Designing Context Models for CARS Incorporating Partially Observable Context

Christos Mettouris(✉) and George A. Papadopoulos

Department of Computer Science, University of Cyprus,
1 University Avenue, Nicosia, Cyprus
{mettour,george}@cs.ucy.ac.cy

Abstract. Context modelling and context reasoning are widely used topics in Context-Aware Recommender Systems research. Based on our research, the development of context models in context-aware recommender systems is problematic in that many domain specific and application specific context models are developed with limited or no reuse and sharing capabilities. Furthermore, context-aware recommender systems that follow the representational view of context, design and model the fully observable context that is known at recommendation time but do not consider partially observable context with unknown values at recommendation time, that can nevertheless enhance the recommendation outcome. In this paper we deal with the above two issues by proposing a CARS design system that enables developers: (i) to easily and effectively design context models by defining, sharing and reusing context parameters and (ii) to utilize partially observable context at recommendation time by using an interactional approach that incorporates user feedback by applying a utility based algorithm on context models.

Keywords: Context modelling · Context model design · Context-Aware recommender systems · Interactive algorithm · Partially observable context

1 Introduction

Traditional Recommender Systems (RS) use limited or none contextual information to produce recommendations, as opposed to Context-Aware Recommender Systems (CARS) that focus in using contextual information to enhance recommendations. Context was first used within recommendation algorithms and methods by Adomavicius by proposing three approaches: the Pre-filtering approach, the Post-filtering approach and the Multi-dimensional Contextual Modelling approach [1, 2].

A contextual design issue related to CARS is the development of domain specific and application specific context models that only represent information on the particular application domain (e.g. recommendation of movies) or information regarding a particular application. The main problem with domain and application specific models is overspecialization, as well as limited or no reuse.

On another dimension, whether the contextual information being used by a context-aware recommender system is known during the recommendation characterizes the context as fully observable, partially observable, or unobservable [4, 5]. Fully observable context refers to all contextual parameters, their structure and values being known during recommendation, partially observable refers to only part of this information being known, while unobservable context refers to the case where the context is unknown at recommendation time. Following a representational view of context in CARS design [6, 7], the context is defined through a predefined set of observable context parameters of static structure which does not change significantly over time, as opposed to the interactional view of context where the context is often unknown as it may not necessarily be an observable feature of an interaction. From the above it is evident that a representational view of context can be achieved only if the context is fully observable, i.e. its parameters and their values are known at recommendation time (can be observed, detected and retrieved) and thus the designer can include such information in context models during design. On the other hand, if not all context information is fully observable, an interactional view of context can be considered, where specific information may or may not be relevant to some activity and contextual parameters may be defined in a dynamic and occasioned manner rather than a static, predefined one [6]. We argue that the most common practice in designing CARS is to follow the representational view of context, during which CARS developers attempt to utilize the fully observable context within their recommenders by using contextual parameters known at recommendation time, which may result in omitting unknown contextual information that nevertheless may be important.

In this paper we build upon prior work [8] to address the aforementioned contextual design problems. A Context-Aware Recommender System design system is proposed which developers can use to: (i) design and build context models at the application layer for their recommender applications that incorporate both fully observable and partially observable context, (ii) apply recommendation algorithms that utilize these context models by facilitating an interactional approach for partially observable context that incorporates user feedback. Regarding designing and building context models (i), the system guides developers towards an easy, efficient and effective selection and usage of context parameters, allowing at the same time for sharing and reuse of context models among recommender applications, regardless of the domain they belong to. On applying recommendation algorithms on context models (ii), the system incorporates a method that facilitates both fully observable and partially observable context. The system supports the developer in specifying system actions using pseudo code to handle the fully observable context, while it relies on user feedback to calculate the utility of a set of possible context values for the partially observable context and recommends the most suitable context value to be used by the system. In this work we will refer to context information that is not fully observable at recommendation time (i.e. its values are not known and cannot be retrieved at the time of recommendation, although some information on the structure of the context may exists) as partially observable context. We will also refer to partially observable context with unknown values as "unknown context", as opposed to fully observable context with values known at recommendation time. The CARS design system was developed as an online web-based system using PHP, MySQL and web technologies[1].

