained Impact, D. Hernandez-Leo, T. Ley, R. Klamma, and A. Harrer, Eds. LNCS, Springer Berlin Heidelberg, Volume 8095, pp. 619-620 (2013).
41. Weiser, M.: The Computer for the 21st Century. Scientific American (1991).
42. Dey, A.K., Abowd, G.D., Salber. D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, vol. 16 97-166 (2001).
43. Waller, V., Johnston, R.B.: Making ubiquitous computing available. Communications of the ACM, vol. 52 127-130 (2009).
44. Henricksen. K., Indulska, J., Rakotonirainy, A.: Infrastructure for Pervasive Computing: Challenges. In Proc. WORKSHOP ON PERVASIVE COMPUTING INFORMATIK 01 214-222 (2001).
45. Davies, N., Gellersen, H.: Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. IEEE Pervasive Computing, vol. 1 26-35 (2002).
46. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications, vol. 8 10-17 (2001).
47. Hinze, A., Buchanan, G.: Context-awareness in mobile tourist information systems: challenges for user interaction. In: Proc. International Workshop on Context in mobile HCI at the Conference for 7th International Conference on Human Computer Interaction with Mobile Devices and Services (2005).
48. Kjeldskov. J., Skov, M.B.: Exploring context-awareness for ubiquitous computing in the healthcare domain. Personal Ubiquitous Computing, vol. 11 549-562 (2007).
49. Schilit, B.N., Lamarca, A., Borriello, G., Griswold, W.G., Mcdonald, D., Lazowska, E., Hong, J., Iverson, V.: Challenge: Ubiquitous location-aware computing and the Place Lab initiative. In: Proc. WMASH '03, The 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots 29-35 (2003).
50. Mancini, C., Thomas, K., Rogers, Y., Price, B.A., Jedrzejczyk, L., Bandara, A.K., Joinson, A.N., Nuseibeh, B.: From spaces to places: emerging contexts in mobile privacy. In Proceedings of the 11th international conference on Ubiquitous computing, New York, NY, USA, 2009, pp. 1–10 (2009).
51. Lenders, V., Koukoumidis, E., Zhang, P., Martonosi, M.: Location-based trust for mobile user-generated content: applications, challenges and implementations. In: Proc. The 9th workshop on Mobile computing systems and applications, ACM, 60-64 (2008).

52. Domnitcheva, S.: Location Modeling: State of the Art and Challenges. In: Proc. THE WORKSHOP ON LOCATION MODELING FOR UBIQUITOUS COMPUTING 13-19 (2001).
53. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. ACM Transactions on Information Systems (TOIS), vol. 23 103–145 (2005).
54. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of recommendation algorithms for e-commerce. Proceedings of the 2nd ACM conference on Electronic commerce, Minneapolis, Minnesota, United States, 2000, pp. 158–167 (2000).
55. Burke, R.: Hybrid Recommender Systems: Survey and Experiments, User Modeling and User-Adapted Interaction, vol. 12, no. 4, pp. 331–370, (2002).
56. Karatzoglou, A., Amatriain, X., Baltrunas, L., Oliver, N.: Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering. In Proceedings of the Fourth ACM Conference on Recommender Systems, New York, NY, USA, pp. 79–86 (2010).
57. Nguyen, T. V., Karatzoglou, A., Baltrunas, L.: Gaussian Process Factorization Machines for Context-aware Recommendations. In Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, New York, NY, USA, pp. 63–72 (2014).
58. Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM TRANSACTIONS ON INFORMATION SYSTEMS, vol. 22, p. 143--177 (2004).
59. Karypis, G.: Evaluation of Item-Based Top-N Recommendation Algorithms, Proceedings of the tenth international conference on Information and knowledge management, p. 247--254 (2000).
60. Bilandzic, M., Foth, M., Luca, A.D.: CityFlocks: designing social navigation for urban mobile information systems. In Proc. of the 7th ACM conference on Designing interactive systems, Cape Town, South Africa: ACM 174-183 (2008).
61. Burrell, J., Gay, G.K.: Collectively defining context in a mobile, networked computing environment. CHI '01 extended abstracts on Human factors in computing systems, Seattle, Washington, ACM, 231-232 (2001).
62. Burrell, J., Gay, G.K.: E-Graffiti: Evaluating real-world use of a context-aware system. Interacting with Computers (2002).
63. Tungare, M., Burbey, I., Pérez-Quiñones, M.A.: Evaluation of a location-linked notes system. In: Proc. The 44th annual Southeast regional conference, ACM 494-499 (2006).

