SN Computer Science (2025) 6:370
https://doi.org/10.1007/s42979-025-03896-4

ORIGINAL RESEARCH

An MDD Framework Towards the Automated Development of Ubiquitous Context-Aware Recommender Systems for Commerce

Christos Mettouris[1] · Achilleas Achilleos[2] · Georgia Kapitsaki[1] · George A. Papadopoulos[1]

Received: 24 April 2023 / Accepted: 19 March 2025

Abstract
While the benefits of product recommender systems (RS) are prominent, due to the complexity of recommendation algorithms and data models, it is difficult for businesses to deploy such systems on their e-stores. Related works that tackle RS development complexity do not entirely abstract the technical details from developers; thus, margins for improvement exist. Moreover, these works do not offer solutions that eliminate the need to write code by developers. These are the motivations for the Ubiquitous Context-Aware Recommender Systems (UbiCARS) Framework proposed in the current work. UbiCARS utilize user feedback acquisition techniques from both e-stores and physical stores to offer recommendations. The framework aims to reduce development complexity, abstract technical details and expedite the development of UbiCARS (facilitating both e-stores and physical stores) by non-RS experts. This is achieved through a Model-Driven Development methodology, that uses a model-based configuration process where models of recommender systems drive the dynamic configuration of UbiCARS on e-stores. The framework was evaluated with developers and experts via the survey method. The evaluation results show the framework's potential.

Keywords Recommender systems for commerce · Ubiquitous product recommendations · Model-driven development · Domain specific modelling language · UbiCARS

Introduction

Recommender systems (RS) discover knowledge about users and based on this knowledge, offer them personalised recommendations. According to literature, "un-contextual" RS refer to systems that utilize little or no contextual information for recommendation computation, while Context-Aware Recommender Systems (CARS) are designed to use more contextual parameters to increase recommendations' accuracy [1]. The term UbiCARS (Ubiquitous CARS) firstly introduced in [2] by the authors, refers to *CARS that are able to acquire and utilize user feedback on items not only from their online behaviour (e.g., an e-store), but from their behaviour in ubiquitous (physical) environments as well (e.g., a physical store)*. In the literature it has been shown that product recommendations to customers boost sales [1, 3], increase customers' satisfaction [1] and improve users' experience [4]. Thus, both customers and e-stores benefit from RS [3].

Whilst e-commerce sees an exponential growth, reports during the past years have shown that physical (traditional) commerce is still more popular than e-commerce [5, 6]. 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 [7]. Information systems, and in particular RS, are important in understanding what customers prefer, so where such systems are not utilized, customer needs and demands cannot be met immediately, risking a reduction on their shopping interest [8].

In e-commerce settings, RS track user behaviour online by acquiring user feedback data on products on the e-store (e.g., users' ratings on products and users' purchase history) to compute personalised product recommendations to users. In the ubiquitous, physical (brick-and-mortar) store scenario though, different methods are used to determine users' preferences on products, e.g., sensing the staying time

✉ Christos Mettouris
mettouris.g.christos@ucy.ac.cy

[1] Department of Computer Science, University of Cyprus, 2109 Nicosia, Cyprus

[2] Frederick University, 7 Y. Frederickou Str., 1036 Nicosia, Cyprus

Published online: 10 April 2025

of customers in various product areas in the store or sensing the shopping path of customers within the store [5, 9]. The aim is to recommend to users products or brand stores to visit, as well as to offer them product ratings and reviews.

A key question on which we focus in this paper, is: *how can businesses deploy RS on their e-stores and physical stores in an easy way? How can e-store experts and developers that are not RS experts deploy RS?* A potential solution is by using open-source RS frameworks such as Easyrec,[1] Lenskit,[2] Librec[3] and MyMediaLite.[4] Such frameworks could potentially be used by retail businesses as recommendation engines towards building RS for their e-stores; however, it is quite difficult for e-store developers and software engineers that are *not experts* in recommender systems to achieve such a task [10, 11]. Such frameworks do not abstract the technical details regarding the inclusion of recommendations to e-stores, which requires developers working on code level to accomplish tasks such as acquiring user feedback data, building data models, and developing software to retrieve the recommendations after their computation by the recommendation engine and display them on the e-store.

On another dimension, recent research on RS suggests that highly complex Machine Learning (ML) algorithms produce the most accurate recommendation results; however, due to their complexity, it is difficult for non-RS experts to use them in their applications [12]. In fact, even researchers face difficulties in tracking how ML algorithms are used in RS [12].

Another solution is outsourcing the process. Several RS expert companies exist that offer proprietary RS as commercial solutions (Google Cloud Machine Learning,[5] SLI Systems Recommender,[6] Azure Machine Learning Studio,[7] Amazon Machine Learning,[8] Yusp[9]). These RS experts deploy their own RS on their clients' e-stores with the aim to increase sales [13]. The advantage of such a solution for client businesses is that the experts manage by themselves the recommendation algorithms and the data needed for the recommendation process (content and context data, users' profile information, users' behaviour history, etc.), while the clients are only responsible for displaying the recommendations under experts' guidance and supervision. At the same time, disadvantages include the added cost, and that clients are not really motivated in investing in the recommendation algorithms per se. Another disadvantage is that the deployed RS cannot be used for further exploration or extension by clients themselves, as it is difficult to determine the algorithms that the experts use, how they use them, whether they combine more than one, etc. Hiring RS expert companies may not be feasible for all businesses, especially smaller ones that would like to offer recommendations to their users in an economical way, and, at the same time, keep control of the recommendation algorithms used, as well as not being dependant of vendors. For these businesses, the best solution would be finding a way to deal with RS development complexity in order to develop their own RS.

A few works in the literature offer methods to tackle RS development complexity [10, 11, 14, 15]; but none of these works manages to adequately abstract the technical details of data acquisition, data model construction, and software development for computing and displaying the recommendations on e-stores. Furthermore, the State-of-the-Art works do not succeed in eliminating or reducing the need to write code by developers, so that RS for commerce can be designed, developed and deployed expeditiously even by non-RS experts.

Having as motivation the above, we propose the UbiCARS Model-Driven Development (MDD) Framework that aims to reduce complexity, abstract the technical details, and expedite the design, development, and deployment of UbiCARS in ubiquitous commerce environments (physical stores) and electronic commerce environments (e-stores) by developers with no expertise and knowledge on RS.

Our research defines the following research questions:

RQ1 Does the UbiCARS MDD Framework reduce the *development time* (expedite the development) of 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 Ubiquitous Context-Aware Recommender Systems for commerce for developers that are non-experts in RS by reducing the lines of code and database queries developers need to write?

In an attempt to minimize the need to write code by the developers, our approach does not use code generation techniques and model-to-code transformations that would require from developers to work on code level in order to use and extend the generated artefacts. Instead, our approach proposes using models designed by developers to drive model-based automatic configurations [16, 17] directly on e-store platforms. The configurations concern the integration of UbiCARS (that realize user feedback techniques on e-stores and physical stores) with e-stores. In the proposed methodology, scenarios can be identified where the need for developers to write code is eliminated. In other scenarios

[1] sourceforge.net/projects/easyrec/.
[2] lenskit.org/.
[3] github.com/guoguibing/librec.
[4] mymedialite.net/.
[5] cloud.google.com/ml-engine.
[6] www.sli-systems.com/.
[7] azure.microsoft.com/en-us/services/machine-learning-studio/.
[8] aws.amazon.com/machine-learning/.
[9] www.leaseweb.com/en/customers/yusp.

however, a set of software modules that execute a particular task must be developed.

The novelty of the work is the UbiCARS framework that includes a graphical DSML (Domain Specific Modelling Language) for UbiCARS and a configuration component, which facilitate respectively a model-based design and dynamic configuration of UbiCARS for physical and online commerce. The UbiCARS DSML is fundamentally a graphical language with a corresponding modelling editor where developers can design UbiCARS and deploy them for use in physical and online commerce. Our methodology does not conduct modelling of recommendation algorithms' logic, as this would make the DSML algorithm specific and would significantly increase its complexity, while too complex recommendation algorithms would be impossible to model. Rather, the framework makes use of existing recommendation algorithms and engines by exposing them to the modelling layer and, in this way, making them available to the framework. To the best of our knowledge, an MDD framework for UbiCARS that provides a DSML for the commerce domain does not yet exist. While in our previous work [18] an initial version of the framework has been presented and tested through experimentation in the premises of a research laboratory, in this paper we fully elaborate on an updated, extended version of the framework that considers the initial feedback received. Furthermore, we describe two evaluations of the framework, one with developers and one with experts. The paper also conducts a comparison of the modelling against the coding methodologies in terms of development effort needed to implement a RS on an e-store. Finally, we examine and discuss our research questions.

In the remaining of this section, we provide an example of use of the UbiCARS framework, aiming to show the benefits for both the customer and the store.

UbiCARS Example of Use

Assume the "SmarTech electronics store that has an e-commerce website and a number of physical showrooms with electronic products. The UbiCARS framework requires a mobile application (UbiCARS app, see "UbiCARS Methodology" section) for usage in the physical showrooms, as well as a server side system (a CARS, see "UbiCARS Methodology" section) functioning on the e-commerce website. Thomas, as a frequent visitor of the SmarTech e-store, has already bought and rated several products online. Thomas fancies the recommendations of products he receives on the e-store since these are personalised and most of the times suit his preferences. Thomas is currently visiting the SmarTech physical store showroom, which only displays a small subset of the available products offered on the e-store due to limited physical space. A laptop on a shelf has caught his attention, so he approaches it to read the specifications on the small tag. Thomas switches on the SmarTech UbiCARS application on his mobile device. The application sends to the server information based on which the latter identifies the product in front of Thomas and calculates his *staying time in front of the product*. Having available this new behavioural data about Thomas, the CARS computes new product recommendations for Thomas that he can access through the UbiCARS app, similar to those he receives while browsing the SmarTech e-store. The recommended products can be found in the showroom (Thomas can immediately check them out), or online through the SmarTech e-store. Thomas finds it informative and entertaining to be able to receive recommendations while in the store.

After Thomas repeatedly visits both the SmarTech physical store and e-store, he is tracked by the system in terms of the following: product ratings (explicit feedback), browsing history and online purchase history (implicit feedback) from the e-store, and *Staying time* in seconds (implicit feedback) from the physical store. From each of the abovementioned, the system is able to compile datasets and use them as input in the recommendation algorithm. Listings 1 and 2 depict examples of such datasets. Thus, in addition to the e-store related datasets, the CARS system is now enabled to also use the staying time dataset to produce recommendations (related with DP5, see "Framework Design Principles" section). The availability of the additional dataset contributes to a potential reduction of the cold start problem. The cold start problem is an inherent problem of RS which dictates that, for RS to be able to produce meaningful recommendations, user interaction with items first needs to take place. The additional dataset makes more user-product interaction data available for the recommendation algorithm to use. Moreover, the additional dataset suggests a reduction of uncertainties; thus, the recommendation algorithms will probably perform better. In this sense, as additional data regarding Thomas' interaction with items is used by the RS, product recommendations for Thomas during his following visits to the SmarTech e-store or the physical stores will potentially be more accurate.

In terms of recommendation availability, since each implicit and explicit 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 potentially increases recommendation availability. In the example of use, recommendations for Thomas can be computed by using any of the following datasets: ratings, purchase history, browsing history and staying time.

The potential improvement of recommendations' accuracy and availability mentioned above has not been currently proven, and therefore is not included in the contributions of this paper. This is left as future work.

Listing 1 Ratings Dataset

UserID	Itemid	Rating	Day	Time
1	15	4	Weekday	Evening
5	24	5	Weekend	Noon

Listing 2 Staying Time Dataset

UserID	Itemid	StayedIn-FrontOf	Day	Time
1	25	231	Weekday	Afternoon
3	29	38	Weekend	Morning

Section "Background and Related Work" provides the background on important concepts of this research, i.e., on RS, CARS and UbiCARS, user feedback acquisition techniques met in e-commerce and physical commerce, as well as background on MDD. Section "Background and Related Work" continues by describing related work on systems in the literature that attempt to address development complexity of RS. Section "The UbiCARS MDD Framework" describes the UbiCARS MDD Framework, and in "Design Demonstration" section, the UbiCARS Demonstrator is presented. Section "Evaluation" discusses the evaluation of the framework and results. The paper completes with discussion of results in "Discussion" section and conclusions and future work in "Conclusions and Future Work" section.

Background and Related Work

After introducing the concepts based on which we have formulated the contributions of the paper, this section discusses related work.

RS, CARS and UbiCARS

RS have attracted the research community's interest for the past twenty years. Many techniques have been proposed, as well as many extensions and improvements. The most well-known recommendation approaches are the Collaborative Filtering (CF), the Content-based filtering and Hybrid recommendation techniques. The most widely used approach, CF, 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 (e.g., ratings) to learn a model for the user that is then used for computing recommendations [19].

Recently, the research community realised that RS have only been using a part of the available information for producing recommendations. The problem was that traditional RS do not utilise context information. Instead, they focus on two dimensions: the user and the items (also called two-dimensional RS), excluding other contextual data that could be used in the recommendation process. 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]. From the many methods of using context for producing recommendations, Adomavicius et al. [19] discuss that *contextual modelling* is the most effective and accurate method.

Contextual modelling refers to the Multidimensional Contextual Modelling approach which incorporates the multidimensional context in the recommendation process (as opposed to other techniques described in [19]). According to Adomavicius et al., the contextual modelling approach promotes truly multidimensional recommendation methods, which essentially represent ML predictive models that incorporate contextual information in addition to the user and item data. The input data of those models include additional dimensions besides users and items. These are CF model-based methods, and specifically latent factor models (ML approaches) that attempt to estimate ratings by characterising both items and users in latent factors inferred from the ratings patterns [20, 21]. 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, also allowing the incorporation of additional information besides explicit user feedback, such as implicit feedback, temporal effects, and confidence levels. More on ML on RSs can be found in [22], a recent comprehensive review on RSs in [23] and on CARS in [24].

Ubiquitous RS on the other hand facilitate users on-location by providing them with personalized recommendations of items in the proximity via mobile devices [2]. Ubiquitous RS use sophisticated recommendation methods to compute the recommendations for their users. Intelligent tourist guides, navigation aids, and shopping recommenders that recommend based upon user activities and behaviour patterns are examples of such systems. Furthermore, Ubiquitous CARS (UbiCARS) are ubiquitous RS that also utilize the context in the recommendation process in a similar manner as CARS do. UbiCARS use contextual modelling to incorporate contextual information in the recommendation process, only that their context also consists of ubiquitous information such as the user's location, items in the

proximity, etc. In [2], we provide a formal definition for ubiquitous RS and UbiCARS.

Challenges

Challenges related to UbiCARS can be categorized in those related to ubiquitous computing and those related to RS [2].