[1] Online: http://www.cs.ucy.ac.cy/~mettour/phd/CARSContextModellingSystemV3/

Section 2 provides related work. Section 3 describes the context modelling design process and system actions on models. Section 4 describes the methodology used in known as well as unknown context settings. In Sect. 5 we talk about our experimentation procedure and the paper closes with conclusions and future work in Sect. 6.

2 Related Work

We have reviewed a number of context-aware recommender systems in the bibliography that use contextual and conceptual models [3, 8]. Although contextual models in the bibliography exist that attempt to facilitate a more generic and cross-domain design [9–12], the majority represent information that either concern particular application domains (e.g. tourism, movies, museums), or more abstracted domains (e.g. products in general, web services, e-learning, etc.).

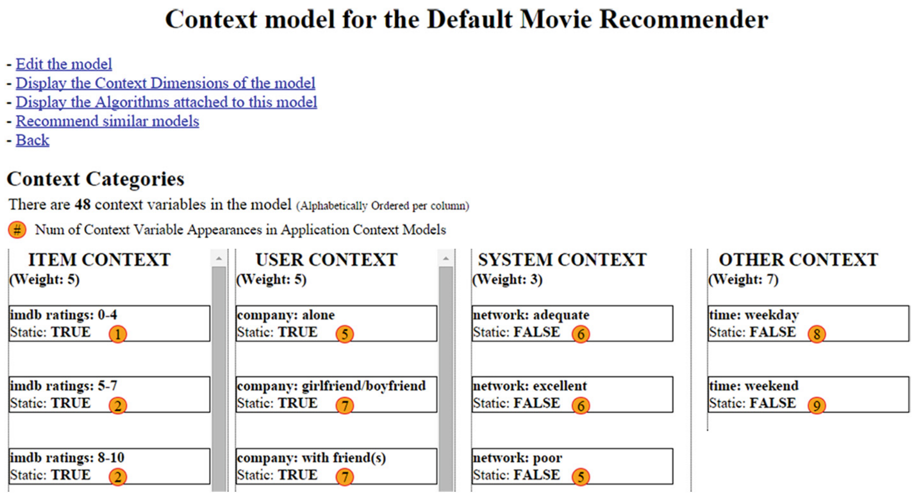
Moreover, related works from the fields of Machine Learning, Data Mining and Information Retrieval incorporate contextual information within their modelling methods to enhance prediction accuracy [13–15]. While these context modelling methods unquestionably enhance recommendation accuracy, their mathematical oriented models require a good understanding of the aforementioned fields by the developer, as well as strong development skills to be implemented.

To the best of our knowledge a context design system that could facilitate the development of reusable and generic contextual models for CARS has not yet been proposed in the literature. The aim of such a system would be to facilitate context model design, model reuse among developers and applications, and model extension/update based on the needs of the developer. Such a tool would simplify the process of contextual modelling in CARS and enable context uniformity, share and reuse.

Furthermore, most of the examined systems in related work follow the representational view of context in which the fully observable context is designed in a particular structure (e.g. using semantic models, ontologies, etc.). In these cases, context parameters known at recommendation time are being modelled. We argue that this practice may not consider important contextual information with unknown values during recommendation that may be nevertheless important for the recommendation outcome. In this paper we implement an interactional approach for including partially observable context in the recommendation algorithms by incorporating user feedback.

3 Context Model Design

All context models in the CARS design system follow the design presented in the figure on the main page of the system (See Footnote 1). From top-to-bottom, the level of abstraction decreases. The "user_item_context_rating" entity represents a single complete recommendation process [1]. For each recommendation run, a recommender examines the utility of each item for a particular user in a certain context. The context variable entity specifies a contextual parameter and its value in a context model: name:value, e.g. Temperature:high. The weight property denotes the importance of a context variable. The static property refers to whether the context variable is static (cannot change dynamically, e.g. user's date of birth) or dynamic (can change, e.g. weather). A context variable may be part of the known or the unknown context. Since the context in CARS is multidimensional [1, 2], the system is able to handle the many dimensions of the context in a context model in an easy and scalable manner: we define that each unique context variable name with all its values represents a unique context dimension. A context model in the system is defined as one or more context variables, each context variable being assigned under one or more of the four context categories (Fig. 1): itemContext, userContext, systemContext and otherContext.



