A System for Context-Dependent User Modeling

Petteri Nurmi[1], Alfons Salden[2], Sian Lun Lau[3],
Jukka Suomela[1], Michael Sutterer[3], Jean Millerat[4],
Miquel Martin[5], Eemil Lagerspetz[1], and Remco Poortinga[2]

[1] Helsinki Institute for Information Technology HIIT,
P.O. Box 68, FI-00014 University of Helsinki, Finland
{firstname.lastname}@cs.helsinki.fi
[2] Telematica Instituut (TELIN), P.O. Box 589,
7500 AN Enschede, The Netherlands
{firstname.lastname}@telin.nl
[3] University of Kassel, Faculty of Electrical Engineering,
Wilhelmshöher Allee 73, 34121 Kassel, Germany
{firstname.lastname}@comtec.eecs.uni-kassel.de
[4] Motorola Labs, Parc Les Algorithmes, Saint-Aubin,
91193 Gif-sur-Yvette Cedex, France
{firstname.lastname}@motorola.com
[5] NEC Europe Ltd, Kurfürster Anlage 36,
69115 Heidelberg, Germany
{firstname.lastname}@netlab.nec.de

Abstract. We present a system for learning and utilizing context-dependent user models. The user models attempt to capture the interests of a user and link the interests to the situation of the user. The models are used for making recommendations to applications and services on what might interest the user in her current situation. In the design process we have analyzed several mock-ups of new mobile, context-aware services and applications. The mock-ups spanned rather diverse domains, which helped us to ensure that the system is applicable to a wide range of tasks, such as modality recommendations (e.g., switching to speech output when driving a car), service category recommendations (e.g., journey planners at a bus stop), and recommendations of group members (e.g., people with whom to share a car). The structure of the presented system is highly modular. First of all, this ensures that the algorithms that are used to build the user models can be easily replaced. Secondly, the modularity makes it easier to evaluate how well different algorithms perform in different domains. The current implementation of the system supports rule based reasoning and tree augmented naïve Bayesian classifiers (TAN). The system consists of three components, each of which has been implemented as a web service. The entire system has been deployed and is in use in the EU IST project MobiLife. In this paper, we detail the components that are part of the system and introduce the interactions between the components. In addition, we briefly discuss the quality of the recommendations that our system produces.

1 Introduction

In order to tailor mobile, context-aware applications and services to better match the needs of users, we need to make them able to adapt to the behavior of a user. In other words, we need to *personalize* the applications and services.

Background and related work. A widely used approach for personalization is user modeling [1]. In user modeling, the aim is to construct models that capture the beliefs, intentions, goals, and needs of a user [2]. In context-aware environments, we also need to associate the captured interests (needs, beliefs etc.) with the situation of the user. The main techniques for user modeling are knowledge representation (KR) methods and predictive user models [3]. The former covers traditional expert systems and logic-based systems, whereas the latter includes statistical machine learning techniques such as rule induction, neural networks, and Bayesian networks.

Previous work on context-dependent user modeling has mainly focused on KR methods. The most common approach has been the use of static rules, which are usually handcrafted and provided by the application designer (e.g., [4, 5]). Also the so-called preference approach (e.g., [6]), where users can specify application-specific rules, falls under the scope of KR based methods. In addition to rules, also ontology reasoning (e.g., [7]) and case-based reasoning (e.g., [8]) have been suggested. Finally, many context-aware middlewares provide system-level support for using KR methods (e.g., [9, 10]).

The KR methods suffer from two major problems. First of all, the techniques do not have a way to cope with uncertainty. Secondly, KR methods are not usually able to generalize their performance, i.e., to work well in previously unseen situations. However, in context-aware environments, there are various sources of uncertainty (e.g, the uncertainty about the goals of a user and inaccurate sensor signals) and the number of different situations that are relevant to a user might be very large. As a consequence, predictive user modeling seems the natural way to go. At the moment, however, work on using predictive user models in context-aware settings has been rather limited and all the uses are confined to a single application (e.g., [11]) or to a well defined spatial area such as a smart home (e.g., [12]) or a smart office (e.g., [13]).