Challenges related to ubiquitous computing:

- 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.
- The mobile device must be able to track user intentions for the system to understand what actions could help the user accomplish his/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: he/she is having in front of him/her or holding at any given time.
- Users operate small devices that need attention.
- Devices may not be transparent since they operate on-field and by considering various contextual parameters: not transparent devices may provoke feelings of frustration to users.
- Context sensing: appropriate technologies and sensors must be utilized 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.
- Appropriate usage of all available context. For example, a UbiCARS for products will consider among other, the user preferences (what the user likes) in combination with environmental context (the current day/time and location) to provide personalized recommendations.
- Privacy concerns regarding location-awareness. The user must trust the system to agree in providing sensitive information such as location.

Challenges related to recommender systems:

- Building appropriate user models to effectively store and use user preferences and the context.
- The "New user" and "New item" problems are two of the most important ones since UbiCARS use CF.

RS for E-commerce

CF relies only on users' behaviour and this characteristic makes it the most suitable method for e-commerce, since explicit profiles for users and/or products are not needed [4]. However, users still need to act on items for the RS to be able to produce recommendations.

In e-commerce, RS use *explicit user feedback* data *on products* (e.g., users' ratings on products) to model users' preferences and, based on the user model, provide personalised product recommendations to users [4, 9]. Where explicit feedback is not available, *implicit user feedback data on products* can be used. Implicit feedback is acquired by *tracking users' behaviour*, e.g., users' purchase history (transaction data), clickstream data,[10] click-through rate[11] (CTR) and browsing history on product webpages [25–27]. Implicit techniques have been used by RS for products on e-stores, as well as movies, music, scientific papers, and other. Explicit techniques require users' cognitive effort, which may act as disincentive, leading thus to data sparsity [28]. Moreover, they interrupt users' task. Problems in using implicit techniques are that users' actions denote what users like but not necessarily what they do not like; and that while users' behaviour can be tracked, true users' preferences and motives can only be guessed [4] (e.g., a purchased item could be a gift, or the user could eventually be disappointed with the product).

Works in the literature that utilize implicit and explicit user feedback acquisition techniques in e-commerce are discussed next.

Yang et al. [26] propose a music RS for which positive user behaviour on songs included explicit and implicit song play, playing a full song, search for a song to add in the playlist and register a new song. Negative user behaviour included explicit song skip by the user, implicit skip (when changing the song), and delete a song from the playlist. In [27], users' dwell time on Yahoo home page items was used to measure the likelihood a page item is relevant to a user. Dwell time on page objects as implicit user feedback data was also utilized by Peska [25]. Sulikowski and Zdziebko [29, 30] utilize the times the cursor is in recommendation areas of their e-commerce website, their physical size, and the users' product interest. In [31], the authors have used gaze tracking solutions, as well as counting the number of times the user moved over or browsed a given element of the website to learn users' preferences. In [32], a CF RS is proposed that uses the social-economic indicators of users who have bought or evaluated an item as implicit user feedback to mitigate the cold start problem. In [33], the usage of AI in e-commerce RS is reviewed, reporting content-based scoring, collaborative filtering, deep learning, and virtual assistants. According to the authors, the benefits of using AI in RS include improved decision-making, reduced shopping duration and effort, increased sales, and overcoming data sparsity and cold-start issues. In [34], an extension of online

[10] Data about which webpages users visit, e.g., product webpages.

[11] The ratio of users who click on a link to the number of total users who view the webpage.

learning methods for re-ranking modelling in e-commerce RS is proposed. In RS, a re-ranking model re-ranks the item candidates by considering additional criteria or constraints. According to the authors, the proposed model can effectively model online learning without waiting for real user feedback, which may be delayed (e.g., item purchases).

A work similar to ours was conducted by Hwangbo et al. [35], who proposed a recommendation method that combines users' click history on products on the e-store of a fashion company,[12] with users' product purchase history from the same company's physical store. The aim was to reflect the online and physical preferences of customers respectively. In their setting, there was no linkage between the customers of the e-store and those of the physical store. 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 at the physical store, where these two are linked together via one user account in the system. Moreover, in [35], users' purchases on products were being tracked in the physical store, whereas our method proposes using the "*Staying Time in front of a product"* within the store (see "UbiCARS Methodology" section). The authors conducted an experimental evaluation with real users utilizing online and physical store data from users. The results indicate that utilizing implicit and explicit user feedback acquisition techniques can improve recommendations' accuracy.

In addition to implicit and explicit techniques, an alternative recommendation approach analyses users' textual reviews on e-commerce (and other) online platforms. As reviews provide insights into users' fine-grained preferences and item features, analysing these reviews contributes to enhancing the performance and interpretability of personalized recommendations [36]. Review-based recommender systems are considered to be a significant sub-field in the recommendation domain. More on the topic can be found in [36]

RS for Physical Stores

Many works in the literature have utilized e-commerce RS techniques to provide recommendations to their customers at physical store locations, such as a shopping mall, a theme park, a restaurant, or a grocery store. While most of these works based their recommendations mostly on customers' purchase history, So and Yada [9] proposed providing recommendations of shops to visit based on users' in-store shopping path. CF was used based on customers' *"staying time"* in each area. Other works [37] have shown that staying time in an area is related to the level of interest of the user in the item positioned in that selling area. It is also possible that users staying in a selling area for an extensive amount of time make spontaneous purchases of products in that area [9, 38–40]. In the following we describe how works have used ubiquitous technologies to understand user preferences for RS in physical stores.

Fang et al. [5] proposed a mobile store RS that relies on a novel indoor mobile positioning approach that uses mobile phone signals, achieving store level accuracy. Jie et al. [41] described a shop RS to be used in a shopping mall. Customers' location detection is achieved via the usage of appropriate RFID devices. Walter et al. [3] utilize a personal shopping assistant (PSA) that recommends products to customers based on the products within the customers' cart, the customers' location, and the customers' purchase history. In [42], a smartphone-based augmented reality shopping assistant application is proposed, which uses augmented reality to display tailor-made offers, product comparison and recommendations, utilizing explainable artificial intelligence. The RS has access to users' personal information such as social media and historical purchase data. Anchored around the product of interest, the application displayed recommendations, offers, and comparison of items [42], e.g., it could identify the product with which the customer was interacting and provide tailored content. Kawashima et al. [43] proposed a shopping RS that assigns user preference scores based on how near a user is to the product, whether a user picks up a product or whether a user scans a product via a RFID reader. Pfeiffer et al. [44] presented a system that uses eye-tracking to understand when users are standing in front of shelves and use it as an implicit user feedback in a non-intrusive way. Reischach et al. [45, 46] facilitated users while shopping at the physical store, not only through personalized product recommendations, but also through the provision of users' comments, suggestions, and ratings on products they are about to buy. The authors in [8] identify customers' preferences by scanning RFID tags of products using smartphones with RFID readers and providing in real time information related to the products and product recommendations. Mora et al. [47] have used a mixed reality (MR) head-mounted display (HMD) to recommend products to customers while shopping via a mobile app. The system uses customers' behaviour in-store such as 3D position, head orientation, eye gaze, gesture recognition and voice commands. Other works for the interested reader are: [48, 49].

From the above discussion it is shown that, in terms of facilitating shopping in physical stores, ubiquitous (mobile and wireless) technology can be used to acquire user feedback in order to elicit user preferences, while RS technology and methods can offer personalised content to users. Table 1

[12] Company statistics: 5 million members; 40 000 products sold yearly online; 1.5 million clicks and 10 000 online transactions are available per month; 1300 physical stores in Korea; 20,000 products sold yearly.

Table 1 Usage of ubiquitous technologies in works that facilitate physical shopping

References	Technology	Methodology
[9]	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
[3]	a	Uses a Personal Shopping Assistant for tracking the products in the user's shopping cart and user's location. All products must be tagged with RFID chips. Able to identify products the user is currently looking at or has added in their shopping card
[5]	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)
[43]	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
[41]	a	RS that recommends shops in a mall: to detect customers' location, RFID devices and related infrastructure have been deployed
[44]	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
[46]	a	Users scan RFID tagged products by using a mobile device to receive/produce recommendations
[8]	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
[47]	f	Used a MR HMD to recommend products to customers while shopping via a mobile app

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

summarises the ubiquitous technologies used in the above works and how they were used.

Model-Driven Development

Model-Driven Development aims at the abstract representation of application domains and the use of mapping tools to transform the abstract model into a working application (model-to-code transformation). Models through automated transformations or interpretations are converted into applications, eliminating or minimising the need to write code. In MDD, an important challenge is to include all required structures to describe the model, while keeping the element of abstraction, that is more challenging in specific areas, such as cross-platform applications [50]. Many works in the literature have used MDD in various domains, such as Internet of Things applications [51].

Using MDD for Automated Configuration of Applications

MDD has been used for automated configuration of systems and applications in other domains than commerce. In [16], White et al. proposed an MDD approach for automated enterprise application configuration. The authors argue that enterprise applications are hard to configure, and thus techniques for their automated configuration are needed. The proposed 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.

MDD has been applied on cloud computing as well. In [17], Achilleos et al. discussed that the diversity of cloud infrastructures, platforms, and tools that is offered to businesses creates challenges, such as hinders interoperability, promotes vendor lock-in, and prevents businesses from making informed and optimal decisions when transitioning to the cloud [17]. There might be a need for businesses to utilise many different cloud providers, e.g., in the case where a hybrid cloud deployment is desirable, where the application 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, an open-source integrated platform has been developed that 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.

Related Work

Few works in the literature propose using modelling and other software engineering techniques for aiding the development of RS by tackling RS development complexity.

This section describes the State-of-the-Art in the literature regarding works that focus on how RS development could be simplified and expedited.

Hussein et al. [10] support that research into RS has been concentrated on the development, optimization and evaluation of recommendation algorithms, giving less attention on the support and facilitation of the development of RS from a software engineering and architectural perspective. They propose a recommendation framework named Hybreed for assisting developers build CARS and hybrid RS. Hybreed supports the development of RS by applying a decentralized, service-oriented approach. It utilizes a dynamic contextualization process to provide an abstraction for developers, reduce programming effort, and increase comprehensibility and usability when developing complex RS. Hybreed requires developers to write code. To develop an application using Hybreed, developers need to utilize Hybreed's View Java interface to create customized views and instantiate a Kernel object by utilizing these views [10].

Rojas et al. [14] research on how to assist web developers in dealing with recommendation algorithm complexity when attempting to use such algorithms in their web applications. The authors emphasize the absence of model-driven methodologies for determining the specifications of RS algorithms and interface characteristics. They define a UML-based modelling approach and model an un-contextual Item-to-Item CF recommendation approach for including recommendations in e-commerce applications. Developers may interact with their system via object creation procedures using the framework's classes and execution of appropriate functions, e.g., create, delete, update, and retrieve. The authors state that the development of their case study of an online travel agency took considerably less time in contrast to the development time for an alternative content-based RS. About modelling the recommendation algorithm itself, we could argue that flexibility is reduced as the model becomes algorithm-specific making it difficult to use other algorithms, the complexity of the modelling process increases since hard to comprehend recommendation specific parameters become available for usage by designers, and 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.

Inzunza et al. [11] propose a user modelling framework for CARS named UM4RS that serves as a tool for building data models for CARS. The framework aims to increase the productivity of developers while building the data model (user, item, and context) for CARS. A model schema for CARS and a UML class description are offered. Developers can use the modelling framework to create objects from its classes and perform 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 development framework for CARS for mobile users is proposed in [15]. The framework proposes a pull-based architecture for pull-based mobile recommendations, i.e., recommendations are provided as a response to a user generated query.

Our work differs from related work in the following six important aspects:

- It defines a novel Domain Specific Modelling Language for UbiCARS (UbiCARS DSML), abstracting technical details to a higher level, minimizing thus the need for technical expertise.
- A DSML model is easily extendable, as new model elements can be added to support additional functionality (e.g., additional explicit/implicit user feedback techniques and alternative recommendation engines).
- It focuses on the ubiquitous scenario of UbiCARS, aiming to improve product recommendations' accuracy and availability: [26] proved that users' implicit feedback improves recommendations' accuracy, whereas [35] has used user explicit feedback data in terms of purchases, together with implicit feedback data in terms of user clicks on products to improve recommendations' accuracy (see "RS for E-commerce" section). The claimed potential improvement of recommendations' accuracy and availability is left as future work ("Conclusions and Future Work" section).
- It can be directly deployed on existing e-stores, making its adoption easier.
- Developers do not need to extend classes or interfaces to use it, while writing code is minimised to the extent possible.
- It supports the usage of sophisticated algorithms and data models from the State-of-the-Art of RS literature. However, it does not model the recommendation algorithms to avoid increased model complexity and maintain flexibility.

Table 2 lists the available tools for aiding the development of RS and their key characteristics.

The UbiCARS MDD Framework

Framework Design Principles

While the works described in "Related work" section related work have utilized innovative methods to tackle RS development complexity, we argue that the level of technical abstraction offered by the proposed methods is somewhat limited and can be further increased, while the need to write code

Table 2 Tools for aiding RS development and their key characteristics

System	Methodology	Purpose	Evaluation	Developers need to do to use it
Hybreed [10]	Component-based approach	Reduce development effort	5 developers: found to reduce effort	Write code: utilise interfaces and instantiate objects
Rojas et al. [14]	OOWS and code generation from UML	Reduce development time	No evaluation	Write code
UM4RS [11]	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. [15]	Pull-based architecture	Assists the development of pull-based mobile CARS	No evaluation	Set up and use the specific infrastructure and algorithms. Author's estimation: need to work on code level (the actual steps for developers are not described in the paper)
UbiCARS MDD framework	Use MDD, defines a DSML and a Modelling Editor	Reduce development complexity & time for non-RS experts (and potentially increase recommendations' accuracy and availability – to be proven)	Evaluated with 20 developers and 17 experts: found to reduce development complexity and expedite the development of UbiCARS (see "Evaluation" section)	(i) Use the UbiCARS DSML and modelling editor to design models; (ii) use the system UI to configure e-stores (see "UbiCARS DSML" and "System Workflow" sections)

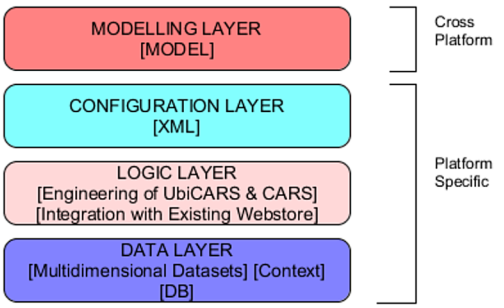

Fig. 1 Software layers of the framework

by developers can be further reduced or even eliminated. To this end, we have specified a list of *Design Principles* (DPs) for the UbiCARS MDD Framework that also constitute the main points of differentiation from related work:

- *DP1.* Technical details be abstracted to the highest level possible.
- *DP2.* Offer automation: UbiCARS models should be defined as abstractions and be able to automatically converted into UbiCARS systems, eliminating the need to write code.
- *DP3.* Recommendation algorithmic details be abstracted from developers: developers, non-experts in RS, should be able to develop RS for commerce in less time than by using any other recommendation framework or developing a RS manually.
- *DP4.* Use intelligent algorithms to not compromise recommendations' accuracy.
- *DP5.* Allow combining user feedback data from both the ubiquitous and online scenarios to potentially improve product recommendations' accuracy and availability (to be proven).
- *DP6.* Be easily extendable: allow for new modules/elements to be added to support additional functionality.
- *DP7.* Increase productivity by offering reuse of UbiCARS models.
- *DP8.* Enable the deployment of UbiCARS on existing and new e-stores.