Context model for the Default Movie Recommender

- Edit the model
- Display the Context Dimensions of the model
- Display the Algorithms attached to this model
- Recommend similar models
- Back

Context Categories

There are **48** context variables in the model (Alphabetically Ordered per column)

Num of Context Variable Appearances in Application Context Models

ITEM CONTEXT (Weight: 5)	USER CONTEXT (Weight: 5)	SYSTEM CONTEXT (Weight: 3)	OTHER CONTEXT (Weight: 7)
imdb ratings: 0-4 Static: **TRUE** 1	**company: alone** Static: **TRUE** 5	**network: adequate** Static: **FALSE** 6	**time: weekday** Static: **FALSE** 8
imdb ratings: 5-7 Static: **TRUE** 2	**company: girlfriend/boyfriend** Static: **TRUE** 7	**network: excellent** Static: **FALSE** 6	**time: weekend** Static: **FALSE** 9
imdb ratings: 8-10 Static: **TRUE** 2	**company: with friend(s)** Static: **TRUE** 7	**network: poor** Static: **FALSE** 5	

Fig. 1. Application context model for the "Default Movie Recommender"

Three context models are defined in the system:

The *Generic Model* includes all context variables currently defined in the system.

The *Application Context* is a context model for a particular RS, e.g. a movies RS. This is a model built by a CARS developer to model the context for a RS. An application context model is built using a system interface via which all available context variables in the system are presented on screen and the CARS developer clicks on those she wants to select for her context model, or adds new context variables. It is very likely that context variables predefined by other developers are suitable for the new application context as well, which not only saves time to the developer, but more importantly it enables the developer to use context parameters that she may not have thought of using, or may think is unable to use in case the particular context is unknown at recommendation time. In the CARS design system, unknown context can be defined in the form of context variables in the same manner as known context. Note that, since each context variable has a name and a specific value (e.g. Temperature: high), the developer must select all needed variables with a particular name for the model to be accurate and complete, e.g. all of the following: Temperature:high, Temperature:medium, Temperature:low. In this manner, the context dimension Temperature is also defined in the application context model.

The *Context Instance* context model is a "screenshot" of the context during an interaction between the user and the item involved in the recommendation process. For example, for a movie recommender, a context instance is the set of context variables that constitute the context during the event of a particular user (user = "Jerry") watching a particular movie (item = "Kill Bill") at a particular time (we assume the first time that Jerry watches Kill Bill). Such a context instance may have title: "Jerry_-KillBill_C1" (C1 results from "Context1") and can be consisted of a number of context variables and dimensions, e.g.: time_of_day, day_of_week, movie_IMDB_ratings, companion_of_user (whom did the user watch the movie with), etc. In a similar way, the context instance around the second time Jerry watches Kill Bill will have a title "Jerry_KillBill_C2" and will again be consisted of a number of context variables and dimensions.

Any context model in the system is accessible and can be shared, edited, extended and reused by others. The general idea behind this concept is that, since all RS of a specific type/domain (e.g. online movie recommenders) function in similar context settings, having one context model for each recommender built by each developer may be inefficient; we propose the definition of one complete context model for all recommenders.

Supported System Actions
Building Models: The system offers to developers pre-existing application context models of similar applications to use as a baseline for their own models, instead of building a model of their own.

Validating Models: The system supports the validation of an application context model through one or more context instance models. An application context model should be able to support any context instance related to the particular recommender; otherwise, the application context model is incomplete. For example, an application context model for a movie recommender should be able to support any movie recommender related context instance, such as "Jerry_KillBill_C1". Application context model validation is a useful tool for the CARS developer to validate her application context model against a number of context instances and thus ensure that her application context is able to properly model the context (i.e. model any context instance of the recommender). A context instance can be created preferably by a CARS user who will reflect her experiences regarding the activity of watching movies in the context instance model. If this is not feasible, the context instance model can be created by another developer. The system provides information whether a context instance model was validated against the application context or not and justifications.

Context Dimensions View: The system supports a context dimensions view for each context model in the system.

Recommendation of Models: The system is able to recommend the top N (currently N = 5) most similar context models to a particular application context model. The comparison is based on the percentage of common context variables and context dimensions. This informs the developer about similar context models as well as the level of similarity. The recommendations are provided both explicitly and implicitly.

Model Comparison: Application context model comparison is supported. Though an easy to use user interface the system depicts common context variables, as well as those that belong only to one of the two application contexts. Colours are used on screen for easier comprehension of the information. The system also provides statistics regarding the percentage of participation of each application context model to the other. If many common variables are noted, the system proposes a merge of the models. This is especially interesting in cases where application context models concern RS of different domains, e.g. a movie RS and a book RS: the systems, although of different domains, use similar contextual information.