64. Cena, F., Console, L., Gena, C., Goy, A., Levi, G., Modeo, S., Torre, I.: Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide. AI Communications, vol. 19 369–384 (2006).
65. Setten, M.V., Pokraev, S., Koolwaaij, J. Context-aware recommendations in the mobile tourist application compass. In: W. Nejdl, and De, P.,Bra, editors, Adaptive Hypermedia 235-244 (2004).
66. Böhmer. M., Bauer, G., Krüger, A.: Exploring the Design Space of Context-aware Recommender Systems that Suggest Mobile Applications. In: Proc. CARS 2010, the 2nd Workshop on Context-Aware Recommender Systems (2010).
67. Jannach, D., Zanker, M., Felfernig, A., Friedrich, G.: Recommender Systems: An Introduction. Cambridge University Press, pp 289-298 (2010).
68. Baltrunas, L., Amatriain, X.: Towards Time-Dependant Recommendation based on Implicit Feedback. Workshop on Context-Aware Recommender Systems, CARS 2009, in: ACM RecSys, vol. 2009 (2009).
69. Baltrunas, L., Ricci, F.: Context-Dependent Items Generation in Collaborative Filtering. Workshop on Context-Aware Recommender Systems, CARS 2009, in: ACM RecSys, vol. 2009 (2009).
70. Domingues, M.A., Jorge, A.M., Soares, C.: Using Contextual Information as Virtual Items on Top-N Recommender Systems. Workshop on Context-Aware Recommender Systems, CARS 2009, in: ACM RecSys, vol. 2009 (2009).
71. Lombardi, S., Anand, S., Gorgoglione, M.: Context and Customer Behavior in Recommendation. Workshop on Context-Aware Recommender Systems, CARS 2009, in: ACM RecSys, vol. 2009 (2009).
72. Oku, K., Nakajima, S., Miyazaki, J., Uemura, S.: Context-Aware SVM for Context-Dependent Information Recommendation. In: IEEE International Conference on Mobile Data Management, vol. 0 109 (2006).
73. Yu, Z., Zhou, X., Zhang, D., Chin, C., Wang, X., Men, J.: Supporting Context-Aware Media Recommendations for Smart Phones. Pervasive Computing, IEEE, vol. 5, no. 3, pp. 68-75 (2006).
74. Bogers, T.: Movie Recommendation using Random Walks over the Contextual Graph. Proceedings of the 2nd Workshop on Context-Aware Recommender Systems. (2010).
75. Bourke, S., McCarthy, K., Smyth, B.: The Social Camera: Recommending Photo Composition Using Contextual Features. Proceedings of the 2nd Workshop on Context-Aware Recommender Systems. (2010).

76. Cantador, I., Castells, P.: Semantic Contextualisation in a News Recommender System. Workshop on Context-Aware Recommender Systems CARS 2009 in ACM Recsys. (2009).
77. Baltrunas, L., Kaminskas, M., Ricci, F., Rokach, L., Shapira, B., Luke, K.: Best Usage Context Prediction for Music Tracks. Proceedings of the 2nd Workshop on Context-Aware Recommender Systems. (2010).
78. Asoh, H., Motomura, Y., Ono, C.: An Analysis of Differences between Preferences in Real and Supposed Contexts. Proceedings of the 2nd Workshop on Context-Aware Recommender Systems. (2010).
79. Aciar, S.: Mining Context Information from Consumer's Reviews. Proceedings of the 2nd Workshop on Context-Aware Recommender Systems. (2010).
80. Panniello, U.; Tuzhilin, A.; Gorgoglione, M.; Palmisano, C., Pedone, A.: Experimental comparison of pre- vs. post-filtering approaches in context-aware recommender systems. Proceedings of the third ACM conference on Recommender systems, pp. 265–268 (2009).
81. Panniello, U. and Gorgoglione, M.: Does the recommendation task affect a CARS performance? RecSys10: Workshop on Context-Aware Recommender Systems (CARS-2010) (2010).
82. Rendle, S.: Factorization Machines with libFM. ACM Transactions on Intelligent Systems and Technology, vol. 3, no. 3, pp. 57:1–57:22 (2012) DOI 10.1145/2168752.2168771.
83. Rendle, S.: Factorization Machines. In 2010 IEEE International Conference on Data Mining. pp. 995–1000 (2010) DOI 10.1109/ICDM.2010.127.
84. Rendle, S., Gantner, Z., Freudenthaler, C., Schmidt-Thieme, L.: Fast Context-aware Recommendations with Factorization Machines. In Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, New York, NY, USA, pp. 635–644 (2011) DOI 10.1145/2009916.2010002.
85. Baltrunas, L., Ludwig, B., Ricci, F.: Matrix Factorization Techniques for Context Aware Recommendation. In Proceedings of the Fifth ACM Conference on Recommender Systems, New York, NY, USA, pp. 301–304 (2011) DOI 10.1145/2043932.2043988.
86. Hidasi, B., Tikk, D.: General factorization framework for context-aware recommendations. Data Min Knowl Disc, vol. 30, no. 2, pp. 342–371 (2016) DOI https://doi.org/10.1007/s10618-015-0417-y.

87. Zheng, Y., Mobasher, B., Burke, R.: CARSKit: A Java-Based Context-Aware Recommendation Engine. In 2015 IEEE International Conference on Data Mining Workshop (ICDMW). pp. 1668–1671 (2015).
88. Zheng, Y.: A User's Guide to CARSKit. CoRR, vol. abs/1511.03780 (2015) Online: http://arxiv.org/abs/1511.03780.
89. Konstan, J.A., Riedl, J.: Recommender systems: from algorithms to user experience. User Model User-Adap Inter 22: 101 (2012) DOI https://doi.org/10.1007/s11257-011-9112-x.
90. Pu, P., Chen, L., Hu, R.: Evaluating recommender systems from the user's perspective: survey of the state of the art. User Model User-Adap Inter 22: 317 (2012) DOI https://doi.org/10.1007/s11257-011-9115-7.
91. Tintarev, N., Masthoff, J.: Evaluating the effectiveness of explanations for recommender systems. User Model User-Adap Inter 22: 399 (2012) DOI https://doi.org/10.1007/s11257-011-9117-5.
92. Knijnenburg, B.P., Willemsen, M.C., Gantner, Z. Soncu, H., Newell, C.: Explaining the user experience of recommender systems. User Model User-Adap Inter 22: 441 (2012) DOI https://doi.org/10.1007/s11257-011-9118-4.
93. Bostandjiev, S., O'Donovan, J., Höllerer, T.: TasteWeights: a visual interactive hybrid recommender system. In Proceedings of the sixth ACM conference on Recommender systems - RecSys '12, Dublin, Ireland, pp. 35-42 (2012) ISBN: 978-1-4503-1270-7, DOI 10.1145/2365952.2365964.
94. Polatidis, N., Georgiadis, C.K.: Factors Influencing the Quality of the User Experience in Ubiquitous Recommender Systems. In: Streitz N., Markopoulos P. (eds) Distributed, Ambient, and Pervasive Interactions. DAPI 2014. Lecture Notes in Computer Science, vol 8530. Springer, Cham (2014).
95. Bottazzi, D., Montanari, R. Toninelli, A.: Context-Aware Middleware for Anytime, Anywhere Social Networks. IEEE Intelligent Systems, vol. 22, no. 5, pp. 23–32 (2007).
96. Eirinaki, M., Gao, J., Varlamis, I., Tserpes, K.: Recommender Systems for Large-Scale Social Networks: A review of challenges and solutions. In Future Generation Computer Systems, Volume 78, Part 1, pp. 413-418 (2018) ISSN 0167-739X, DOI https://doi.org/10.1016/j.future.2017.09.015.
97. Guy, I.: Social Recommender Systems. In: Ricci F., Rokach L., Shapira B., Kantor P. (eds) Recommender Systems Handbook. Springer, Boston, MA, pp. 511-543 (2015) DOI https://doi.org/10.1007/978-1-4899-7637-6_15.