UbiCARS Methodology

The framework facilitates the design, development and deployment of UbiCARS on e-stores and physical stores in an automated manner, during which developers do not need to write code. It defines the DSML that models the entire recommendation process for UbiCARS, and a graphical modelling editor for the DSML. The editor enables developers to use the DSML towards designing UbiCARS through a model-based approach, and dynamically configuring them on e-stores, both new and existing ones. Figure 1 depicts

the framework's multi-layered software architecture. The DSML, through the modelling editor, acts on the Modelling layer, enabling developers to design UbiCARS models.

Upon UbiCARS design completion, UbiCARS models are exported in XML format, where they serve as configuration files that are then being processed by the Configuration layer via a Parser. The Logic layer is responsible for the engineering of UbiCARS: data models and database tables are being built, while system configurations are also being realized, as these were dictated by the Configuration layer. The Logic layer integrates the new RS with e-stores, provided that the necessary information has been included during design time, such as database URL, database name and platform type (`configFile` element), as well as user feedback-specific information (`DatabaseResourse` elements), see respective paragraphs in "UbiCARS DSML" section. System configurations for the utilization of data from the UbiCARS app (see the following paragraph) are also provided by the Logic layer. The Data layer generates the multidimensional datasets needed to be fed to the CARS system. These datasets include context-aware user-product interaction patterns.

The UbiCARS MDD framework configures a *CARS system* and a *UbiCARS app* as follows:

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

- Enables the tracking of user-product interaction on the e-store: *user ratings*, *browsing history* and *purchase history*.
- Computes personalized recommendations.
- Presents recommendation of products to users within the e-store.

UbiCARS app is a mobile application that:

- Enables the tracking of user-product interaction within the physical store. Similar to [25, 27] that have used the dwell time online as users' implicit feedback (see "RS for E-commerce" section), we propose using the *"Staying Time in front of a product"*.
- Provides access to product recommendations to users within the physical store.

Figure 2 presents a block diagram of the framework that focuses on its components and their interactions. Starting from the DSML that enables the creation of UbiCARS models, these models feed the CARS system that uses them to configure the plugins and the database (DB) on the e-store, the recommendation engine and the UbiCARS app.

From the users' perspective, the framework tracks their activities on the e-store and physical store, as well as related context, provided that the developer has included in the model the corresponding `ContextParameter` model elements, and that the corresponding context sensing plugins exist in the system (see "UbiCARS DSML" section).

From the developer's perspective, interaction with the framework is firstly made through the UbiCARS DSML, via which the developer creates elements in models. Based on these elements, e-store configurations will be conducted. Once the UbiCARS model is created (an XML-based file), it is uploaded to the framework by the developer through the *system UI*. The system UI is an easy to use, three-button user interface pre-installed on the e-store, through which the developer: (i) uploads the model, (ii) conducts configurations—including the creation of the datasets, and (iii) selects a dataset from the set of available datasets and computes recommendations on demand. Recommendations are computed by utilizing the selected dataset.

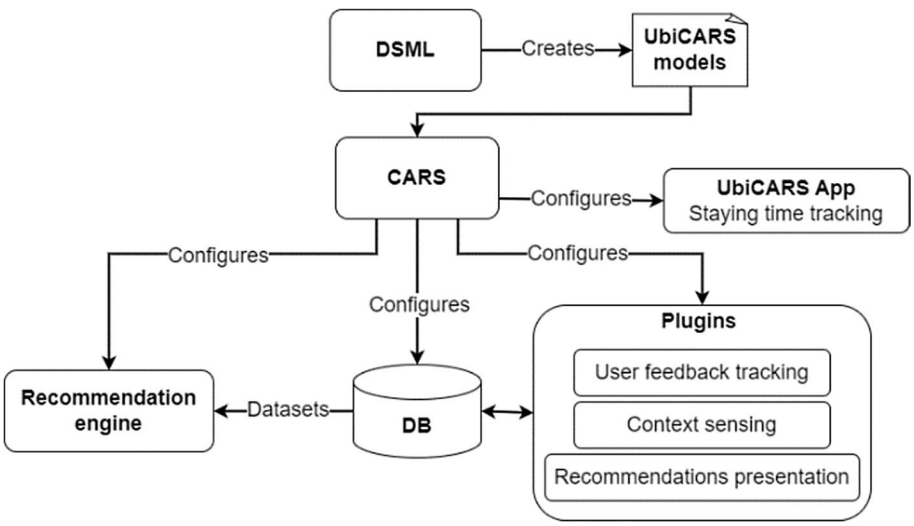

Fig. 2 UbiCARS framework block diagram

Prerequisites

For the integration of the framework with the e-store, the following software modules need to be installed on the e-store in the form of plugins. These modules are provided by the framework.

- System UI: the main UI for the developer to manage the framework.
- Plugins for acquiring user feedback data from the e-store: such plugins are required only for types of user feedbacks for which the e-store does not already store data (e-stores usually store user ratings on products and product purchases).
- Context sensing plugins: used for sensing and storing the context in the database in corresponding tables created by the framework.
- A plugin for presenting the recommendations to users, as well as plugins that offer other functionalities such as registering user mobile device's Bluetooth friendly name (see paragraph "User Identification Across Devices" in "Mobile App Specific Elements" section).
- A recommendation engine: a stand-alone recommendation engine that can be configured and executed by the framework (e.g., via a script). It is required that the engine receives as input datasets of the same format as those of the framework. Framework's dataset format per line (see also Listings 1 & 2 in "Introduction" section), where context is not mandatory:

```
user,item,userFeedback[,context]
```

Configurations

The following configurations are conducted by the framework.

Infrastructure configurations

- Creation of the necessary database infrastructure: the required database tables are created.
- Configuration of the recommendation engine: based on information from the model, specific settings are provided by altering configuration files of the engine.
- Configuration of the pre-installed plugins mentioned in the subsection above "Prerequisites".
- Configuration of the UbiCARS app involves selection of the ubiquitous technology to be used to acquire user feedback data from the physical store.

Context Related Configurations

- Creation of context-aware datasets, one for each of the acquired user feedbacks: ratings, browsing history, purchase history, staying time in front of products, scanning of products.
- Context is automatically included in a user feedback dataset if:
  - The developer has included a `ContextParameter` element in the model and linked it to the corresponding user feedback element, and
  - There is a context sensing mechanism available on the e-store or the physical store for sensing and storing the context in the database in the corresponding table created by the framework. Then, the inclusion of the context data in the dataset is undertaken by the framework (see related paragraph in "UbiCARS DSML" section).

Once recommendations are produced, the framework makes them available to users (customers) through the e-store and the UbiCARS app. The frequency with which recommendations are computed is left upon the developer. The framework does not provide the means for scheduling the computation of recommendations automatically to handle new user preference data that may be added to user profiles at runtime; this can be done using utilities from the hosting server, such as cron jobs.[13]

UbiCARS Architecture

Figure 3 presents the architecture of the framework, focusing on users' (customers') interaction and tracking users' behaviour (user feedback acquisition). The DSML usage and model creation process is not depicted here; rather, we show the framework components after system configurations were conducted. The client-side frontend of the e-store enables for tracking users' behaviour on products via the browser and other third-party software.

To track users' behaviour server-side, users' access to the products' webpages is recorded, as well as their purchases. Ubiquitous user-product interaction is tracked through the UbiCARS app via Bluetooth beacons (estimating users' staying time in front of an item). It is possible for the framework to be extended with more ubiquitous technologies. NFC tagging would be a possibility, where a user would use his/her smartphone to scan a product's tag to indicate a preference to it; however, this requires the user to take explicit action, and thus interrupt his/her current task. All user-product interaction data are stored in the database (DB). These data are later retrieved by the CARS system to compile the corresponding

[13] A utility on Unix-based systems that allows scheduling a script on a server to run automatically at a specified time and date, or repetitively.

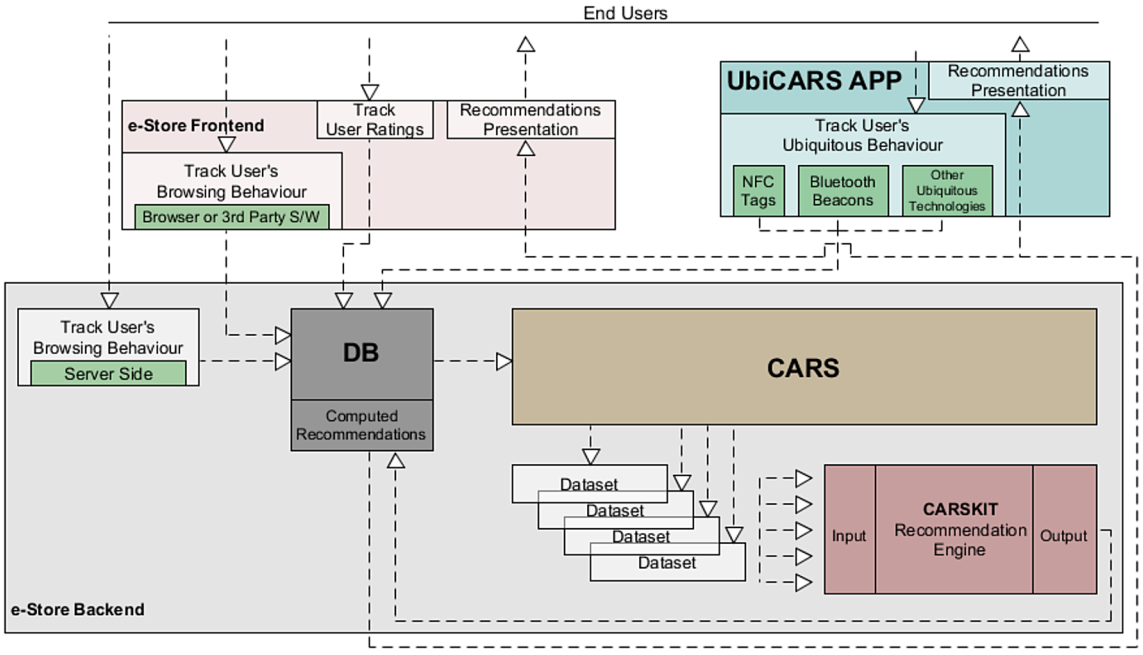

Fig. 3 Framework architecture

datasets and make them available to the recommendation engine, CARSKIT. Resulted recommendations are stored in the database and may be presented through the e-store front-end and the UbiCARS app. *CARSKIT* [52] is a Java-based, open-source engine for CARS that offers algorithms from the State-of-the-Art of Recommender Systems research area (related with DP4). We have selected CARSKIT among other engines because of the variation of algorithms offered, while it is also flexible, easy to use and efficient.

UbiCARS DSML

The UbiCARS DSML aims to abstract the technical details concerning the ubiquitous product recommendation domain to the highest level possible, as well as to expedite RS development, to simplify the usage and instantiation of models by developers. For the creation of the DSML metamodel, we have considered RS research, as well as existing RS and parameters available in popular algorithms. 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 to reduce development complexity and expedite the design of UbiCARS for commerce (related with DP1 and DP3).
- Details of complex recommendation algorithms (such as model-based CF approaches) are abstracted from developers (related with DP3). Their usage is offered in an easy and straightforward manner.
- The DSLM is cross-platform: it can be used to design UbiCARS models for different types of e-store platforms. This approach abstracts platform specific technical details from developers at design time (related with DP1), such as how to create the database resources, how to acquire user feedback data on products, how to integrate a recommendation engine, etc. This reduces complexity and expedites the design. By using the produced models, the framework conducts platform-specific configurations for the platforms it supports, as shown in Fig. 1.
- It offers automation (related with DP2).
- 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 (related with DP7, DP8).
- It is easily extendable, as model elements can be added (removed) for additional (less) functionality, and the updates are reflected in the CARS system in an automated fashion (related with DP6).
- It enhances comprehensibility of the physical and electronic commerce recommendation domain, as well as promotes communication between RS professionals via standardisation of the terminology in the domain.

For designing the DSML, the open-source Eclipse project Sirius[14] has been used. Sirius enables the creation of custom

[14] www.eclipse.org/sirius/.

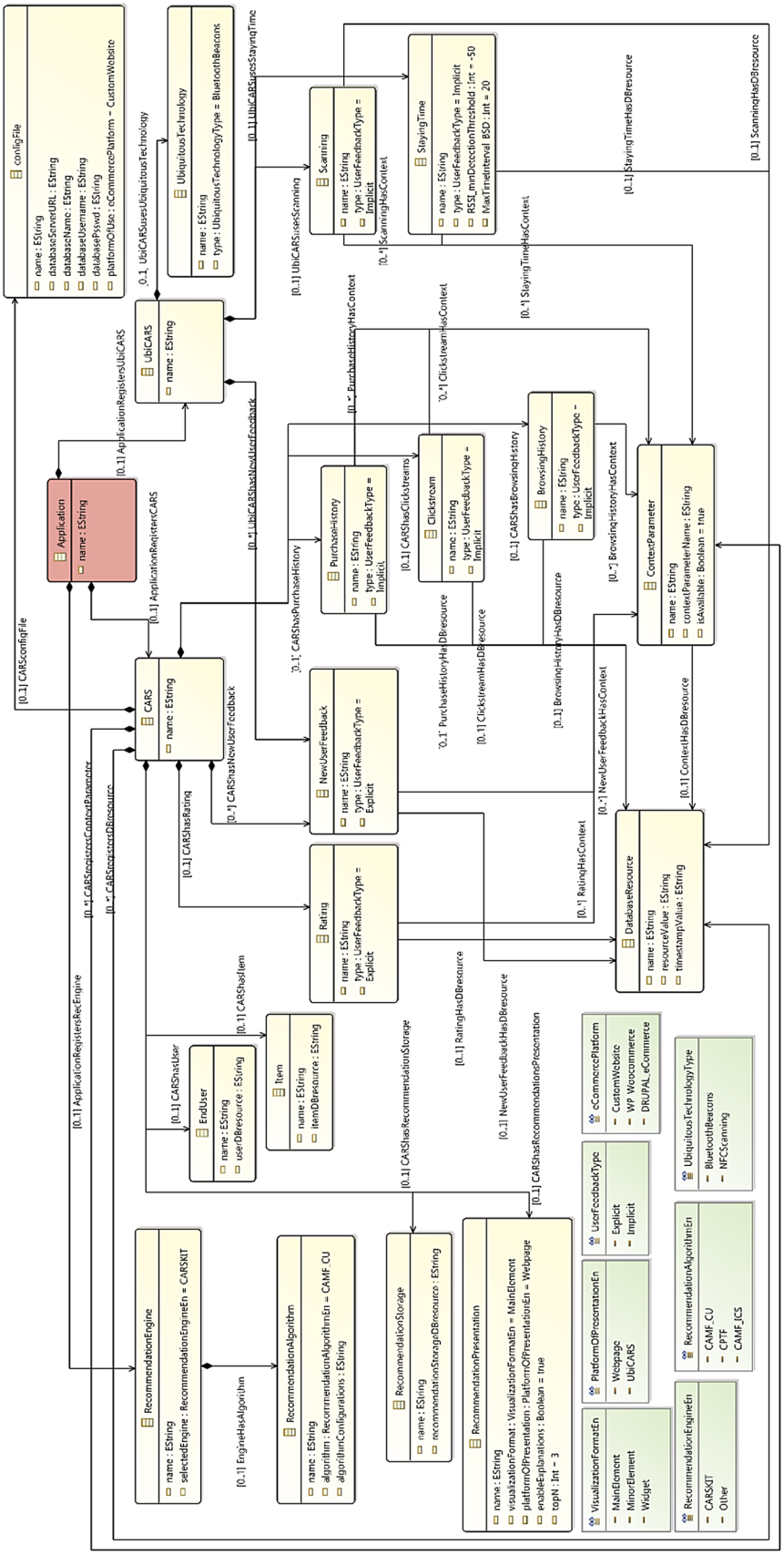

Fig. 4 The proposed UbiCARS DSML

graphical modelling workbenches by utilizing EMF and GMF (Eclipse Modelling technologies). The Eclipse Modelling Framework[15] (EMF) itself is a framework and code generation facility for building tools and applications based on a structured data model. As the authors in [53] note, designing an editor using the Eclipse platform with EMF and GML results in a high-quality product, especially regarding usability aspects. A comparison of development tools for DSML conducted in the same paper (tools compared are: GME, Tau G2, RSA, XMF-Mosaic, Eclipse EMF + GEF) concluded that the Eclipse platform is the tool offering the highest level of graphical completeness, as well as the best usability in terms of user experience, tool feedback, and overall number of features. For DSML development, we have used the Sirius with Obeo Designer Community edition[16] that simplifies the creation of graphical modelling workbenches.