4 Assigning Algorithms to Context Models

In order to handle the known context the system supports the developer in specifying system actions using pseudo code. For unknown context however, the system relies on user feedback to calculate the utility of a set of possible context values (context variable candidates) and then recommends the context value with the highest utility to the CARS developer in order to use it as the best context setting in subsequent system recommendations to the user. This context value is considered as the best context candidate for the unknown context to be used in the recommendation process.

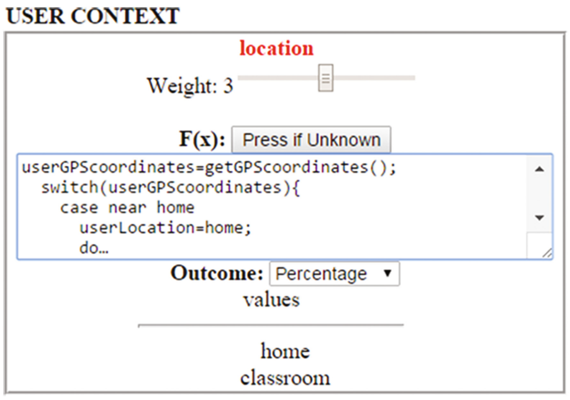
We have implemented an algorithm as a prototype, which is based on a utility scoring method. Given an application context model (as specified in Sect. 3), an end user u and an item j, the system uses context utility function - CUF (1) to measure the utility of item j for user u in relation to each context dimension in the model:

$$\mathrm{item\_util}_{\mathrm{ju}} = \mathrm{f}_1 * \mathrm{W}_1 + \ldots + \mathrm{f}_\mathrm{i} * \mathrm{W}_\mathrm{i} + \ldots + \mathrm{f}_\mathrm{N} * \mathrm{W}_\mathrm{N} \quad (1)$$

$\mathrm{item\_util}_{\mathrm{ju}}$ is the context utility of item j for user u. It specifies how much the particular context setting satisfies user u when recommended item j. Items with the highest utility are recommended to the user. N is the number of context dimensions in an application model. f_i*W_i represents context dimension i. f_i is the description of context dimension i for user u and item j (in computable format as explained in Sect. 4.1). W_i is the weight for context dimension i: denotes the importance of the context dimension in relation to the other context dimensions (an integer, values 1–5). The weights are defined by the developer based on the importance she would like to denote for each context dimension – different applications may require different weights on context dimensions such as location, temperature, etc.

4.1 Specifying Functionality on Fully Observable Context Parameters

To facilitate the CARS developer in incorporating a context model within a recommendation process, we provide a simple interface by which she can use pseudo code to describe the required functionality to handle the context (Fig. 2). More particularly, for each context dimension in an application context model (i.e. for each f_i, i = 1…N in (1)), the system supports description of the required functionality to handle the particular context dimension. The description is currently text based and has no effect in system decisions on context; this is planned for future work. Using pseudo code to describe the required functionality on how to handle the context is preferred than using linguistic text based description as it is more descriptive, it is easier for a developer to understand what is needed to be done precisely (especially when the developer is someone other than the context model designer), it avoids ambiguity and it can easier facilitate a system language definition in the future.



```
userGPScoordinates=getGPScoordinates();
  switch(userGPScoordinates){
    case near home
      userLocation=home;
      do…
```

Fig. 2. System interface: CARS developer describes functionality for "location"

The above procedure is feasible for context parameters known at recommendation time (fully observable context), such as the location of the user. As an example suppose the design of a restaurant RS that recommends restaurants nearby the user, hence user location is important. User location can be detected by using GPS coordinates from the user mobile phone. Normally, the developer would specify in her application context model a single context parameter for location, e.g. user location; in our system, she can define context variable name/value pairs so that the context is more descriptive and less ambiguous. Let's suppose she has defined three context variables as one context dimension in her application model: context variables: location:home, location:work, location:other, context dimension: location. In this setting the developer denotes that user location at home and user location at work constitutes important context and should be handled explicitly. On the contrary, user location in other cases other than home or work are not equally important and can be handled collectively. The developer threfore needs to define the functionality for the context dimension location (noted as $f_{location}$ below) and specify the system functionality for each of the three possible context values, e.g.:

```
Function: f_location
  userGPScoordinates=getGPScoordinates();
  switch(userGPScoordinates){
    case near home
      userLocation=home; do…
    case near work
      userLocation=work; do…
    default
      userLocation=GPScoordinates;  do…}
```

The above code is clear as to what needs to be done: do specific tasks if user is at home or work, or another task if somewhere else. Note that the system requests from the developer a functionality description for the context dimension $f_{location}$ instead for each of the context variables (e.g. $f_{location:home}$) for simplicity and to avoid repetition.