98. Davidson, J., Livingston, B., Sampath, D., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., et al.: The YouTube Video Recommendation System. Proceedings of the fourth ACM conference on Recommender systems - RecSys '10 pp. 293–296 (2010) DOI 10.1145/1864708.1864770.
99. White, J., Schmidt, D.C., Czarnecki, K., Wienands, C., Lenz, G., Wuchner, E., Fiege, L.: Automated Model-Based Configuration of Enterprise Java Applications. 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), Annapolis, MD, 2007, pp. 301-301 (2007) DOI 10.1109/EDOC.2007.22.
100. Achilleos, A.P., Kapitsaki, G.M., Constantinou, E., Horn, G., Papadopoulos, G.A.: Business-Oriented Evaluation of the PaaSage Platform. In 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 322–326 (2015) DOI 10.1109/UCC.2015.51.
101. Núñez-Valdéz, E. R., Lovelle, J. M. C., Martínez, O. S., García-Díaz, V., de Pablos, P. O., Marín, C. E. M.: Implicit feedback techniques on recommender systems applied to electronic books. Computers in Human Behavior, Volume 28, Issue 4. (2012).
102. Peska, L.: Using the Context of User Feedback in Recommender Systems. In Proceedings of the 11th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, MEMICS 2016, pp. 1-12 (2016).
103. Yang, B., Lee, S., Park, S., Lee, S.: Exploiting Various Implicit Feedback for Collaborative Filtering. In Proceedings of the 21st International Conference Companion on World Wide Web, WWW '12 Companion, ACM, New York, NY, USA, pp. 639–640 (2012).
104. Yi, X., Hong, L., Zhong, E., Liu, N. N., Rajan, S.: Beyond Clicks: Dwell Time for Personalization. In: Proceedings of the 8th ACM Conference on Recommender Systems, RecSys '14, ACM, New York, NY, USA, pp. 113–120 (2014).
105. Jawaheer, G., Weller, P., Kostkova, P.: Modeling User Preferences in Recommender Systems: A Classification Framework for Explicit and Implicit User Feedback. ACM Transactions on Interactive Intelligent Systems vol. 4, Issue 2, Article 8, pp. 1-26 (2014) DOI http://dx.doi.org/10.1145/2512208.
106. Oard, D. W., Kim, J.: Implicit Feedback for Recommender Systems. In Proceedings of the AAAI workshop on recommender systems. Vol. 83. WoUongong (1998).
107. Nguyen, H.T., Almenningen, T., Havig M., Schistad, H., Kofod-Petersen, A., Langseth, H., Ramampiaro, H.: Learning to Rank for Personalised Fashion Recommender Systems via Implicit Feedback. In: Prasath R., O'Reilly P.,

Kathirvalavakumar T. (eds) Mining Intelligence and Knowledge Exploration. Lecture Notes in Computer Science, vol 8891. Springer, Cham (2014) DOI https://doi.org/10.1007/978-3-319-13817-6_6.

108. Hwangbo, H., Kim, Y.S., Cha, K. J.: Recommendation system development for fashion retail e-commerce. Electronic Commerce Research and Applications, Elsevier B.V. vol. 28, pp. 94-101 (2018) DOI https://doi.org/10.1016/j.elerap.2018.01.012.
109. Shen, Y., Lv, T., Chen, X., Wang. Y.: A collaborative filtering based social recommender system for E-commerce. (2016) 17. 9.1-9.6. 10.5013/IJSSST.a.17.22.09.
110. Zhou, L., Zhang, P., Zimmermann, H.D.: Social commerce research: An integrated view. Electronic commerce research and applications, Elsevier B.V., vol.12, No. 02, pp.61-68 (2013) DOI https://doi.org/10.1016/j.elerap.2013.02.003.
111. Han, H., Xu, H., Chen, H.: Social commerce: A systematic review and data synthesis. Electronic Commerce Research and Applications, Elsevier B.V., vol. 30, pp. 38-50, (2018) DOI https://doi.org/10.1016/j.elerap.2018.05.005.
112. Li, Y.M., Wu, C.T., Lai, C.Y.: A social recommender mechanism for e-commerce: Combining similarity, trust, and relationship. Decision Support Systems, vol. 55, No. 03, pp.740-752 (2013).
113. Hooda, R., Singh, K., Dhawan, S.: Social Commerce Hybrid Product Recommender. International Journal of Computer Applications, vol. 100, issue 12, pp. 43-49 (2014) DOI 10.5120/17581-8419.
114. Anindya, G., Li, B., Liu, S.: Mobile Advertising Using Customer Movement Patterns. Doctoral dissertation, Working paper, New York University, New York (2015).
115. Bell, D. R., Corsten, D., Knox, G.: From Point of Purchase to Path to Purchase: How Preshopping Factors Drive Unplanned Buying. Journal of Marketing 75(1):31–45 (2011).
116. Granbois, D. H.: Improving the Study of Customer In-Store Behavior. The Journal of Marketing 32(4):28–33 (1968).
117. Jeffrey, I.J., Winer, R.S., Ferraro. R.: The Interplay between Category Characteristics, Customer Characteristics and Customer Activities on In-Store Decision Making. Journal of Marketing 73:19–29 (2009).
118. Jie, C., Dong W., Canquan, L.: Recommendation system technologies of intelligent large-scale shopping mall. Proceedings of 2nd International Conference on Computer Science and Network Technology, Changchun, pp. 1058-1062 (2012).