In Fig. 4, the DSML is presented as a metamodel.[17] The `Application` element represents a commerce RS that consists of a `CARS` system and a `UbiCARS` app. `CARS` defines the explicit user feedback element `Rating` that represents users' ratings on products, and the implicit user feedback elements `PurchaseHistory`, `ClickStream` and `BrowsingHistory`. A CARS can instantiate at most one of each of these elements.

The `NewUserFeedback` elements may be defined by the developer if needed as new, custom user feedback techniques that can be either explicit or implicit (default is explicit). `NewUserFeedback` elements essentially constitute a means for the developer to define new types of user feedback on products within the model. An example of new type of user feedback on products is the number of times a user has watched a product video description through the e-store: the more times watched, the more interested the user is in that product.

If additional user feedbacks are included, then the developer needs to develop the necessary functionality to acquire the new user feedback data from the users via the e-store or app.

The `UbiCARS` app acts in the physical store and utilizes the `StayingTime` implicit user feedback element that represents the staying time in front of products. The DSML also includes the `Scanning` implicit user feedback element as an alternative to the `StayingTime`, that represents the scanning of NFC tags of products. `UbiCARS` element has at most one of each of these elements, while explicit/implicit `NewUserFeedback` elements are also available to be defined by designers.

It is mandatory for each of the user feedback elements, i.e., `Rating`, `PurchaseHistory`, `ClickStream`, `BrowsingHistory`, `StayingTime`, `Scanning` and `NewUserFeedback` to be linked to one `DatabaseResource` element, while it may also be linked to a number of `ContextParameter` elements. The `DatabaseResource` element, through parameter `resourceValue`, defines where in the database the user feedback element that is linked to it will be stored and how its information can be retrieved. The `timestampValue` parameter is used to denote whether timestamp is to be used as a time related contextual information (by retrieving the time that the user-product interaction occurred) 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.

To record the context in which user-product interaction occurs (e.g., users' staying time), the `ContextParameter` element is used and linked to the respective user feedback element (e.g., `StayingTime` element). As an example of context, we note the location of the user, which can be expressed either by means of numbers (i.e., GPS coordinates), or by using arbitrary keywords, such as "smartphone section within store" or "1st floor". `isAvailable` specifies whether the respective context sensing mechanism for the `ContextParameter` element has been implemented and can be used, or whether it needs to be implemented by the developer. The `ContextParameter` element is linked to a `DatabaseResource` element as well, that specifies the database location where the respective context information will be stored (or has already been stored). With the above design, the framework becomes aware that a dataset is context aware. Implementation of context sensing mechanisms and context plugins (e.g., to acquire users' location) is undertaken by the developer; however, the entire infrastructure in terms of context data storage, the inclusion of context in datasets, the compilation of 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.

The `RecommendationEngine` element is responsible for the recommendations' computation. The CARSKIT recommendation engine is specified in the metamodel as the default engine; nevertheless, the framework allows for other engines to be integrated and utilized by selecting "Other" as the value of the `selectedEngine` parameter that corresponds to the `RecommendationEngineEn` enumeration in the metamodel. The `RecommendationAlgorithm`

[15] projects.eclipse.org/projects/modeling.emf.emf.

[16] www.obeodesigner.com/en/download.

[17] The metamodel in high resolution: www.cs.ucy.ac.cy/~mettour/Journal/CARSMetamodel.png.

element is linked to the recommendation engine and specifies the algorithm in use. The `RecommendatioAlgorithmEn` enumeration specifies the available recommendation algorithms for the corresponding recommendation engine, which can be selected through the `algorithm` parameter of the `RecommendationAlgorithm` element. In case CARSKIT is the developer's recommendation engine of choice, the metamodel assigns context-aware matrix factorization `CAMF_CU` as the default algorithm in use. Two additional algorithms (`CAMF_ICS` and `CPTF` Tensor Factorization [52]) are available in the metamodel. The metamodel is extendable with additional algorithms from CARSKIT; this is simply done by adding more enumerations in `RecommendatioAlgorithmEn` (see Fig. 4). For example, the developer may select a hybrid algorithm, or any other recommendation algorithm that could produce better results when lack of data exists for a new user or a new item (cold start problem), in which case the matrix in a matrix factorization method would be sparse, and hence, the algorithm ineffective. Algorithmic configurations to the selected algorithm can be conducted 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.

While `RecommendationStorage` defines the place in the database, i.e., the database resource, where recommendations are stored, the `RecommendationPresentation` element defines, among other, the platform on which the recommendations will be presented to users (`platformOfPresentation` element) – whether this would be the e-store via a web interface or through a mobile-friendly interface on the UbiCARS app (these are defined through the `PlatformOfPresentationEn` enumeration). Currently, the framework provides a web interface through a plugin for presenting the recommendations to users (see "UbiCARS Methodology" section), which also serves users via the UbiCARS app, provided that they are logged-in to the e-store via the smartphone's browser. In addition, the `RecommendationPresentation` element defines the `visualizationFormat` of the recommendations through the `VisualizationFormatEn` enumeration: should the recommendations be presented as a main screen element (e.g., as a webpage dedicated to recommendations), as a minor one, or perhaps in the form of 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, among other, to system transparency, trust, and satisfaction [54]. For example, Amazon.com uses *"Customers Who Bought This Item Also Bought…"*. Although the task of offering correct and precise explanations is difficult, especially when algorithms with high complexity are being used, developers are enabled to offer a simpler and broader version of recommendation explanations, e.g.: *"Based on your product ratings to date, as well as on your previous transactions and product interaction in our showroom, the following products are recommended for you!"*. The `topN` parameter specifies the total number of recommendations to be displayed to users (e.g., top-5).

Supported Platforms

Two widely used, open-source e-commerce platforms are used by the framework: WordPress WooCommerce[18] and Drupal Commerce.[19] The WooCommerce plugin is updated for acquiring and storing users' browsing history on items. The users' ratings and product purchases are stored in the database by the e-stores and the framework is configured to use these data.

Custom platforms are also supported. A custom platform refers to any other type of e-store platform. In case a custom platform is used, for user feedbacks for which the platform does not store the data (e.g., most platforms already store users' ratings on products and product purchases), the framework creates the database tables required to store the user feedback data. The acquisition of these user feedback data that stem from user interaction with the e-store is left on the developer.

For user feedbacks for which the custom platform stores the data, as the structure of the corresponding database tables is unknown to the framework, the developer needs to write code to retrieve the data and compile the corresponding datasets.

The model allows developers to select their platform via the `platformOfUse` parameter of the `configFile` element (see Fig. 4), which uses the enumeration `eCommercePlatform`.

Model Reuse

A UbiCARS model may be reused for the configuration of other e-stores, increasing thus productivity. For this to be feasible, the platform of the target e-store needs to match with the platform specified in the model (`platformOfUse` parameter). Then, for the model to be reused as is, the following apply.

[18] wordpress.org/plugins/woocommerce/#installation.

[19] drupalcommerce.org/.

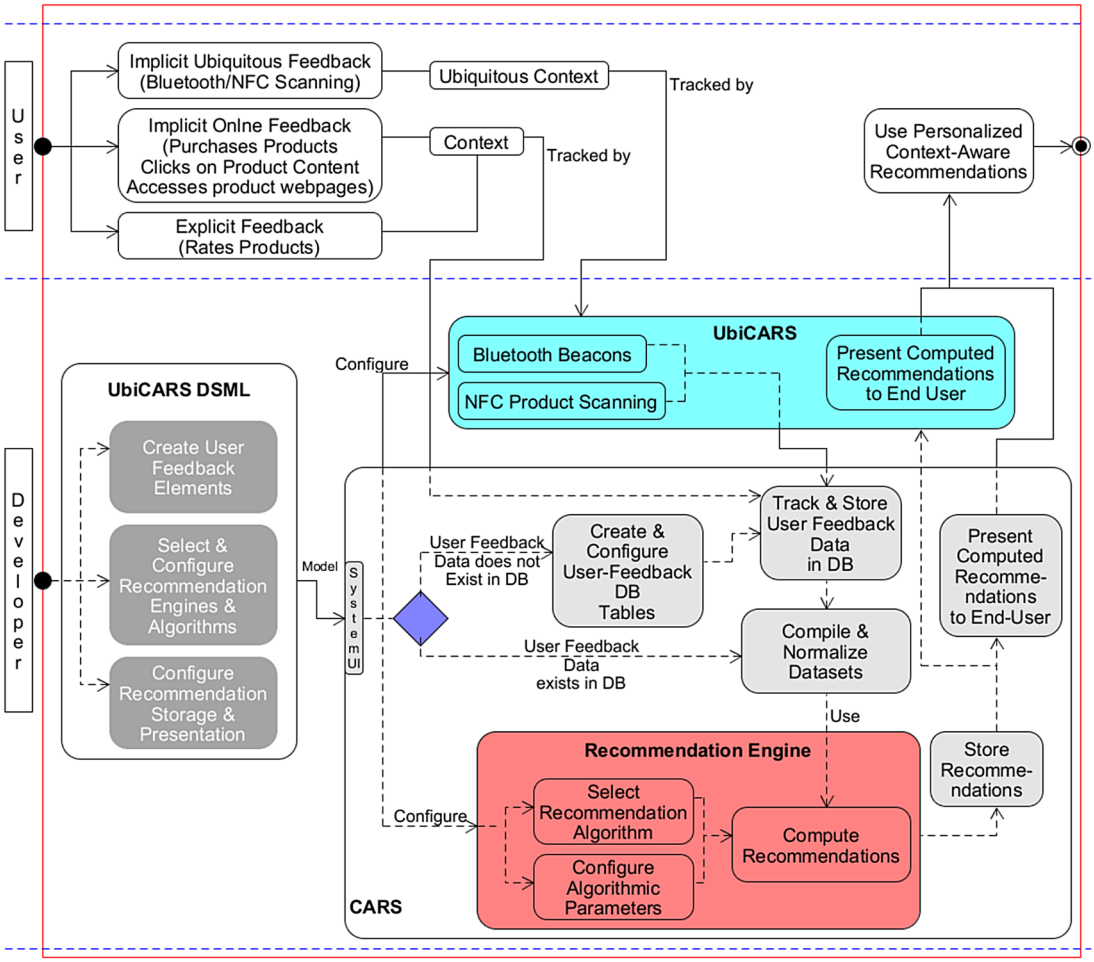

Fig. 5 System workflow

In case a custom platform is used, for each database resource in the model, the system checks whether the corresponding table exists in the database to use its data, otherwise it creates it. This enables the usage of pre-existing data, provided that the pre-existing table follows the framework's data format.

In case WordPress WooCommerce or Drupal Commerce are used, based on user's input regarding the resource parameter of the database resource element in the model (see Fig. 8 as an example), the system uses the e-stores' pre-defined tables and any pre-existing data in terms of user feedback techniques (for technical details see paragraph "Pre-defined tables in supported e-stores" in "Appendix"). Therefore, for model reuse to be feasible, the target e-store should be set-up having the same characteristics with the source platform, as specified in the model. If not, the model needs to be updated in terms of the resource parameter of the corresponding database resource to adhere to the target e-store's characteristics.

Model Correctness

To avoid errors during configuration, UbiCARS models need to be verified. The first verification step takes place during design, as the DSML metamodel and the editor restrict the user from performing a number of invalid moves. For example, a `PurchaseHistory` element may be connected at most to one `DatabaseResourse` (see Fig. 4), and this can only be done by using a specific type of edge, the `PurchaseHistoryHasDBResource`. Requiring a specific edge type to connect a particular element to another element makes the design more comprehensible, transparent and error free, but also extends the number of available tools in the editor's toolbox (see Fig. 8), possibly making the tool selection process longer. The second step takes place during configurations, where the system informs the user in case of errors, e.g., more than one element is requesting access to the same database recourse.

System Workflow

Figure 5 depicts the system workflow with a clear representation of the two user roles: the developer and the user (customer). After the model has been designed and uploaded to the system through the System UI, the actions depicted in the figure are then performed by the framework.

After the model has been parsed by the system UI, the CARS system checks whether user feedback data pre-exist in the database. For every user feedback element in the model, a `DatabaseResource` element defines the database table where the corresponding data will be stored. There are two cases where such tables pre-exist: (i) when the default e-store database includes such tables (e.g., for product purchases and user ratings on products), and (ii) when the model is re-executed by the system: a developer may update parts of the model and repeat the execution process, in which case the tables (and any data in them) will pre-exist. If the database table does not exist, the system creates it and configures the system so that, from that point onward, the corresponding user feedback data from the e-store or the app are stored in that table.

Tracking and storing user feedback data is the next step after creating user feedback tables (Fig. 5). This step occurs only in cases where the user feedback data are not already being tracked and stored by the e-store. The framework requires from the developer to create the necessary mechanisms to acquire the user feedback data and store them in the corresponding database table created by the framework in the previous step. In the case of the WordPress WooCommerce platform for example, as the user browsing history data do not exist, a plugin was developed and is provided by the framework that tracks the users' browsing history and stores the data in the database.

In cases where the default e-store database includes the user feedback data (alternative flow in Fig. 5), the system uses them. For example, for the WordPress WooCommerce platform, the system checks whether the user ratings and user product purchases exist and uses them.