Describing system functionality as in the above example is valuable when the CARS developer does not want to deal with implementation details at that moment, or has no full knowledge of the platform requirements. Moreover, omitting implementation details that would otherwise be essential for software development saves time and enables the developer not to be precise, as she might not yet be fully aware of other context parameters, their values and their role in the recommendation process. Moreover, it can be the case that the CARS designer is not a software developer and thus cannot code; in such cases ordinary text can be used.

Applying the Context Utility Function. If all context dimensions are known at recommendation time, then Eq. (1) is straightforward: the utility of an item equals to the sum of the context dimension descriptions multiplied by their weights. Of course, context dimension descriptions must return *computable values* so that the total utility of the CUF for each item can be computed. For example, the CUF for the restaurant RS above can utilize context dimension location as follows (the following code is to be appended in function $f_{location}$):

```
For each restaurant in Database Do:
  if dist(restaurant,userLocation)<2km
    return 0.6;
  elseif dist(restaurant,userLocation)>3km
    return 0.1;
  else
    return 0.3;
```

The return values of context dimension descriptions are expressed as the percentage item j fits the particular context dimension for user u. In the above example it is assumed that a walking distance of 2 km is the maximum convenient walking distance to a restaurant for the average user ($f_{location}$=0.6). The utility item_util of each item is computed and the items with the highest utility are recommended to the user.

A variety of CUFs can be specified for each application context model, each of which will differ in context dimension descriptions f_i and/or the assigned weights W_i. These context utility functions can be implemented and incorporated in the RS. The variety of CUFs produced based on the above process are stored in the system and can be shared and reused among CARS developers regardless of application domain.

4.2 Specifying Functionality on Partially Observable Context Parameters

Not all contextual information is known at recommendation time. An example is the context parameter user companion that specifies with whom the user is at a particular time. This contextual information may be important for a movie RS (recommendations on what movie to watch heavily depend on whether the user is with her partner, a friend, a family member or alone), a restaurant RS (a partner's gastronomic preferences can be as important as the user's if they are dining together), etc. User companion cannot be known to the system unless the user explicitly states it or in cases where location information of the person being with the user is known.

To the best of our knowledge in situations of partially observable or unobservable context, the case with systems following a representational view of context is that such context is not defined or used in their context models. On the other hand, systems following an interactional view of context can better cope with unobservable or partially observable context as context may be defined in a more dynamic manner [5].

The interface of the CARS design system includes a button *"Press if Unknown"* in the case of an unknown context dimension such as user companion (Fig. 2). When pressed, it is specified that the particular context dimension is unknown at recommendation time and that user feedback should be utilized for its value to become known.

Applying the Context Utility Function. Suppose that unknown context "user companion" is defined within an application context model as follows: context variables: user_companion:partner, user_companion:friend, user_companion:familyMember; context dimension: user_companion. While a context dimension may be currently unknown, the context-aware recommender system could have utilized this contextual information within the recommendation process, had it been known. For instance, there are no technological means to acquire the value of the context dimension user_companion, but had there be any; the CARS developer would have been able to exploit this information to enrich the recommendation outcome. In the case of context dimension "user companion", the profile of the user's companion u′ can be used in combination with the profile of user u to produce recommendations. This provides added value to the outcomes of a movie RS, a restaurant RS, as well as RS of various domains where user companion is important. For each context variable candidate of an unknown context dimension $f_{unknown}$ the CARS developer is expected to define and implement the desired functionality so that it returns a computable value, as stated in Sect. 4.1. In this manner, each one of the three context variables of context dimension "user companion" can participate in (1) as a context variable candidate of $f_{user\_companion\_unknown}$.