119. Kawashima, H., Matsushita, T., Satake, S., Imai, M., Shinagawa, Y., Anzai, Y.: PORSCHE: A physical objects recommender system for cell phone users. In Proceedings of 2nd international workshop on personalized context modeling and management for UbiComp applications, California, USA (2006).
120. Bohner, F.: Adaptive user modelling and recommendation in constrained physical environments. In Proceedings of the 5th international conference on adaptive hypermedia and adaptive web-based systems, Springer-Verlag, Berlin, Heidelberg, pp. 378–383 (2008).
121. Pfeiffer, J., Pfeiffer, T., Meißner, M.: Towards attentive in-store recommender systems. Reshaping Society through Analytics, Collaboration, and Decision Support, v. 18 (2015).
122. Murray, K. B., Liang, J., & Häubl, G.: ACT 2.0: The next generation of assistive consumer technology research. Internet Research, 20 (3), 232–254 (2010).
123. Reischach, F.V., Michahelles, F., Schmidt, A.: The design space of ubiquitous product recommendation systems, in: Proc. The 8th International Conference on Mobile and Ubiquitous Multimedia 1-10 (2009).
124. Reischach, F.V., Michahelles, F.: Apriori: A ubiquitous product rating system. In: PERMID '08, Workshop on Pervasive Mobile Interaction Devices at Pervasive Conference (2008).
125. Reischach, F.V., Guinard, D., Michahelles, F., Fleisch, E.: A mobile product recommendation system interacting with tagged products. In: Proc. The 2009 IEEE International Conference on Pervasive Computing and Communications, 1-64 (2009).
126. Resatsch, F., Karpischek, S., Sandner, U., Hamacher, S.: Mobile sales assistant: NFC for retailers. In: Proc. Mobile HCI ’07, The 9th International Conference on Human Computer Interaction with Mobile Devices and Services (2007).
127. Resatsch, F., Sandner, U., Leimeister, J.M., Krcmar, H.: Do point of sale RFID-based information services make a difference? Analyzing consumer perceptions for designing smart product information services in retail business, Electronic Markets, 18(3) 216–231 (2008).
128. Takeuchi, Y., Sugimoto, M.: A user-adaptive city guide system with an unobtrusive navigation interface. Personal and Ubiquitous Computing, vol. 13, no. 2 119-132 (2007).
129. Adomavicius, G., Jannach, D.: Preface to the special issue on context-aware recommender systems. User Model User-Adap Inter, vol. 24, no. 1–2, pp. 1–5 (2014).

130. Rodríguez-Hernández, M. del C., Ilarri, S.: Pull-based Recommendations in Mobile Environments. Comput. Stand. Interfaces, vol. 44, no. C, pp. 185–204 (2016).
131. McDonald, DW.: Ubiquitous Recommendation Systems. Computer, vol. 36, no. 10, p. 111–, Oct. 2003 (2003).
132. Amyot, D., Farah, H., Roy, J.F.: Evaluation of Development Tools for Domain-Specific Modeling Languages. In System Analysis and Modeling: Language Profiles, pp. 183–197 (2006).
133. Sirius, http://www.eclipse.org/sirius/, Last Accessed: 25/11/2018.
134. Eclipse Modelling Framework (EMF), https://www.eclipse.org/modeling/emf/, Last Accessed: 25/11/2018.
135. ObeoDesigner 10.1, https://www.obeodesigner.com/en/download, Last Accessed: 25/11/2018.
136. lifehacker, "How Retail Stores Track You Using Your Smartphone (and How to Stop It)", https://lifehacker.com/how-retail-stores-track-you-using-your-smartphone-and-827512308, Last Accessed: 25/11/2018.
137. Fu, B., Kirchbuchner, F., von Wilmsdorff, J., Grosse-Puppendahl, T., Braun, A., Kuijper, A.: Indoor Localization Based on Passive Electric Field Sensing. In: Braun A., Wichert R., Maña A. (eds) Ambient Intelligence. AmI 2017. LNCS, vol 10217. Springer (2017).
138. estimote, https://estimote.com/products/, Last Accessed: 25/11/2018.
139. Huffingtonpost. "How Bluetooth Beacons Will Transform Retail in 2016". https://www.huffingtonpost.com/kenny-kline/how-bluetooth-beacons-wil_b_8982720.html, Last Accessed: 28/11/2018.
140. von Reischach, F., Michahelles, F., Guinard, D., Adelmann, R., Fleisch, E., & Schmidt, A.: An evaluation of product identification techniques for mobile phones. In Proceedings of the 12th IFIP TC 13 international conference on human-computer interaction: Part I (pp. 804–816). Uppsala, Sweden (2009).
141. Econsultancy.com. "Five retailers using NFC and RFID to enhance shopping: but do they work?" https://econsultancy.com/blog/65307-five-retailers-using-nfc-and-rfid-to-enhance-shopping-but-do-they-work, Last Accessed: 28/11/2018.
142. NFC-FORUM.org. "Major Retail Study: Mobile Consumers Prefer NFC Technology over Competing Alternatives". https://nfc-forum.org/newsroom/major-retail-study-mobile-consumers-prefer-nfc-technology-competing-alternatives/, Last Accessed: 28/11/2018.