In case the UbiCARS framework is to be applied on a new type of e-store platform other than those already supported, then in cases where the default e-store database includes user feedback data, the developer is required to write code to retrieve the data and compile the new datasets.

Figure 6 depicts the parsing and configuration steps followed by the UbiCARS Framework. Input to this process is the UbiCARS model. 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). If the model specifies one of the two supported platforms, then the required configurations are conducted by the framework. If the model specifies a custom platform, then, during the first execution, all the database tables that correspond to the `DatabaseResource` elements will be created. 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). 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 only and make it available for the computation of recommendations. The developer may at any time initiate compilation of any dataset, provided that the aforementioned requirements are met, and then, initiate computation of recommendations.

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 Fig. 6). 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. During following system iterations where context resources and the respective context data will have been included in the database, context-aware datasets will be compiled, and context-aware recommendation computation will be initiated.

Mobile App Specific Elements

The `UbiquitousTechnology` element specifies the ubiquitous technology in use by the UbiCARS app, as depicted in Fig. 4. Currently, `BluetoothBeacons` and `NFCScanning` are supported, as shown by the `UbiquitousTechnologyType` enumeration. Alternative potential technologies for use are Wi-Fi, or more modern indoor positioning technologies such as *smart floors* [55].

Bluetooth Beacons

Bluetooth beacons can be used for indoor positioning, by communicating with Bluetooth-enabled devices in the proximity. Smartphones for example, through appropriate software, can estimate their distance from Bluetooth Beacons. Typical beacon range is 7 m to a few hundreds of meters. There are already examples of physical stores that use beacons to offer a more engaging experience to customers while shopping [56].

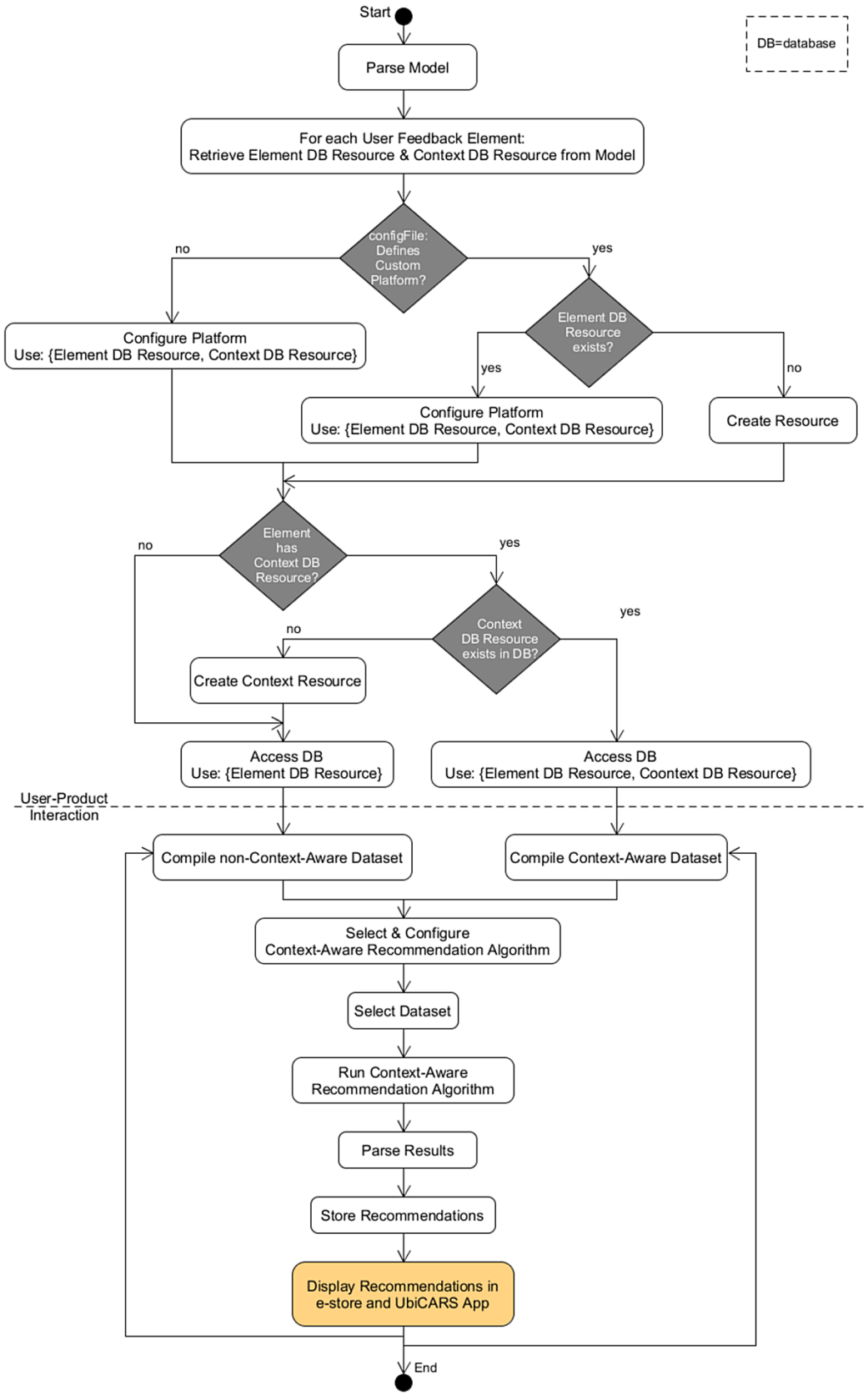

Fig. 6 UbiCARS framework configuration steps

Attaching Bluetooth beacons on products that are placed adjacent to each other (in shelves or showroom tables) causes signal coverage overlap. To overcome this, the Bluetooth Received Signal Strength Indicator (RSSI) is used by the UbiCARS app to estimate users' staying time in front of a specific product. Users only need to install and run the UbiCARS mobile application. RSSI serves as an indication for the distance between the mobile device and the beacon: the closer the device is to a beacon, the higher the registered RSSI value. Depending on the particular beacons sensed by the user's device and the respective RSSIs registered, the system estimates the distance of the device from each Beacon and opines on the products the user has stayed in front of and for how long. The `RSSI_minDetectionThreshold` parameter of the `StayingTime` element (see Fig. 4) is used to denote the minimum possible RSSI value that indicates that the user is near enough to the product to be considered that they "have stayed in front of the particular product". During experiments in the laboratory, we have acquired RSSI values from −30 (~2 m) to −120 (~8 m), based on which, the `RSSI_minDetectionThreshold` default value was set to be −50. As retrieved RSSI values heavily depend on the type of sensors used, the characteristics of the space and the contained objects, a number of tests will be required to properly adjust the `RSSI_minDetectionThreshold` to suit the specific use case.

`MaxTimeInterval_BSD` refers to the maximum time interval between successive detections of a user in front of a product, so that these detections are considered in the same "staying time session" in front of that product (the default value is 20). Figure 7 describes the staying time reasoning algorithm for the abovementioned default values.

User Identification Across Devices

Using a simple interface on the e-store, users are able to register to their user profile the Bluetooth friendly name of their mobile device. This name does not correspond to the real name of the user; rather, it is used as an identifier when pairing Bluetooth devices with the user's mobile device. In

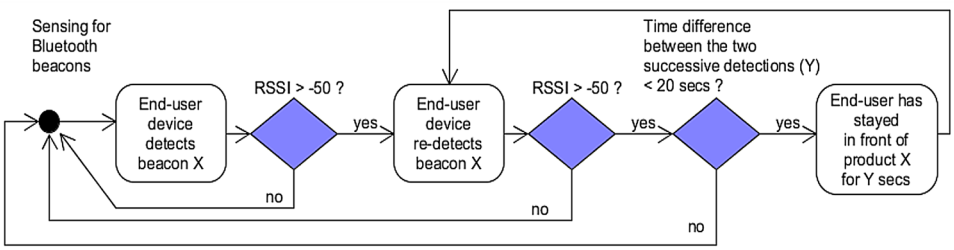

Fig. 7 Staying time reasoning algorithm

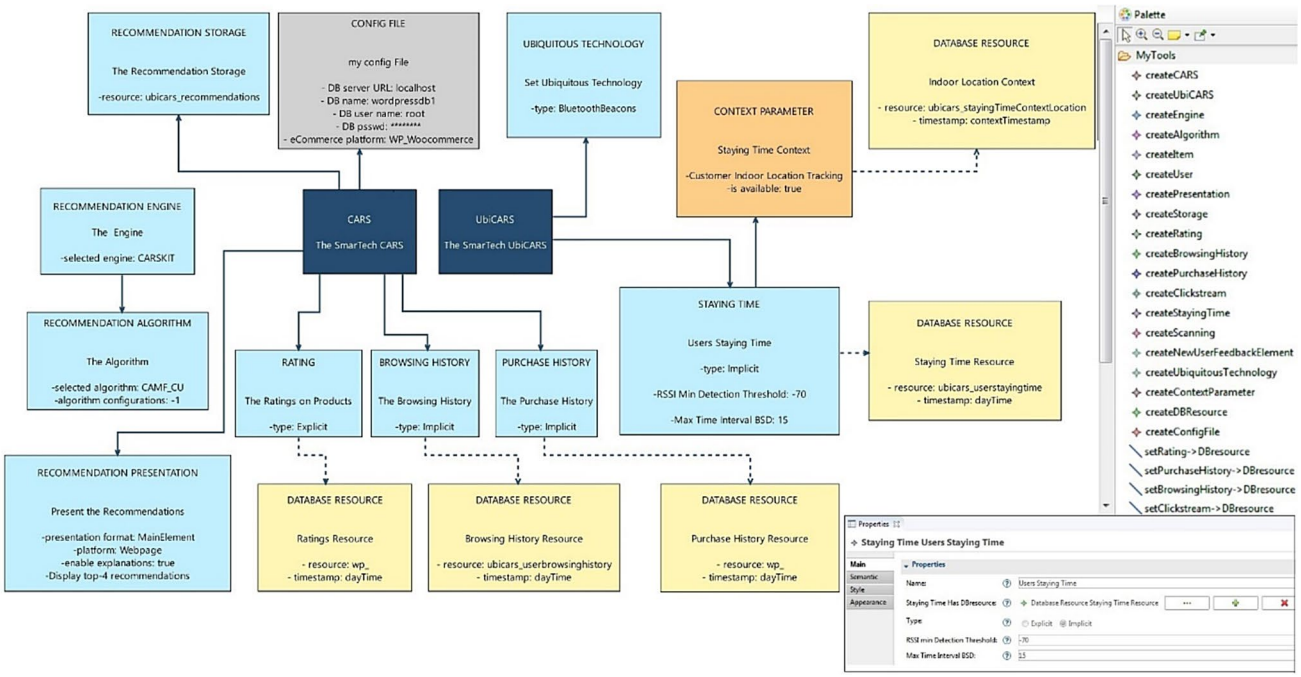

Fig. 8 UbiCARS model instance

this manner, the user device's friendly name is associated with the user's profile. After sensing a beacon, the UbiCARS app sends to the system the beacon's H/W address that uniquely identifies the item, together with the RSSI value and the Bluetooth friendly name of the user's mobile device. The system associates the Bluetooth friendly name with the user ID from the database and stores the received data to the database as tracking data. Thus, the user is being identified in both the physical store and the e-store. The system ensures that Bluetooth friendly names of users stored in the system are unique by checking whether the new friendly name already exists in the system with each new name insertion. If it does, the user is prompted to change his/her device friendly name and try again (changing the name is a simple process).

Privacy Concerns

The tracking of users is strictly conducted within a physical store. Tracking data could potentially reveal that the user was indeed in the store during a certain period of time. In this paragraph we discuss that our method does not raise any privacy concerns for the following reasons:

- The data are anonymized: the user is being tracked using his/her Bluetooth friendly name that is associated to the user ID, and in relation to his/her distance from a H/W address of a beacon. Such tracking data alone cannot trace back to the name of the user, neither the items that the user interacted with, unless the entire database is compromised.
- The tracking data are stored in a database table. They are not available to any type of users such as administrators or e-store owners, and furthermore, they are not accessible from any user interface within the system UI, the e-store or the app, as such functionality was not implemented.

The above information should be available to users though the privacy policy of the system and all users should give their consent before using it.

Design Demonstration

This section demonstrates the usage of the DSML in designing a UbiCARS model. In Fig. 8, a model within the editor and part of the editor's toolbox[20] are depicted. The toolbox is situated at the right-hand side of the editor. To add an element in the editor, the user clicks on the respective item in the toolbox and then clicks in the white space within the editor or drags-and-drops the element from the respective item toolbox to the canvas. Through the properties view (also shown in Fig. 8), developers add/edit information regarding the corresponding element.

Regarding the UbiCARS models, there are 4 types of Elements: dark blue are the CARS and UbiCARS elements that serve as starting points for designing the model; light blue are the main elements of a model, including all user feedback elements, ubiquitous technology elements, recommendation engine, algorithm, storage, and presentation elements; with yellow colour are the Database Resource elements; and with orange are the Context elements.

For testing purposes, both WooCommerce and Drupal Commerce platforms have been set up and used as example e-stores. Each e-store included several electronic products, similar to real life e-stores.

Cold Start Problem

At this point, the cold start problem affected system execution, since after system configuration has completed, the CARS system was unable to compute recommendations due to the lack of user-product interaction data (new user problem). To address this, lab personnel interacted with the e-stores to produce user feedback data. This meant to represent real-life user-product interaction activity on an e-store (rating, browsing, and buying products).

To simulate the physical store scenario, the typical layout of a smartphones showroom was used in which lab personnel interacted with the products (Bluetooth enabled smartphones were used that could act as Bluetooth beacons as well). We note that the system is able to utilize any pre-existing data that has resulted from prior user-product interaction. For example, if users' ratings on products exist in the Drupal Commerce e-store prior to system configuration, the system will use the data during configuration and for computing recommendations.

Using the abovementioned system setup, the framework was able to produce four datasets: ratings, purchasing history, browsing history (resulting from users' accesses on product webpages) and staying time. Each dataset was able to produce some recommendations.

Evaluation

Process Description

The framework was evaluated via three different evaluation processes.

[20] A video on the usage of the UbiCARS Modelling Editor: www.youtube.com/watch?v=FRJ7nw2J3c4.

Table 3 Developers' effort comparison for the three implementation methods

Metrics	Methods		
	Modelling using supported e-stores	Modelling using custom e-stores	Manual method: coding
Number of Lines of Code (logical lines of code excluding blank lines, comments, and prints)	18 (c)	12 (uf) 18 (c) 13 (o) ratings 13 (o) purchasing h Total: 56	358
Number of database queries	0	2 (uf) 2 (o) Total: 4	30
Development Time (hours)	2 (c)	4 (uf) 2 (c) 2 (o) Total: 8	44
Software modules to be developed	(c) Purchase History	(uf) Browsing History (c) Purchase History (o) Compilation of datasets	(uf) Browsing History (uf) Staying Time (c) Purchase History (o) Compilation of datasets (o) Recommendation engine configuration (o) Computation of recommendations (o) Storage of recommendations (o) Presentation of recommendation

The *modelling against coding comparison* was conducted by the authors and aimed to directly compare the proposed modelling method with the manual (coding) method to estimate the effort a developer needs to put for each of them in developing a UbiCARS. The metrics used were: number of logical lines of code, number of database queries and development time in hours.