Given a CUF of an application context model (active CUF) and a user u, we define as *Context Utility Function run* (CUF run) the following steps:

i. the computation of $item\_util_{ju}$ for each item j using Eq. (1)
ii. the recommendation of the item(s) with the highest utility score to user u
iii. user u provides feedback on the recommended item, in the form of a rating score, integer from 1 (min) to 10 (max)

Considering an unknown context dimension $f_{unknown}$, in step i. above the available context variable candidates are used as possible values for $f_{unknown}$. One candidate is used for each CUF run, initially in a random manner. After user u provides feedback in step iii, this rating score is used as an evaluation metric for the context variable candidate. In this manner the system evaluates each context variable candidate of an unknown context dimension based on the received user feedback.

4.3 Recommendation of Unknown Context

The system uses a prediction algorithm that recommends to the CARS developer the best context variable candidate for an unknown context dimension to be used in subsequent CUF runs. This method retrieves the user feedback for each CUF run and calculates a score for the context variable candidate. After many CUF runs, the system is able to recommend the context variable candidate with the highest score. The prediction algorithm also considers:

- CUF runs of other CUFs of the same application model: other CUFs may have been defined to facilitate other situations
- CUF runs of CUFs of other application context models

Note that, for using other CUFs than the active one it is necessary that these CUFs include the unknown context variable candidate that is being evaluated as well.

5 Experimental Evaluation

We have evaluated our system by using partially observable context parameters and simulating CUF runs and user feedback.

Experimental Assumptions. For simplicity reasons we have selected an application context model with 1 unknown and 2 known context dimensions: assume an online movie RS that recommends movies to users based on their profile preferences and the context parameters: network capabilities (nc), time of day (td) and user companion (uc). Let the network capabilities and time of day be the 2 known context dimensions, while user companion is unknown. The CUF for item j is:

$$\mathrm{item\_util}_{ju} = f_{nc} * W_{nc} + f_{td} * W_{td} + f_{uc\_unknown} * W_{uc}$$

For simplicity reasons we let $W_{nc} = W_{td} = W_{uc} = 1$. The descriptions of the two known context dimensions f_{nc} and f_{td} have been defined in the system in pseudo code in a similar manner as the location was in Sect. 4.1. We assume these context factors are computable and hence the utility of the recommended item j for user u concerns only the unknown context:

$$\mathrm{item\_util}_{ju} = [\text{some computable values}] + f_{uc\_unknown} \tag{2}$$

Assume that $f_{uc\_unknown}$ has 7 context variables: user_companion:partner, user_companion:closeFriend, user_companion:socialFriend, user_companion:familyMember, user_companion:colleague, user_companion:otherPerson and user_companion:none. For each of the above context variables the CARS developer defines appropriate functionality in her system so that when a context variable is selected as a candidate for $f_{uc\_unknown}$, the corresponding functionality is executed and a computable value is returned to be used in (1) that depicts the utility of the item regarding the particular context dimension. For instance, if "user_companion:partner" is selected as a candidate, provided that the partner is a user with a profile known to the system, the recommender can combine both user profiles into one extended user profile and calculate the utility of each restaurant for the extended user profile. Similarly, the functionality for the other context variables can be defined.

Based on (2) the following assumption is valid (user feedback assumption): the user feedback on the recommended item j (item with highest item_util_{ju}) corresponds to the current context variable candidate used for the unknown context dimension $f_{\text{uc\_unknown}}$. In this manner, user feedback score evaluates context variable candidates, enabling the system subsequently to recommend to the developer the candidate with the highest score.

Running Simulations. We have used a program to simulate CUF runs in order to show that the prediction algorithm is able to discover the preferred context variable and recommend it to the CARS developer to use in subsequent CUF runs. CUF runs are simulated by assigning user feedback scoring on unknown context variable candidates (user feedback assumption and (2)). User feedback reflects user preferences via an integer score from 1–10.