143. WooCommerce, https://wordpress.org/plugins/woocommerce/installation/, Last Accessed: 25/11/2018.
144. DRUPAL COMMERCE, https://drupalcommerce.org/, Last Accessed: 25/11/2018.
145. Mohagheghi, P.: An approach for empirical evaluation of model-driven engineering in multiple dimensions. Proc. C2M:EEMDD 2010 workshop at ECMFA 2010- from Code Centric to Model Centric: Evaluating the Effectiveness of MDD, pp. 6–17 (2010).
146. Davis, F.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Quarterly 13(3), 318--339 (1989).
147. Heijden, H.: User acceptance of hedonic information systems. MIS Q. 28, 4, 695-704 (2004).
148. Schrepp, M.: "User Experience Questionnaire Handbook". DOI 10.13140/RG.2.1.2815.0245.
149. Laugwitz B., Held T., Schrepp M.: Construction and Evaluation of a User Experience Questionnaire. In: Holzinger A. (eds) HCI and Usability for Education and Work. USAB 2008. Lecture Notes in Computer Science, vol 5298. Springer, Berlin, Heidelberg (2008).
150. Al-Rubaie, M., Chang, J.M.: Privacy Preserving Machine Learning: Threats and Solutions. CoRR. vol. abs/1804.11238 (2018) Online: http://arxiv.org/abs/1804.11238.
151. Wang, C., Zheng, Y., Jiang, J., Ren, K.: Toward Privacy-Preserving Personalized Recommendation Services. Engineering, vol. 4, Issue 1, pp. 21-28 (2018) DOI https://doi.org/10.1016/j.eng.2018.02.005.
152. usability.gov Improving the User Experience. Eye Tracking. https://www.usability.gov/how-to-and-tools/methods/eye-tracking.html. Last Accessed: 20/12/2018.

Appendix **A**

Evaluating Learnability with Framework Revisits

A.1 Methodology

To better measure the *learnability* of the UbiCARS Framework's DSML and modelling editor, i.e. the quality related to usability that indicates how quickly the DSML and editor allow users to become familiar with them and enable them to make good use of their features and capabilities, we had participants of the evaluation revisiting the framework a second time through new evaluation sessions. In the revisiting evaluation sessions, the aim was to re-conduct the modelling part of the evaluation; however, participants this time did not have in their disposal a detailed description of tasks, only an outline of what they should achieve with the editor.

We have used the definition from [145] where learnability of a solution is defined as the *time* a user needs to achieve a specified proficiency level with a solution. However, instead of measuring learnability in terms of the time participants needed to become familiar with the editor, we have measured the level of familiarity and the extent that participants make good use of its features and capabilities, provided that they are using the framework only for the second time, i.e. after about *one hour* of previous usage.

For participants to take part in this evaluation session, the following requirements should apply:

1. The participants have used the framework exactly once during the first evaluation session.
2. A time period of more than 30 days is passed from the participant's first evaluation session with the framework.

The first requirement ensures that participants had not interacted with the framework prior to the framework revisit evaluation session, apart from their interaction with it during the first evaluation session. The second requirement ensures that enough time has passed from the participant's first evaluation session.

The framework revisit evaluation session included the following *metrics* for measuring familiarity and proficiency level with the editor:

- **Time taken to conduct the framework revisit session:** the measured time is to be compared with the participant's time during the modelling part of the first evaluation session.
- **Number of usability errors made:** we are interested in errors that have to do with the usability of the modelling editor. For example, selecting an incorrect tool from the toolbox for conducting a task.
- **Number of validity errors made:** These types of errors render the model non-valid, e.g. inserting an incorrect parameter value to an element. They determine also the quality of the results: whether the model is able to be successfully executed on system UI. During the first evaluation sessions, there were 22 subtasks that included validity errors that were avoided by participants, probably because the tasks description provided adequate information for them to be able to correctly execute those subtasks. In this evaluation session, we aim to observe the number of validity errors the participants will be able to avoid from the total number of possible validity errors.

Two developers had agreed to participate in the framework revisit evaluation session (in two separate sessions). As opposed to the first evaluation sessions, participants were not being observed by the author during the process. Participants were instructed to follow the guidelines to conduct the evaluation session to the maximum of their abilities and to stop whenever they think that their model is completed, or when they could not proceed with the process by themselves. The session was video recorded by means of screen capturing during the entire process, so that the session could be studied in detail afterwards. The evaluation session guidelines were the following:

"Use the MDD editor to create and work on a UbiCARS model. Try to design a full model without seeking assistance from the researcher present. The model should include the following elements: a Context-Aware Recommender System (CARS), a

Ubiquitous CARS (UbiCARS), an engine with an algorithm, elements to present and store recommendations, as well as the user-product interaction elements (Rating, Browsing History and Purchase History) with their database resources. In terms of the UbiCARS app, please add the Ubiquitous Technology used and the Staying Time as a user-product interaction element, along with its database resource. When you believe you have concluded your model, please proceed with the questionnaire by clicking NEXT".

A.2 Results

Table 14 summarises the results. The column "Days Between Sessions" mentions the number of days passed from the participant's first evaluation session until the framework revisit evaluation session.

Table 14: Results of Framework Revisit evaluation sessions.