The second evaluation process was conducted via a *survey by using a task-oriented questionnaire*. The aim was e-store developers and engineers with no expertise in RS to use the UbiCARS MDD Framework to deploy a UbiCARS on an e-store: use the DSM to create a model, upload it via the system UI, perform the configurations on the e-store and generate recommendations. Twenty participants participated in this evaluation.

The third evaluation was a *remote evaluation* that took place after the task-oriented evaluation was completed. The aim of the third and final evaluation method was to allow for more experts to use the UbiCARS framework and receive feedback on framework usage. Seventeen participants participated in the remote evaluation.

In the remaining of this subsection, we describe the evaluation processes in detail.

Modelling Against Coding Comparison

We consider developing a RS of an e-store that uses users' ratings on products, users' browsing history on products (number of product webpage accesses), users' purchase history and users' staying time in front of products as user feedbacks. In addition, the purchase history user feedback is context-aware regarding the type-of-day (whether the purchase was conducted on a weekday or weekend) and the time-of-day (whether the purchase was conducted in the morning, noon, afternoon, evening, or late at night). For the computation of context values, the purchase timestamp is used (no database query needed).

We assume that the e-store has records of its users' ratings on products and its users' product purchases in its database, as any modern e-store. Thus, for acquiring these data, a developer would only need one database query for each of these feedbacks.

We examine the development of the UbiCARS in three different ways:

- Using the UbiCARS framework with one of its supported e-store platforms (modelling using supported e-stores).

The context sensing and storing plugin needs to be developed.

- Using the UbiCARS framework with any other type of e-store platform (modelling using custom e-stores). In terms of user feedbacks, the developer is required to develop only one module, to acquire the missing user feedback (browsing history). The modules for the compilation of datasets for ratings and purchasing history also need to be developed, as the framework is not aware of the respective database tables' format. For each one of them, a query for retrieving the data and respective code for compiling the dataset is required. Regarding browsing history, the framework will automatically create the database table based on the model, thus its respective mechanism for dataset compilation will be used. The context sensing and storing plugin also needs to be developed.
- By developing the UbiCARS (manual method: coding in PHP, JavaScript, and MySQL).

For the three ways to be comparable, we did not develop a recommendation algorithm for the manual method; instead, we used the CARSKIT recommendation engine as with the other methods. In addition, for the analysis of the manual method, we excluded the code for the mobile app, as the modelling process does not directly act upon that code. Instead, any configurations in relation to the UbiCARS App specific elements (e.g., applying the Staying Time Reasoning Algorithm, see Fig. 7) are conducted on the e-store and its database (see "Mobile App Specific Elements" section).

The above methodology aims to estimate the effort a developer needs to put for each of the three ways of developing the UbiCARS. The objective is to compare the three methods in terms of the metrics shown in Table 3. One expert computer scientist from the authors performed the task of developing the UbiCARS using the three different ways and the coding task was reviewed by one of the other authors to ensure the quality of the approach and the code.

The metric "Software modules to be developed" is measured independently from the other metrics and indicates the different software modules the developer needs to develop from the following types: module to acquire user feedback (uf), module for context sensing and storing (c), other software modules (o).

The metrics for the manual method stem from an analysis of the respective source code of the developed UbiCARS that is based on the source code of the UbiCARS framework (PHP, JavaScript, MySQL). In this analysis, we have used parts of the code that are required to develop a UbiCARS, according to the requirements set in the beginning of this section. The values of the metrics are perceived as an estimation of the effort a developer would need to develop a UbiCARS. Please note that we cannot guarantee that our implementation is the most efficient in terms of these metrics.

Evaluation with Developers: Task-Oriented Evaluation

The evaluation of the framework was conducted via a survey by using a task-oriented questionnaire.[21] The aim was for users in our target group—e-store developers and engineers with no expertise in RS—to use the UbiCARS MDD Framework to deploy a full UbiCARS on an e-store to the point where recommendations are presented. For a detailed description of the steps followed, interested readers may refer to the tasks defined in the questionnaire. The evaluation was conducted by involving one participant at a time, with the first author observing the entire process with each participant. Participants were also given a short developer's manual.[22] Estimated duration for an evaluation session was 60 min (reading time for manual excluded). Professional developers with no RS expertise were invited by the authors to participate. 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. For the purposes of this evaluation, participants were asked to assume that they are developers for an e-store for electronic products that also has a physical store.

The survey was split into three parts: the first one constituted the modelling part, where participants were requested to use the modelling editor and its toolbox to add and edit elements on the canvas to create a UbiCARS model (see Fig. 8). The process consisted of nine tasks: tasks 1–8 and task 14, as shown in Table 4 (in "Appendix"). Each of the nine tasks was comprised of several subtasks. As participants were requested to use a modelling language they have not used before in one and only session to build a UbiCARS, a detailed description of the initial tasks was needed. Especially for the initial tasks where participants were inexperienced with the editor, the subtasks were designed to be simple activities; as the evaluation progressed though, subtasks' complexity increased while their description details decreased. Furthermore, towards the end of the evaluation, Task 14 required from participants to re-execute the whole process without having any instructions on how to do so, i.e., to return to the modelling editor to enhance the model by including additional elements. This task aims to examine whether participants became familiar with the language. Table 4 summarizes the aim of each task, as well as the number of their subtasks. By successfully completing the first 8 tasks, a full UbiCARS model was constructed that could successfully be deployed on the system UI. Task 14 was

[21] The questionnaire can be found here: www.cs.ucy.ac.cy/~mettour/Journal/UbiCARSQuestionnaire.pdf.

[22] Developer's manual can be found here: www.cs.ucy.ac.cy/~mettour/Journal/UbiCARSDevelopersManual.pdf.

executed only after the next part of the evaluation survey, namely the configuration part, was completed. The aim was to include customers' location in the physical store as a context parameter, to enable context-aware recommendations.

The second part of the survey, namely the configuration part, consisted of six tasks (tasks 9–14, see also Table 4 in "Appendix"), during which participants were instructed on how to upload their model on the system UI to conduct e-store configurations, datasets compilation, recommendations computation and recommendations presentation through the e-store. These tasks were short in duration and simple to accomplish, mainly requesting from participants to use the system UI and observe the computed results. Here, task 14 includes the execution of the model designed in task 14 of the modelling part.

The third and final part of the evaluation was the questionnaire. It did not include any tasks for participants to work on. Instead, it included demographic questions, questions related to users' experience with the modelling editor and the system UI, as well as questions on the usefulness of the framework and its ease of use. Similarly to the work in [57], we have used questions from the Technology Acceptance Model (TAM) satisfaction questionnaire [58, 59], and from the User Experience Questionnaire—UEQ [60, 61]. Additional questions were also defined that explicitly targeted the evaluation of the UbiCARS framework. For the questionnaire, we have used 7-point Likert scale agreement questions (1 corresponds to "Strongly Disagree" and 7 to "Strongly Agree").

Nineteen developers and an e-store administrator participated in the evaluation for about one full hour each (briefing excluded), completing tasks and answering questions. Although the number of participants is statistically small, our results are significant since these 20 participants fall well within our target group, i.e., developers with no experience in developing RS, and they have each used the framework for a considerable amount of time. The e-store administrator, although not a developer himself, he was leading a team of e-store developers. The 19 developers had at least four years of programming experience each and 10.52 years on average. In terms of experience 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.

Evaluation with Experts: Remote Evaluation

To allow for more experts to use the UbiCARS framework, a remote evaluation[23] of the framework took place after the task-oriented evaluation with developers was completed. Professionals and researchers on MDD and recommender systems were invited through social networks, professional groups, and mailing lists to participate. They could participate either by downloading and using the modelling framework[23], or, in case they did not have time available, by watching a video[24] about the framework and then respond to a questionnaire.[25] The questionnaire included similar questions to the one used for the task-oriented evaluation, but it needed less time to be completed. By downloading the framework, participants were able to interact with a model that represented a RS for an e-store. The model was provided instead of guiding participants to design it themselves to keep the evaluation session short. In case, however, they would like to design it from start to end, they were provided with a design guide.[26] After interacting with the modelling editor, participants were instructed[23] to access the e-store and use the System UI to upload the model, compute recommendations and view them through the e-store.

Seventeen participants participated in the remote evaluation. Fifteen of them by watching the video, whereas two participants downloaded and used the framework after they had watched the video. Reasons that most participants preferred to watch the video are that this was the easiest and least time-consuming approach, whereas using the framework required downloading a zip file and following instructions to run it[23]. While we are aware that users may avoid downloading and using software if there is an alternative, we have made attempts to make the process as easy and quick as possible, estimating that it would take about 20 min for the participants to complete.

Thirteen participants stated to be developers among other, four participants stated to be MDD experts among other and five participants stated to be RS experts among other. The two participants that used the framework were not among the MDD experts, neither among the RS experts.

[23] Instructions: www.cs.ucy.ac.cy/seit/wp-content/uploads/2020/11/UbiCARS-MDD-Framework-Evaluation.pdf.

[24] www.youtube.com/watch?v=YvYm4EBZ1pw&feature=youtu.be.

[25] The questionnaire can be found here: www.cs.ucy.ac.cy/~mettour/Journal/Questionnaire.pdf.

[26] www.cs.ucy.ac.cy/~mettour/Journal/UbiCARSModelDesignGuide.pdf.

Summary of Evaluation Results

Table 5 summarizes the evaluation results by displaying the mean and standard deviation (σ) for each question. Evaluation results were overall very positive. In terms of the task-oriented questionnaire, in about one hour on average, participants non-experts in the RS domain were able to successfully utilize the capabilities of the UbiCARS DSML, the modelling editor and the system UI, to design, develop and deploy UbiCARS on an example e-store. Participants stated that they have understood the results of each completed task. 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 much mental effort was not needed to interact with the framework, and, furthermore, stated feeling positive toward the UbiCARS framework.

Regarding the remote evaluation, results were also positive, but the means were smaller by 1.22 on average ($\sigma = 0.3$), in comparison to the results of the task-oriented evaluation. This indicates that, although participants see the framework positively in terms of usefulness (average mean = 5.2) and ease of use (average mean = 5.11), they find it less useful and not as easy-to-use in comparison

Table 5 Summary of evaluation results (7-point Likert scale was used: 1 corresponds to "strongly disagree" and 7 to "strongly agree")

Question	Task-oriented evaluation		Remote evaluation	
	Mean	σ	Mean	σ
Perceived usefulness of the UbiCARS MDD framework				
Using the UbiCARS framework would enable me to develop Recommender Systems more quickly	6.75	0.64	5.47	1.18
Using the UbiCARS framework would improve my performance in developing Recommender Systems	6.6	0.68	5.06	1.39
Using the UbiCARS framework would increase my productivity in developing recommender systems	6.75	0.55	5.12	1.41
Using the UbiCARS framework would make it easier to develop recommender systems	6.7	0.57	5.12	1.5
I would find the UbiCARS framework useful in developing recommender systems	6.8	0.41	5.29	1.53
Using the UbiCARS framework would enhance my effectiveness in developing recommender systems	6.45	0.88	5.12	1.45
Perceived ease-of-use of the UbiCARS MDD framework				
I would find the UbiCARS framework easy to use	6.5	0.76	5.18	1.38
Learning to operate the UbiCARS framework would be easy for me	6.3	0.92	5.59	1.46
I would find it easy to get the UbiCARS framework to do what I want it to do	5.95	0.94	4.94	1.39
My interaction with the UbiCARS framework would be clear and understandable	6.3	0.92	4.81	1.05
I would find the UbiCARS framework to be flexible to interact with	6.05	0.94	4.71	1.69
It would be easy for me to become skilful at using the UbiCARS framework	6.2	1.15	5.41	1.28
Other questions				
Using the UbiCARS framework for developing Recommender Systems would reduce my effort in terms of lines of code and database queries needed	6.7	0.73	5.65	1.54
I would revisit the UbiCARS framework within a week's time if it was available for use	5.95	1.36		
I would revisit the UbiCARS framework regularly if it was available for use	5.65	1.30	4.82	1.74
It does not require a lot of mental effort to interact with the UbiCARS framework	6.2	0.95	5.12	1.22
Using a Model Driven Development approach through the UbiCARS framework to develop Recommender Systems is a good idea	6.75	0.44	5.88	1.32
I feel positive toward the UbiCARS framework	6.8	0.41	5.47	1.46

Table 6 UEQ mean scores for remote evaluation on a scale from 3 (highly positive) to −3 (highly negative)

Enjoyable	0.88	Understandable	1.56	Creative	1.13	Easy to learn	1.31	Valuable	1.31
Exciting	1.13	Interesting	1.69	Predictable	1.13	Fast	1.69	Inventive	1.38
Supportive	0.88	Good	1.73	Easy	1.38	Pleasing	1.31	Leading edge	0.69
Pleasant	1.38	Secure	0.06	Motivating	1.06	Meets expectations	1.25	Efficient	1.05
Clear	1.13	Practical	1.19	Organized	1.06	Attractive	0.31	Friendly	1.13
Innovative	1.13								

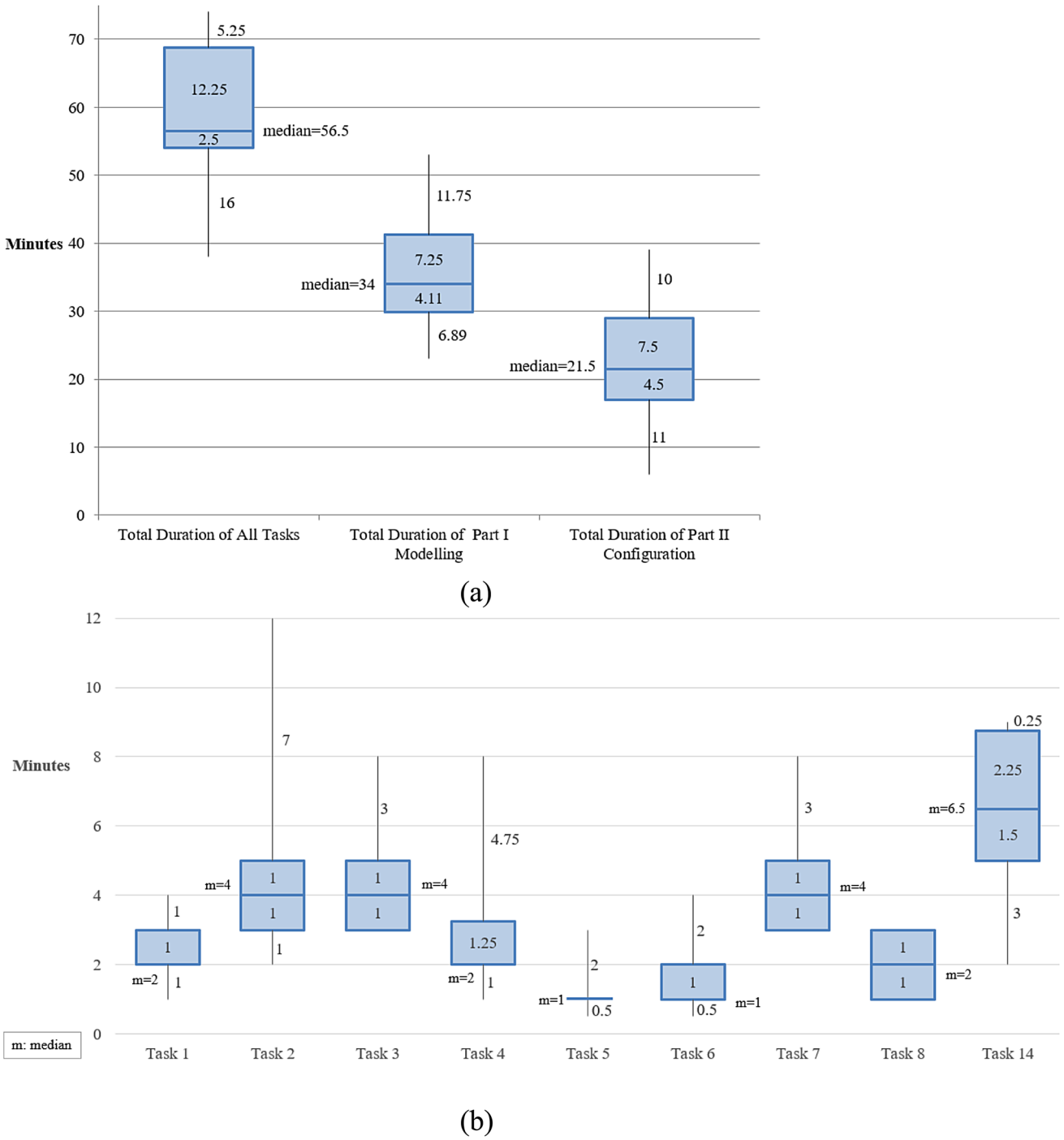

Fig. 9 Time distributions of tasks (**a**) and modelling tasks' duration (**b**)

to the participants of the task-oriented questionnaire that spent more time with the framework.