Consider a user that watches movies online and uses the online movie RS to receive movie recommendations. We assume that most of the times the user is with her partner (user_companion:partner) and hence this context setting is often favourable, otherwise a random companion (1 of the 6 remaining context variables). We have defined the following probabilities for this user:

where $P_f(x)$: the probability the user submits feedback score x.

```
if $f_{uc\_unknown}$ equals user_companion:partner
  $P_f(1)=P_f(2)=P_f(3)=P_f(4)=0.05$;
  $P_f(5)=P_f(6)=P_f(7)=P_f(8)=P_f(9)=P_f(10)=0.15$;
else
  $P_f(1)=P_f(2)=P_f(3)=P_f(4)=P_f(5)=P_f(6)=P_f(7)=P_f(8)=P_f(9)=P_f(10)=0.1$;
```

The rationale is that, when the context-aware recommender system randomly selects to use $f_{\text{uc\_unknown}}$ = user_companion:partner, the user is more likely to rate higher the recommended movie (selecting a score ranging from 5 to 10) than otherwise. This happens since the particular context variable fits the user's real context, as opposed to any other context variable (since they watch movies together), which will positively affect the recommendation outcome in recommending more suitable movies. Based on various tests with similar probability values and to maintain our argumentation on logical assumptions, we argue that a probability of 15 % instead of 10 % for a high score and a probability of 5 % instead of 10 % for a low score in a preferred context setting are within logical boundaries.

The simulation program executes a number of CUF runs with the above probabilities and stores the user feedback for each run in the system.

6 Results

As stated in Sect. 4.3, the best context variable candidate for an unknown context dimension is recommended by a prediction algorithm that considers CUF runs from various context utility functions and application contexts. Our experiment uses the 7 context variables candidates for the context dimension user_companion presented in Sect. 5. For each of these 7 context variables, the algorithm considers a number of CUF runs, calculates the score for each context variable candidate and recommends the one with the highest score. Assuming that the user prefers movie recommendations computed considering the context variable user_companion:partner (i.e. in the context "user companion is the user's partner"), we consider a valid answer only if the prediction algorithm returns user_companion:partner.

We have conducted three experiments. Initially we have used only CUF runs of the active CUF (Experiment 1, Table 1). Following, we have additionally used CUF runs of other CUFs of the same application model, as well as CUF runs of CUFs of other application context models (Experiment 2, Table 2). At this point note that other than the active CUF mentioned above were defined in the same manner as the active CUF (i.e. based on the user feedback assumption), ensuring that these functions and their application context models utilize the same unknown context dimension user_companion. Finally, we have conducted experiments with random number of CUF runs to simulate more realistic settings (Experiment 3, Table 3).

Table 1. Experiment 1

# CUF runs	Prediction accuracy
200	0.867
175	0.8
100	0.6

Table 2. Experiment 2

# CUF runs	Prediction accuracy
200	1.0
150	0.98
75	0.833
40	0.633

Table 3. Experiment 3

# CUF runs	Prediction accuracy
Random (350–700)	1.0
Random (150–350)	0.9
Random (100–150)	0.833
Random (75–100)	0.8
Random (35–75)	0.633

We argue that a minimum prediction accuracy of 80 % (0.8) is needed, meaning that in a real scenario that a user provides feedback in line with the probabilities of Sect. 5, the system recommends the correct context variable to the CARS developer to use in subsequent CUF runs 8 out of 10 times.

The experiments show that accuracy is enhanced when more CUF runs are used. Experiment 1 needs more than 175 CUF runs to meet the accuracy threshold. This suggests that the context-aware recommender system must provide to the user 175 recommended items within the context specified by the particular CUF, and the user must provide feedback for each of these recommendations for the system to meet the accuracy threshold. Experiments 2 and 3 need considerable less CUF runs to meet the accuracy threshold because the prediction algorithm considers in addition other CUFs than the active one.

Finally, when attempting random CUF runs it is observed that at least 75 CUF runs are needed to meet the accuracy threshold. Moreover, it is noted that for experiments 2 and 3 at least 35 CUF runs are needed for the prediction algorithm to achieve 60 % success. This means that, to be successful more than once out of two times, the context-aware recommender system must provide to the user at least 35 recommended items for feedback.

7 Conclusions and Future Work

The CARS design system presented in this paper aims to provide to CARS developers an efficient and effective way to select and use known, as well as unknown at recommendation time context information for building their own application context models, allowing at the same time for sharing and reuse of context models and information among applications, regardless of the domain they belong to. The novelty of the system relies also in applying a utility-based recommendation algorithm that utilizes these context models by facilitating an interactional approach for partially observable context that incorporates user feedback. The system supports the developer in describing fully observable context, while it relies on user feedback to calculate the utility of a set of possible context values for the partially observable context and recommends the most suitable context value to be used by the system in future recommendation attempts.