		Time Taken (in Minutes)		Number of Errors	
Participant #	**Days Between Sessions**	**First Evaluation**	**Revisit Evaluation**	**Usability Errors**	**Validity Errors**
1	78	26	26	2	0
2	46	26	17	4	4

For the first participant, the evaluation took exactly the same time as for the first evaluation session. No validity errors were found (validity score was 100%), meaning that the outcome model of the session could be parsed by the system UI to configure the e-store with no changes needed. Video analysis showed that participant one had made two usability errors while trying to add elements on the canvas. In particular, she tried to add an element on the canvas that was already there. This happened twice. Other than that, her interaction with the editor was flawless.

The second participant made more usability errors and also four validity errors. She managed to finish earlier than the first participant, which may indicate spending less time in thinking about - or trying to remember about - the steps of the process. Regarding the validity errors, the second participant avoided 18 of the 22 possible validity errors, resulting in a validity score of 82%. The first error included adding a context element that

was not connected to a database resource, although this database resource was created. Moreover, two database resources had the same resource values, which cause a parsing error. The third and fourth errors included adding database resources for rating and purchase history that were not in the appropriate format to be configured on the Wordpress platform. Regarding usability errors, these included adding elements in an incorrect order (three times), and selecting an incorrect tool for linking two elements together.

Moreover, the second participant had managed to create a context element and correctly link it with the staying time element, something that was not included in the instructions; clearly, she had done it by memory from the first evaluation session. She had also remembered to create a database resource for that context element, which is quite interesting, although eventually she did not manage to correctly link it with the context element.

Learnability for the UbiCARS MDD Framework

To determine learnability for the UbiCARS MDD Framework, we first provide the total interaction time with the framework for the two participants during the first evaluation session (note that the numbers in column "Time Taken, First Evaluation" in Table 14 are times for the modelling part and not for the entire evaluation session): for the first participant the total interaction time is 56 minutes; for the second participant the total interaction time is 47 minutes. After the end of the revisit evaluation sessions, the total interaction time for participant one is 82 minutes (56+26) and for participant two is 64 minutes (47+17).

From the results above, we can state that learnability for the UbiCARS MDD Framework for the first participant is *82 minutes*, since she managed to achieve maximum results for metrics "Time taken to conduct the framework revisit session" and "Number of validity errors made", while for the "number of usability errors made" metric, the score is regarded high as these were very few.

In terms of the second participant, learnability cannot be determined from this evaluation session and is *estimated to be more than 64 minutes*. While achieving maximum results for metric "Time taken to conduct the framework revisit session", nevertheless, for the remaining two metrics, the results were good but not adequate. In particular, due to the validity errors made, the second participant will require more sessions with the framework to achieve an acceptable level of familiarity with it, thus eliminating validity errors. It is possible that, during a third session, validity errors would be avoided.

Responses to the Questionnaire

The questionnaire included four 7-Likert scale agreement questions specifically designed for the framework revisit evaluation session (again, 1 being strongly disagree and 7 being strongly agree).

Question: *I think I was able to complete the task at 100%.*

Participant one agrees (point 6 in scale), while participant two slightly agrees (point 5 in scale). It seems that participant one was more aware of her actions and the process she is following. Participant two had some doubts, which is also reflected through the errors made.

Question: *The task was difficult for me to accomplish.*

Participant one disagrees (point 2 in scale), participant two replied neutrally (point 4 in scale). In regard to participant one, who in the first evaluation session had strongly disagreed (point 1 in scale) in finding the evaluation tasks difficult at a rate of about 89% and furthermore had expressed disagreement (point 2 in scale) in the remaining 11%, this task was also not difficult to accomplish. Participant two, however, appears to have some trouble remembering the steps she had conducted during the first evaluation session. In the first evaluation session she had strongly disagreed in finding the evaluation tasks difficult at a rate of about 78% and disagreed in the remaining 22%. These statistics are similar to those of participant one for the first evaluation session.

Question: *Revisiting the UbiCARS Framework to develop a recommender system was easy for me.*

The response of participant one is in correspondence with her response to the previous question, as she agreed to the statement (point 6 in scale). The response of participant two however, has moved to the positive side of the scale. In particular, she replied to slightly agree to the statement (point 5 in scale).

Question: *Do you believe that using a Model-Driven Development approach via the UbiCARS Editor to develop Recommender Systems is a good idea?*

A variation of this question was also asked in the first evaluation questionnaire, where participants were asked whether they believe that using a model-driven development approach through the UbiCARS Framework (instead of editor) to develop RSs is a good idea. With this question, we seek to investigate whether participants had changed their minds after conducting the framework revisit evaluation session.

Both participants strongly agreed (point 7 in scale) in that using a model-driven development approach via the UbiCARS Editor to develop recommender systems is a good idea. To the question in the first evaluation questionnaire, participant one had responded agreement (point 6 in scale), while participant two had responded strong agreement. It is therefore valid to say that the two participants maintained their opinion in that using a model-driven development approach through the UbiCARS Framework (or editor) to develop RSs is a good idea.

Other Comments from Participants

The following comments from the participants were received through the questionnaire.

The first participant expressed the opinion that the framework revisit evaluation session was *"not so easy in comparison to the first evaluation, nevertheless it was easy enough, considering the final recommendation system to be produced out of this"*. By the sentence "considering the final recommendation system to be produced out of this", the participant meant that the process was easy, considering the effort one would need to make in order to develop a recommender system by other means. Moreover, she mentioned that *"some details needed some extra "thinking", but nothing more than that"*. This statement reflects that the participant needed to make some extra thinking in comparison to the first evaluation session where tasks where described in more detail.