Participants from both evaluations agree that using a MDD approach through the UbiCARS framework to develop UbiCARS is a good idea. Furthermore, although results for both the perceived usefulness and perceived ease-of-use are positive, participants seem to value higher the *usefulness* of the framework. Across all questions and all participants, mean value for perceived usefulness is 5.94, while mean value for perceived ease-of-use is 5.66.

In terms of the UEQ, on a scale from 3 (highly positive) to −3 (highly negative), participants of the task-oriented evaluation found the framework to be enjoyable (2.25), pleasing (2.1), interesting (2.1) and exciting (2). Table 6 shows the

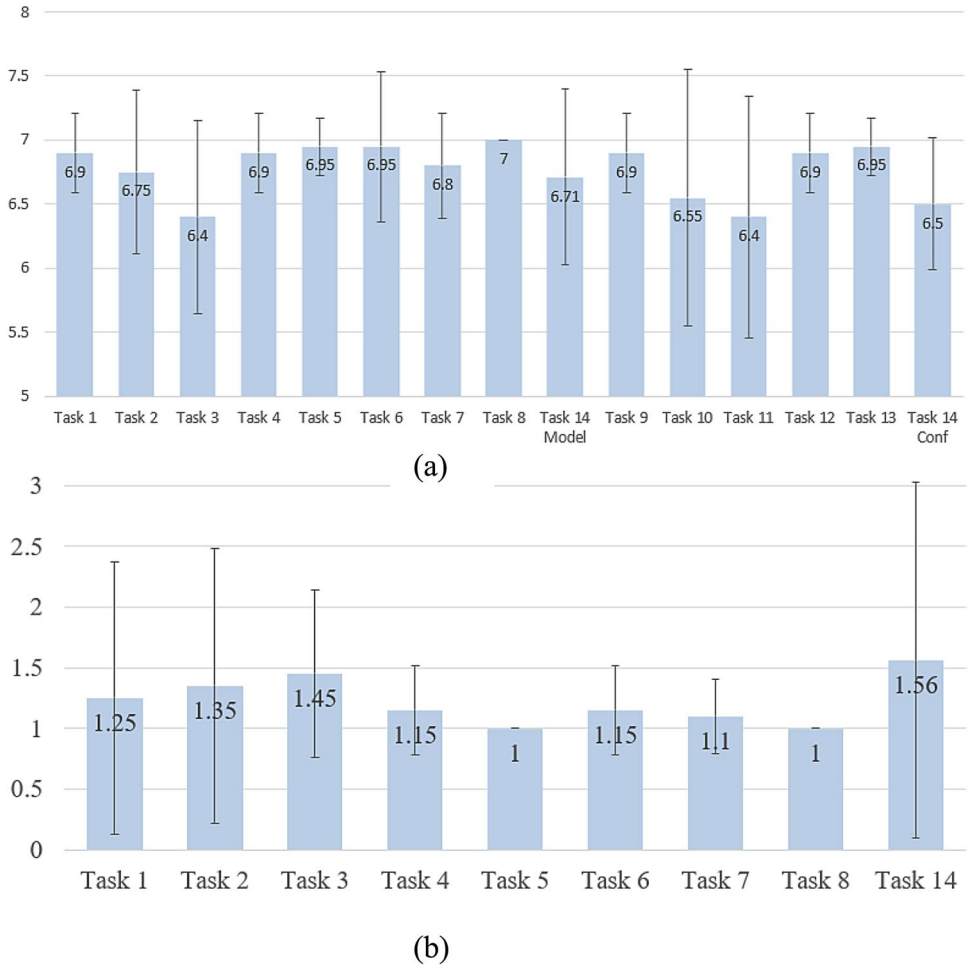

Fig. 10 Mean and Standard Deviation of participants' perceived comprehension of tasks and their results (**a**) and perceived difficulty for modelling tasks (**b**)

mean values for the remote evaluation, where participants replied for all the 26 items of the UEQ. The results are positive: all items but four have received mean scores of more than 1. The items with the highest scores are: good (1.73), interesting (1.69), fast (1.69) and understandable (1.56).

Tasks' Duration (for Task-Oriented Evaluation)

The time that participants took to complete each of the evaluation tasks, as well as the total time spent to deploy a UbiCARS is of particular importance as it is directly related to RQ1 on whether the framework reduces the development time and expedites the development of UbiCARS for commerce (see "Introduction" section). During the task-oriented evaluation process, the tasks' durations have been measured. Figure 9a depicts three time distributions: the distribution of the total duration of all tasks (modelling and configuration tasks), the distribution of the total duration of modelling tasks, and the distribution of the total duration of configuration tasks.

The numbers in the figure show the quartile ranges in minutes (the first quartile is the lower one). It can be seen that participants achieving the best results in the overall process (lower end of first quartile of left box plot) have comparable timings with participants achieving the worst results in the modelling tasks (upper end of the fourth quartile of middle box plot). Moreover, the boxplot for the modelling tasks has a similar distribution with the one for the configuration tasks and is located about 13 min higher in the minutes scale, which is an acceptable time difference: it suggests that the modelling tasks did not introduce considerable delays to participants, in comparison to the simpler configuration tasks. Figure 9b depicts the time distributions of the duration of each task for the tasks of the modelling part. Most of or all the participants were able to complete each task but task 14 in under 5 min. Task 2 lasted for 12 min for two participants; the remaining 18 participants where around the 5-min mark or lower. Task 14 is regarded as the most time-consuming task, as it required from participants to alter their model and re-execute the configuration tasks without instructions. Still, participants were able to complete the task in under 9 min, the median being at 6.5 min.

The following mean values should be noted: mean value for the total duration of all tasks was 59.16 min, mean value

for the duration of modelling tasks was 36.16 min and mean value for the duration of configuration tasks was 23 min. Thus, participants required a bit more than half an hour to complete the modelling tasks and less than that to complete the configuration tasks.

Perceived Task Comprehension and Difficulty (Task-Oriented Evaluation)

Task comprehension and perceived difficulty by participants during the task-oriented evaluation are examined in this paragraph as they are related to RQ2 on whether the framework reduces development complexity in developing UbiCARS. After completing each modelling task (Tasks 1–8 & Task 14) and configuration task (Tasks 9–13 and Task 14 that included executing the model designed in Task 14 of the modelling part), participants were requested to indicate their agreement with the statement *"I was able to understand the task"* (for modelling tasks) and *"I was able to understand the produced results"* (for configuration tasks), by using a 7-point Likert scale (1 for "Strongly Disagree" to 7 for "Strongly Agree"). Figure 10a shows the mean (numbers in the graph) and standard deviation (error lines) of the responses for each task. Participants strongly agree to have understood the tasks and their outcomes, indicating that the modelling framework was easy to adopt, understand and use.

After completing each modelling task, participants were also requested to indicate their agreement with the statement *"This task was difficult for me to accomplish"*, using the same 7-point Likert scale. For the configuration tasks this question would not apply due to the tasks' simplicity. Figure 10b shows the results. Participants disagree that the tasks were difficult for them to accomplish. The high Standard Deviations for Tasks 1, 2 and 14 are a result of a single negative response for each of these tasks. For Tasks 1 and 2, the negative response originated from the same participant, who responded positively for all other tasks.

Discussion

Based on the evaluation results, in this section we answer to the Research Questions stated in "Introduction" section.

RQ1: In the following paragraphs we discuss how the evaluation results indicate that the UbiCARS framework reduces the development time of UbiCARS.

Based on the results of the task-oriented evaluation, the framework enables the development of UbiCARS for commerce by developers that are non-experts in RS on average in 59.16 min. In cases where additional modules need to be developed by the developer, the results reported in the "Modelling Against Coding Comparison" evaluation ("Process Description" section) indicate an estimated time of 2 development hours for developing a context sensing module and 4 development hours for developing a browsing history user feedback module. It is important to note that the author involved in the implementations shown in Table 3 had knowledge on UbiCARS and the concepts of user feedback acquisition and context sensing. It is possible that developers lacking this knowledge would need more time to complete these implementations.

In the hypothetical scenario where a developer that participated in the task-oriented evaluation was asked to participate to the "Modelling against Coding comparison" scenario to implement a CARS using the method "modelling using supported e-stores", the estimated time would be one hour to use the modelling framework (according to the task-oriented evaluation), and about 2 additional hours to develop the context sensing module for the purchase history. This time is significantly less than the time the authors spent on developing the UbiCARS using the manual method, which was estimated to 44 h.

Based on the evaluation results, the framework is considered by participants to (i) enable them to develop UbiCARS more quickly: this statement refers to development time; and (ii) increase their productivity in developing UbiCARS: this statement refers to finishing tasks related to UbiCARS development more quickly or at a more rapid rate. RQ1 is validated.

RQ2: In the following paragraph we discuss how the evaluation results indicate that the UbiCARS framework reduces development complexity in developing UbiCARS by reducing the lines of code and database queries developers need to write.

The task-oriented evaluation proved that participants while being non-experts in RS, were able to complete all tasks successfully in less than one hour on average, having also understood all the tasks performed and their outcomes, supporting thus that the modelling framework was not complicated for them to use. In that scenario there was no need to write code, nor conduct database queries to accomplish this. During both evaluations with participants, developers and RS/MDD experts agreed that using the framework to develop UbiCARS would reduce their effort in terms of lines of code and database queries, since they observed that by using the DSML to create models and through the automated system configurations conducted by the framework, the required UbiCARS was created and configured in an automated manner. The evaluation "Modelling against Coding comparison" serves as an indication of how much development effort the framework would reduce for the respective scenario under examination. Based on the RQ2 is validated.

Table 4 Task description for the modelling and configuration parts of the evaluation

Task #	Number of sub-tasks	Aim of task
Modelling part		
1	3	Familiarize participants with modelling editor and toolbox by creating simple model elements for CARS and UbiCARS
2	6	Add elements in the model relevant to computation of recommendations, i.e. a recommendation engine and an algorithm
3	5	Add elements related to tracking of customers' browsing history on products in the e-store, i.e. customer accesses in product webpages
4	4	Add elements related to customers' "staying time in front of a product" as user feedback data on products from the physical store
5	2	Add elements for enabling the UbiCARS app to utilize customers' staying time in front of a product via Bluetooth beacons
6	2	Configure RSSI min Detection Threshold and Max Time Interval BSD parameters of Staying Time element
7	7	Complete model by adding more user-product interaction elements and database resources and link them together
8	2	Add a Configuration File element
14	4	(Without instructions to participants) Alter the model to include customers' location while staying in front of products as context, to enable context-aware recommendations
Configuration part		
9	–	Parse the model to configure the e-store and its database
10	–	Fill new database with data
11	–	Compile datasets and then compute recommendations
12	–	Store recommendations
13	–	Present recommendations
14	–	Execute the altered context-aware model to enable context-aware recommendations

Conclusions and Future Work

In this paper we have presented our work, a Model Driven Development Framework for Ubiquitous Context-Aware Recommender Systems that makes their development easier and faster. The evaluation results show the potential of the UbiCARS MDD Framework. Participants appreciated the abstraction of the technical details that the framework offers and liked the fact that they did not need to write code to develop UbiCARS, neither to learn any specifics about RS. The expedited development of UbiCARS that the framework offers, in comparison to any manual method that requires coding and comprehension by developers of Recommender Systems domain specifics, was a clear benefit mentioned by participants. In addition, during a direct comparison of the effort needed to develop a UbiCARS for an e-store using the framework against the manual method (coding), the authors report that using the framework together with its supported e-stores eliminates the need to write code, while using the framework with non-supported e-stores reduces the development effort in terms of lines of code and database queries by 84.4% and 86.7% respectively against the manual method (see Table 3). Time was estimated to have been reduced by 81.8%.

As stated in "Design Demonstration" section, user-product interaction data used during the evaluation stemmed from our limited interaction with the example e-stores and the physical store scenario, in which lab personnel interacted with the products. As such, these data are few and not of the required quality. In this sense, validating the claim for a potential improvement of recommendations' accuracy and availability was not possible due to the lack of real-world data. The recommendations currently provided by the framework should only be considered as a proof-of-concept. As future work, we aim to validate the claimed accuracy and availability improvement of the recommendations produced by the UbiCARS framework by using it in real world scenarios for an extended period of time. We plan to offer the framework for usage to interested e-commerce businesses that currently do not provide recommendations to their customers, and study whether the recommended products are indeed being selected by customers.

While recommendation explanations have been included in the DSML and can drive enablement or disablement of simple explanations on e-stores, explanations can be facilitated to a greater extend within the DSML, by abstracting the different explanation techniques and making them available in the model.

We aim to investigate whether recommendation algorithm specific details could be included in the DSML through new elements and parameters, while two critical requirements are met: (i) the abstraction of the DSML is maintained: such elements should facilitate a cross-recommendation algorithm/engine design, (ii) the complexity of the DSML is not increased, as too complex DSMLs result in designers not using them, or, in the best-case scenario, not using their entire potential.