Experiments in simulating CUF runs using simple probabilities depicting user feedback patterns have shown that accuracy is enhanced when more CUF runs are used. Less CUF runs are needed if the prediction algorithm considers in addition other context utility functions of the same, as well as other application contexts. In the latter setting, at least 75 CUF runs are needed to meet the accuracy threshold (80 %), while 35 CUF runs are needed for the prediction algorithm to achieve at least 60 % success.

The experimental evaluation described in this work serves as a proof of concept that the CARS design tool, if used in real settings by CARS developers building application context models and context utility functions, can provide valuable recommendations regarding unknown context dimensions. We argue that the assumptions made in this paper are realistic, but fine tuning of the system will be needed in real settings.

As future work we plan to evaluate the system by involving CARS developers, initially people from the university premises. This evaluation will serve as a proof that the system is able to function in real settings and rely on user feedback to recommend partially observable context information to be used within context-aware recommender systems. In addition, we plan to incorporate more recommendation algorithms to the system and experiment in more ways of handling partially observable context, as well as unobservable context. We argue that well known recommendation algorithms can be used within our tool with promising results.

References

1. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.) Recommender Systems Handbook, pp. 217–253. Springer, New York (2011)
2. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. ACM Trans. Inf. Syst. (TOIS) **23**(2005), 103–145 (2005)
3. Mettouris, C., Papadopoulos, G.A.: Contextual modelling in context-aware recommender systems: a generic approach. In: Haller, A., Huang, G., Huang, Z., Paik, H.-y., Sheng, Q.Z. (eds.) WISE 2011 and 2012. LNCS, vol. 7652, pp. 41–52. Springer, Heidelberg (2013)
4. Adomavicius, G., Mobasher, B., Ricci, F., Tuzhilin, A.: Context-aware recommender systems. AI Mag. **32**(3), 67–80 (2011)
5. Hariri, N., Mobasher, B., Burke, R.: Context adaptation in interactive recommender systems. In: Proceedings of the 8th ACM Conference on Recommender Systems, New York, NY, USA, pp. 41–48 (2014)
6. Dourish, P.: What do we talk about when we talk about context. Pers. Ubiquit. Comput. **8** (1), 19–30 (2004)
7. Anand, S.S., Mobasher, B.: Contextual recommendation. In: Mladenic, D., Semeraro, G., Hotho, A., Berendt, B. (eds.) WebMine 2007. LNCS (LNAI), vol. 4737, pp. 142–160. Springer, Heidelberg (2007)
8. Mettouris, C., Papadopoulos, G.A.: CARS context modelling. In: Proceedings of the 9th International Conference on Knowledge, Information and Creativity Support Systems, KICSS 2014, Limassol, Cyprus, pp. 60–71, 6–8 Nov 2014
9. Costa, A., Guizzardi, R., Guizzardi, G., Filho, J.: COReS: context-aware, ontology-based recommender system for service recommendation. In: UMICS 2007, 19th International Conference on Advanced Information Systems Engineering (CAISE 2007) (2007)
10. Emrich, A., Chapko, A., Werth, D.: Context-aware recommendations on mobile services: the m:Ciudad approach. In: Meissner, S., Moessner, K., Presser, M., Barnaghi, P. (eds.) EuroSSC 2009. LNCS, vol. 5741, pp. 107–120. Springer, Heidelberg (2009)
11. Moscato, V., Picariello, A., Rinaldi, A.M.: A recommendation strategy based on user behavior in digital ecosystems. In: MEDES 2010 Proceedings of the International Conference on Management of Emergent Digital EcoSystems, p. 25 (2010)
12. Uzun, A., Räck, C., Steinert, F.: Targeting more relevant, contextual recommendations by exploiting domain knowledge. In: HetRec 2010 Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems, pp. 57–62 (2010)
13. Rendle, S., Gantner, Z., Freudenthaler, C., Schmidt-Thieme, L.: Fast context-aware recommendations with factorization machines. In: SIGIR 2011 Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 635–644 (2011)
14. Rendle, S.: Factorization machines with libFM. ACM Trans. Intell. Syst. Technol. (TIST) **3**(3), 57 (2012)
15. Hidasi, B., Tikk, D.: General factorization framework for context-aware recommendations. Data Mining and Knowledge Discovery, 1–30, Springer (2015). doi: 10.1007/s10618-015-0417-y