Another remark of the first participant was that, it would be nice having available explanations relevant to the modelling process appearing when hovering on the question mark icon. The participant thought that this would make the process easier and the editor more user-friendly. Note that the question mark icon is not part of the modelling editor designed in this work; rather, it is part of the Eclipse modelling framework used to build the editor.

In terms of the second participant, she would like to be allowed to copy and paste, as well as to be able to auto-format the model to make it tidier. In fact, the latter functionality is provided by default within the Eclipse platform, and thus, it was inherited by our editor. The participant did not express this opinion while using the editor; therefore we were not aware of it to inform her accordingly.

Appendix **B**

Remote Evaluation

B.1 Methodology

Attempts were made to facilitate the remote evaluation of the UbiCARS Framework. In the remote evaluation session, participants were requested to use the UbiCARS Modelling Editor to create a Recommender System application model. Feeding the model to the system through the System UI was out of the scope of the remote evaluation for two reasons: firstly and primarily, to make the evaluation shorter in duration, and secondly due to the overhead of setting up a number of e-stores for many participants to be able to use simultaneously, or, alternatively, the overhead of scheduling evaluation sessions among many participants in the case where only one e-store was used.

Instead of the aforementioned process, we had prepared a video for participants that demonstrated in detail how a model that was designed via the modelling editor is used by the System UI to configure an existent e-store and mobile app. For practicality reasons, the video demonstrated the configuration of a Wordpress e-store, omitting other platforms, as the steps for their configuration are similar. The configuration included using appropriate plugins to retrieve user-product interaction data from the e-store, a mobile application to retrieve user staying time in front of products in the physical store and configuration and usage of recommendation engines.

To enable the remote evaluation, we had compressed and uploaded the UbiCARS modelling editor online, as well as prepared detailed guidelines for participants to use in order to conduct the evaluation. The guidelines included written instructions on downloading the Eclipse framework with the UbiCARS modelling editor and setting it up (5-minute process).

The evaluation included the same methodology, tasks and tasks description as the modelling part of the first evaluation (tasks 1-8, see Section 6.2.1). Task 14 was also excluded since it required prior usage of the system UI. Each task description in the questionnaire was followed by the questions *"Please record your time: I started at..."* and *"Please record your time: I stopped at..."*, as was also the case in the modelling part of the first evaluation. Moreover, after completing each task, participants were asked to express their agreement with the statement *"I was able to understand the task"* and *"the task was difficult for me to accomplish"* by using a 7-point Likert scale (1 being strongly disagree and 7 being strongly agree).

After completing all tasks and the evaluation questionnaire, participants were instructed (through the guidelines document) to send to the author via email the model they had designed as outcome of this evaluation session. They were also urged to watch the video mentioned earlier in this section about how a model similar to the one they had designed is used by the framework to configure an existent e-store. Participants were made to understand that, in case their model was according to the Tasks specifications (as described in the survey questionnaire), the system would be able to configure the e-store and provide recommendations to potential customers. On the contrary, in case the model was not according to specifications, the system would not be able to proceed with configuration. Participants were told that they would be contacted by the author to be informed about the outcomes of system usage with their model. It was made clear to all participants that any kind of scoring on any of the tasks they were instructed to complete was not kept.

B.2 Results

Three participants completed the remote evaluation process. Mean task duration for each task was very close to mean task duration values for all participants during the first evaluation, as presented in Section 6.2.3 and in Figure 24. Thus, these results are not presented here.

A total of five participants have responded the full UEQ questionnaire, as the UEQ was also completed by the two participants during the framework revisit evaluation sessions. Although few participants have completed the UEQ questionnaire in order to perform a statistical analysis with significant results, we present the results below as an

indication of the framework's performance. Figure 59 depicts the participants' responses to the question: *"My interaction with the UbiCARS Framework was"*.