In terms of users' privacy, the 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. RS, as systems functioning exclusively on users' data, are affected by the regulation. Examples of articles that may affect RS are:

- "Data Minimisation": affects preference elicitation processes of RS when they explicitly ask users for personal information or implicitly tracking their behaviour.
- The "Right to Erasure": affects RS performance in terms of recommendations' accuracy, as it reduces the amount of data upon which the recommendations are computed. Moreover, it enhances the cold start problem, as it may reduce the ratings of particular items that have few ratings in overall.
- The "Right to Rectification": in case of implicit user feedback techniques, should the system provide means for users to be able to update their data, i.e. be able to change data related with their interaction with the system? This would interfere with the recommendation process, rendering it inaccurate.

Based on the above discussion, future work will focus on the extension of the UbiCARS DSML with GDPR compliance enabling elements, aiming to guide the design and development of GDPR compliant UbiCARS. Since UbiCARS utilize user behaviour data from the e-store and the physical store, such data are inherently sensitive. As an example, we note that during indoor positioning, while location estimation software is run on the mobile client and not on the beacon, the data are nevertheless transferred to the CARS system and stored in the server's database to be exploited by the recommendation engine. Thus, privacy-preserving methods are required to ensure users' privacy.

Appendix

Pre-defined Tables in Supported E-stores

WordPress WooCommerce defines tables for storing user ratings and user purchase history. The WordPress database uses the "wp_" prefix for all its tables, which the user may change to any other prefix during installation. The framework facilitates three possible options: (i) "wp_" prefix is maintained (see `DatabaseResource` of `RATING` in Fig. 8), (ii) "wp_" is replaced with a user defined prefix: user provides the new prefix in the `DatabaseResource` element, and (iii) user changes the table names entirely: user provides in the `DatabaseResource` element the names of the new tables (single string comma-separated). The framework can automatically detect which of the three abovementioned options the user has selected and proceed accordingly.

Authors' Contributions CM: Conceptualization, Methodology, Investigation, Software, Writing—Original Draft. APA: Conceptualization, Validation, Writing—Review and Editing. GMK: Validation, Writing—Review and Editing. GAP: Methodology, Writing—Review and Editing, Supervision.

Funding Open access funding provided by the Cyprus Libraries Consortium (CLC). No funding was received for the preparation of this manuscript.

Data Availability The data supporting the findings of this study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Informed Consent Informed consent was obtained from all individual participants included in the study.

Research Involving Human and/or Animals Not Applicable.

References

1. Ricci F, Rokach L, Shapira B, Kantor PB. Recommender systems handbook. Springer; 2011. https://doi.org/10.1007/978-0-387-85820-3.
2. Mettouris C, Papadopoulos GA. Ubiquitous recommender systems. Computing. 2014;96(3):223–57. https://doi.org/10.1007/s00607-013-0351-z.
3. Walter FE, Battiston S, Yildirim M, Schweitzer F. Moving recommender systems from on-line commerce to retail stores. Inf Syst E-Bus Manag. 2012;10:367–93. https://doi.org/10.1007/s10257-011-0170-8.
4. Hu Y, Koren Y, Volinsky C. Collaborative filtering for implicit feedback datasets. In: Proceedings of the 8th IEEE international conference on data mining; 2008. pp. 263–72. https://doi.org/10.1109/ICDM.2008.22.
5. Fang B, Liao S, Xu K, Cheng H, Zhu C, Chen H. A novel mobile recommender system for indoor shopping. Expert Syst Appl. 2012;39(15):11992–2000. https://doi.org/10.1016/j.eswa.2012.03.038.
6. 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/. Accessed 02 Aug 2021.
7. International Council of Shopping Centers-ICSC. Shopping Centers: America's First and Foremost Marketplace. https://www.icsc.org/uploads/research/general/America-Marketplace.pdf. Accessed 25 July 2021.
8. Chen CC, Huang TC, Park JJ, Yen NY. Real-time smartphone sensing and recommendations towards context-awareness shopping. Multimedia Syst. 2015;21(1):61–72. https://doi.org/10.1007/s00530-013-0348-7.
9. So WT, 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; 2017. pp. 33:1–33:4. https://doi.org/10.1145/3092090.3092130.
10. Hussein T, Linder T, Gaulke W, Ziegler J. Hybreed: a software framework for developing context-aware hybrid recommender systems. User Model User-Adap Int. 2014;24(1–2):121–74. https://doi.org/10.1007/s11257-012-9134-z.
11. Inzunza S, Juárez-Ramírez R, Jiménez S. User modelling framework for context-aware recommender systems. In: Recent advances in information systems and technologies; 2017. pp. 899–908. https://doi.org/10.1007/978-3-319-56535-4_88.
12. Portugal I, Alencar P, Cowan D. The use of machine learning algorithms in recommender systems: a systematic review. Expert Syst Appl. 2018;97:205–27. https://doi.org/10.1016/j.eswa.2017.12.020.
13. Aldrich SE. Recommender systems in commercial use. AI Mag. 2011;32(3):28–34. https://doi.org/10.1609/aimag.v32i3.2368.
14. 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; 2009. pp. 252–63. https://doi.org/10.1007/978-3-642-03964-5_24.
15. Rodríguez-Hernández MC, Ilarri S. Pull-based recommendations in mobile environments. Comput Stand Interfaces. 2016;44(C):185–204. https://doi.org/10.1016/j.csi.2015.08.002.
16. White J, Schmidt DC, Czarnecki K, Wienands C, Lenz G, Wuchner E, Fiege L. Automated model-based configuration of enterprise Java applications. In: 11th IEEE international enterprise distributed object computing conference (EDOC 2007), Annapolis, MD. 2007; p. 301. https://doi.org/10.1109/EDOC.2007.22.
17. Achilleos AP, Kapitsaki GM, Constantinou E, Horn G, Papadopoulos GA. Business-oriented evaluation of the PaaSage platform. In: 2015 IEEE/ACM 8th international conference on utility and cloud computing (UCC); 2015. pp. 322–6. https://doi.org/10.1109/UCC.2015.51.
18. Mettouris C, Achilleos AP, Kapitsaki GM, Papadopoulos GA. The UbiCARS model-driven framework: automating development of recommender systems for commerce. In: Kameas A, Stathis K, editors. Ambient intelligence, AmI 2018; 2018. LNCS, vol. 11249. Cham: Springer; 2018. p. 37–53. https://doi.org/10.1007/978-3-030-03062-9_3.
19. Adomavicius G, Sankaranarayanan R, Sen S, Tuzhilin A. Incorporating contextual information in recommender systems using a multidimensional approach. ACM Trans Inf Syst TOIS. 2005;23:103–45. https://doi.org/10.1145/1055709.1055714.
20. 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; 2010. pp. 79–86. https://doi.org/10.1145/1864708.1864727.
21. Nguyen TV, 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; 2014. pp. 63–72. https://doi.org/10.1145/2600428.2609623.
22. Shafiee S. Unveiling the latest trends and advancements in machine learning algorithms for recommender systems: a literature review. Procedia CIRP. 2024;121:115–20. https://doi.org/10.1016/j.procir.2023.08.062.
23. Raza S, Rahman M, Kamawal S, Toroghi A, Raval A, Navah F, Kazemeini A. A comprehensive review of recommender systems: transitioning from theory to practice. 2025. arXiv preprint arXiv:2407.13699.
24. Mateos P, Bellogín A. A systematic literature review of recent advances on context-aware recommender systems. Artif Intell Rev. 2024;58(1):1–53. https://doi.org/10.1007/s10462-024-10939-4.
25. 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; 2016. pp. 1–12. https://doi.org/10.4204/EPTCS.233.1.
26. 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; 2012. pp. 639–40. https://doi.org/10.1145/2187980.2188166.
27. Yi X, Hong L, Zhong E, Liu NN, 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; 2014. pp. 113–20. https://doi.org/10.1145/2645710.2645724.
28. Oard DW, Kim J. Implicit feedback for recommender systems. In: Proceedings of the AAAI workshop on recommender systems; 1998. vol. 83. WoUongong.
29. Sulikowski P, Zdziebko T. Horizontal vs. vertical recommendation zones evaluation using behavior tracking. Appl Sci. 2021;11:56. https://doi.org/10.3390/app11010056.
30. Sulikowski P, Zdziebko T, Turzynski D, Kantoch E. Human-website interaction monitoring in recommender systems. Procedia Comput Sci. 2018;126:1587–96. https://doi.org/10.1016/j.procs.2018.08.132.
31. Fahim Shahriar ABM, Zaman Moon M, Mahmud H, Hasan K. Online product recommendation system by using eye gaze data. In: Proceedings of the international conference on computing advancements (ICCA 2020). Association for Computing Machinery, Dhaka, Bangladesh, 10–12 January 2020; Article 61; pp. 1–7. https://doi.org/10.1145/3377049.3377108.

32. Atashkar M, Safi-Esfahani F. Item-based recommender systems applying social-economic indicators. SN Comput Sci. 2020;1:113. https://doi.org/10.1007/s42979-020-0115-8.
33. Necula SC, Păvăloaia VD. AI-driven recommendations: a systematic review of the state of the art in E-commerce. Appl Sci. 2023:13, Article 5531. https://doi.org/10.3390/app13095531.
34. Wang Y, Li Z, Zhang C, Chen S, Zhang X, Xu J, Lin Q. Do not wait: learning re-ranking model without user feedback at serving time in E-commerce. In: Proceedings of the 18th ACM conference on recommender systems (RecSys '24). Association for Computing Machinery, New York, NY, USA; 2024. pp. 896–901. https://doi.org/10.1145/3640457.3688165.
35. Hwangbo H, Kim YS, Cha KJ. Recommendation system development for fashion retail e-commerce. Electron Commer Res Appl. 2018;28:94–101. https://doi.org/10.1016/j.elerap.2018.01.012.
36. Hasan E, Rahman M, Ding C, Huang JX, Raza S. Review-based recommender systems: a survey of approaches, challenges and future perspectives. 2024. arXiv preprint arXiv:2405.05562.
37. Anindya G, Li B, Liu S. Mobile advertising using customer movement patterns. Doctoral dissertation, Working paper, New York University, New York; 2015. https://doi.org/10.1287/mnsc.2018.3188.
38. Bell DR, Corsten D, Knox G. From point of purchase to path to purchase: how preshopping factors drive unplanned buying. J Mark. 2011;75(1):31–45.
39. Granbois DH. Improving the study of customer in-store behavior. J Mark. 1968;32(4):28–33. https://doi.org/10.2307/1249334.
40. Jeffrey IJ, Winer RS, Ferraro R. The interplay between category characteristics, customer characteristics and customer activities on in-store decision making. J Mark. 2009;73:19–29. https://doi.org/10.1509/jmkg.73.5.
41. Jie C, Dong W, Canquan L. Recommendation system technologies of intelligent large-scale shopping mall. In: Proceedings of 2nd international conference on computer science and network technology, Changchun; 2012. pp. 1058–62. https://doi.org/10.1109/ICCSNT.2012.6526108.
42. Zimmermann R, Mora D, Cirqueira D, Helfert M, Bezbradica M, Werth D, Weitzl WJ, Riedl R, Auinger A. Enhancing brick-and-mortar store shopping experience with an augmented reality shopping assistant application using personalized recommendations and explainable artificial intelligence. J Res Interact Mark. 2023;17(2):273–98. https://doi.org/10.1108/JRIM-09-2021-0237.
43. 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.
44. Pfeiffer J, Pfeiffer T, Meißner M. Towards attentive in-store recommender systems. In: Reshaping society through analytics, collaboration, and decision support. 2015; vol. 18. https://doi.org/10.1007/978-3-319-11575-7_11.
45. Reischach FV, Michahelles F, Schmidt A. The design space of ubiquitous product recommendation systems. In: Proceedings of the 8th international conference on mobile and ubiquitous multimedia; 2009. pp. 1–10. https://doi.org/10.1145/1658550.1658552.
46. Reischach FV, Michahelles F. Apriori: a ubiquitous product rating system. In: PERMID '08, workshop on pervasive mobile interaction devices at pervasive conference; 2008.
47. Mora D, Jain S, Nalbach O, Werth D. An in-store recommender system leveraging the Microsoft HoloLens. In: HCI International 2020—posters. Springer, Cham; 2020. pp. 99–107. https://doi.org/10.1007/978-3-030-50729-9_14.
48. Dadouchi C, Agard B, Montreuil B. Context-aware interactive knowledge-based recommendation. SN Comput Sci. 2022;3:472. https://doi.org/10.1007/s42979-022-01328-1.
49. Barbosa NM, Wang G, Ur B, Wang Y. Who am I? A design probe exploring real-time transparency about online and offline user profiling underlying targeted ads. Proc ACM Interact Mob Wearable Ubiquitous Technol 5(3), Article 88 (2021). https://doi.org/10.1145/3478122.
50. Saad S, Inayat I. Model-driven development based cross-platform development: a review. J Inf Sci Eng. 2017;33(6):1561–73. https://doi.org/10.6688/JISE.2017.33.6.11.
51. Sosa-Reyna CM, Tello-Leal E, Lara-Alabazares D. Methodology for the model-driven development of service oriented IoT applications. J Syst Archit. 2018;90:15–22. https://doi.org/10.1016/j.sysarc.2018.08.008.
52. Zheng Y, Mobasher B, Burke R. CARSKit: a Java-based context-aware recommendation engine. In: 2015 IEEE international conference on data mining workshop (ICDMW); 2015. pp. 1668–71. https://doi.org/10.1109/ICDMW.2015.222.
53. Amyot D, Farah H, Roy JF. Evaluation of development tools for domain-specific modeling languages. In: System analysis and modeling: language profiles; 2006. Pp. 183–97. https://doi.org/10.1007/11951148_12.
54. Tintarev N, Masthoff J. Evaluating the effectiveness of explanations for recommender systems. User Model User-Adap Int. 2012;22:399–439. https://doi.org/10.1007/s11257-011-9117-5.
55. 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, editors. Ambient intelligence, AmI 2017; 2017. LNCS, vol. 10217. Springer; 2017. https://doi.org/10.1007/978-3-319-56997-0_5.
56. Huffingtonpost. How Bluetooth beacons will transform retail in 2016. https://www.huffingtonpost.com/kenny-kline/how-bluetooth-beacons-wil_b_8982720.html. Accessed 28 Nov 2020.
57. Mohagheghi P. An approach for empirical evaluation of model-driven engineering in multiple dimensions. In: Proceedings of C2M:EEMDD 2010 workshop at ECMFA 2010-from code centric to model centric: evaluating the effectiveness of MDD; 2010. pp. 6–17.
58. Davis F. Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. 1989;13(3):318–39. https://doi.org/10.2307/249008.
59. Heijden H. User acceptance of hedonic information systems. MIS Q. 2004;28(4):695–704. https://doi.org/10.2307/25148660.
60. Schrepp M. User experience questionnaire handbook; 2015. https://doi.org/10.13140/RG.2.1.2815.0245.
61. Laugwitz B, Held T, Schrepp M. Construction and evaluation of a user experience questionnaire. In: Holzinger A, editor. HCI and usability for education and work. USAB; 2008. LNCS, vol. 5298. Berlin: Springer; 2008. https://doi.org/10.1007/978-3-540-89350-9_6.