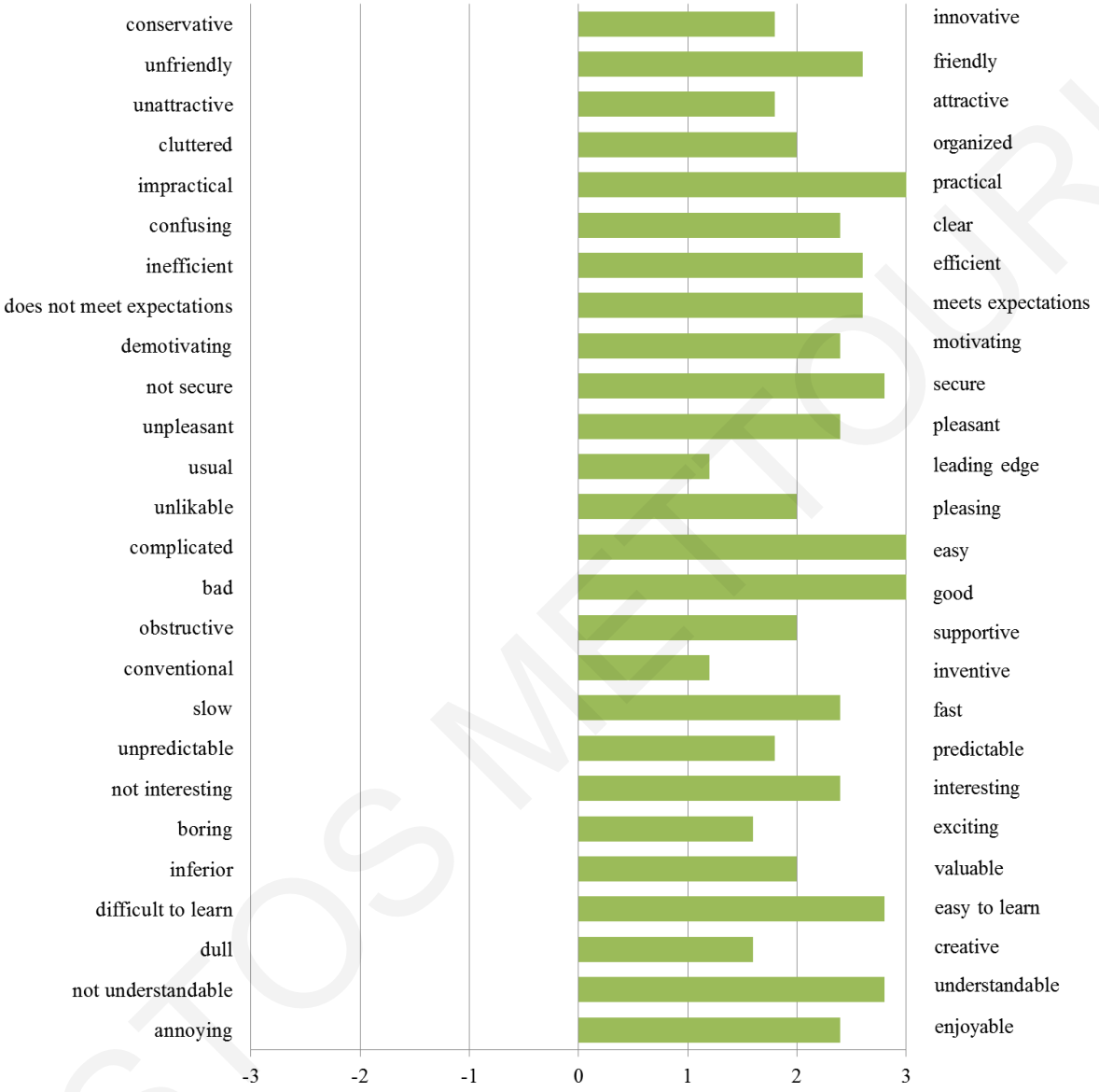

Figure 59: UEQ questionnaire results for a total of five participants.

As stated in "UEQ questionnaire" sub-section of Section 6.2.1, a number of the 26 items of UEQ are in common with questions included in the TAM questionnaire used in the first evaluation. To the point where UEQ items presented above are common with items in the TAM, we note that these results agree between them and indicate a very positive opinion of users concerning their experience with the framework. Table 15 lists these common items and their mean values.

Table 15: UEQ and TAM common items and mean values.

UEQ Item and Range	**UEQ Score**	**TAM Item Range: 1 strongly disagree; 7 strongly agree**	**TAM Score**
Efficient: +3 Inefficient: -3	2.6	Using the UbiCARS Framework would improve my performance in developing RSs	6.6
Easy to learn: +3 Difficult to learn: -3	2.8	Learning to operate the UbiCARS Framework would be easy for me.	6.3
Clear: +3 Confusing: -3	2.4	My interaction with the UbiCARS Framework would be clear and understandable.	6.3
Easy: +3 Complicated: -3	3	I would find the UbiCARS Framework easy to use	6.5

Literature related to the UEQ questionnaire [148, 149] categorises the 26 UEQ items under six UEQ scales, as follows.

- Attractiveness: Investigates whether users like or dislike the product.
- Perspicuity: Investigates whether it is easy to get familiar with the product and learn how to use it.
- Efficiency: Investigates whether users are able to solve their tasks without unnecessary effort.
- Dependability: Investigates whether users feel being in control of the interaction.
- Stimulation: Investigates whether it is exciting and motivating to use the product.
- Novelty: Investigates whether the product is innovative, creative and interesting.

Figure 60 shows the results per UEQ scale for the five participants. In [148], the author presents the standard interpretation of the scale in the figure as follows: the range of the scale is between -3 (horribly bad) and +3 (extremely good). Values between -0.8 and 0.8 represent a neural evaluation of the corresponding scale, values > 0,8 represent a positive evaluation and values < -0,8 represent a negative evaluation. The error bars in Figure 60 represent the 95% confidence intervals for the scale mean [148, 149] and they can be regarded as an indication of how accurate our measurement is.

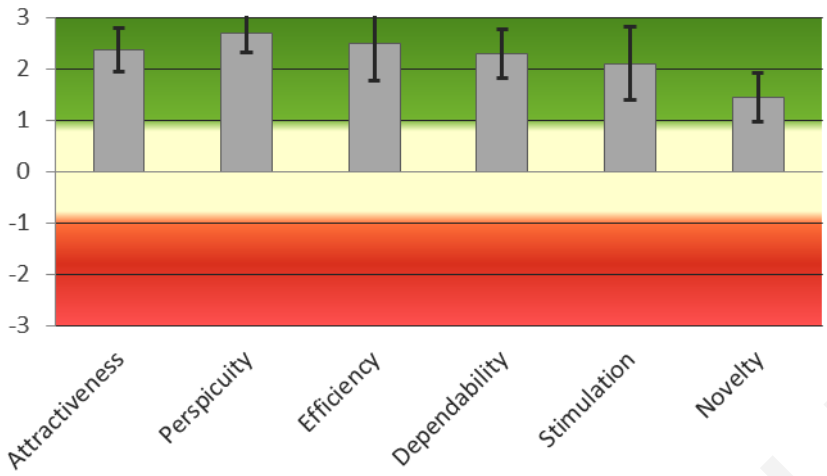

Figure 60: Results per UEQ Scale for five participants.

From the results we have with five participants, we can report an extremely positive evaluation of the UbiCARS framework in the UEQ scales. As stated above, since the number of participants conducting the full UEQ questionnaire is small, these results are only indicative of the framework's potential.