Design process. Our work has been conducted within the EU project MobiLife[1]. In MobiLife, we have followed a user-centric design process (UCD). As the first step of the UCD process, we envisioned a set of high-level scenarios and evaluated them with users. With the help of the user feedback, the scenarios were used to construct mock-ups[2] of new, context-aware, mobile applications and services. The mock-ups were evaluated with users and both the feedback and the mock-ups were thoroughly analyzed.

[1] See `http://www.ist-mobilife.org` for more information.

[2] mock-up: a model of something that is used to show other people how it will work or what it will look like.

While analyzing the mock-ups, we discovered several uses for context-dependent personalization in the application ideas. Examples of the uses include modality recommendations (e.g., switching to speech output when driving a car), service category recommendations (e.g., journey planners at a bus stop) and recommendations of group members (e.g., people with whom to share a car).

Contribution. In this paper, we describe a generic system for context-dependent user modeling. The system can be used with the diverse set of applications described above. We have followed ideas from the field of user modeling and made the system independent of the applications and services that use it [1]. The structure of our system is highly modular; this ensures that the user modeling techniques that are used can be easily replaced. Furthermore, the modularity makes it easier to evaluate how well different techniques perform in different domains. The current implementation supports rule based reasoning and tree augmented naïve Bayesian classifiers (TAN). We have also decoupled learning and inference, which makes it possible to extend the system so that inferences are run on a mobile device. The system consists of three components, each of which has been implemented as a web service. The entire system has been deployed and is in use in the MobiLife project.

Our system is, to our best knowledge, the first generic user modeling system for context-aware settings. By following a UCD design process, we have been able to ensure that our system is suited to the requirements of future mobile, context-aware applications and services. Finally, we are, to our best knowledge, the first to apply predictive user models in context-aware settings in a task independent way.

Structure of the paper. The rest of the paper is organized as follows. Sect. 2 introduces the three application mock-ups that were used in the design process of our system. Sect. 3 details the components that are part of our system, introduces the interactions between different components, and discusses the quality of the recommendations made by our system. Sect. 4 concludes the paper and discusses future work.

2 Application Mock-Ups and Design Goals

In this section, we briefly introduce the application mock-ups that were the main drivers in the design of our system. In addition, we describe the design goals that we derived from the mock-ups.

Multimedia Infotainer is a mobile application that allows users to interact seamlessly with different input and output devices. For example, when the user is driving with a car, the output modality of the phone should be speech. Or, when the user is at home, she could project all her non-confidential messages, e.g., to a plasma screen for easier reading. In order to make the application truly seamless, it needs to be able to infer what input or output modality is best suited for the current situation.

MobiCar is a car sharing application for mobile users. Users ask the application to find them a suitable car for going into a particular location at a specific time. As additional information, the user may choose to include personal preferences related to the car. Examples of potentially relevant user preferences include smoking or non-smoking, pets or no pets, or music and trustworthiness preferences of the user. The goal of the application is to find a suitable allocation, i.e., a set of people whose requests match as closely as possible and who could be willing to share a car. In MobiCar, we need to know which preferences have the most impact on the acceptance of suggestions made by the system. To this end, we need user models that associate *groups* of people and their *preferences* to a particular situation (going by car to X on a weekday night).

Personal Communicator is an application that integrates the interactions between a user and mobile services. The Personal Communicator can be used, among others, for reading RSS news feeds, planning trips using online travel planners, and printing on Bluetooth-based printing services. In order to simplify and make the services more easily accessible to the user, we need to recommend services that might interest the user in her current situation. Thus, the Personal Communicator offers another application where situation-dependent user modeling could significantly reduce the amount of user interactions.

The three mock-ups above are rather diverse and, in order to simplify reuse of machine learning methods, we need a way to provider a *generic* and *unified* approach for context-dependent user modeling. In addition, we should be able to provide (context-aware) personalization support for different entities (individual users or groups) in different and potentially very diverse domains. Finally, as different tasks have traditionally called for different kinds of solutions, we need to be able to replace and easily reconfigure the machine learning algorithms that are used for user modeling. A summary of the main design goals is given below.

- User modeling algorithms are independent of the application and the entity for whom the recommendations are made (user or group).
- User modeling algorithms can be easily replaced and the methods that are used for a particular entity and application can be easily reconfigured.
- The system is applicable (without changes) to diverse domains.

3 System Structure

In this section, we discuss the structure of our context-dependent user modeling system. A component diagram of the system is shown in Fig. 1 and in the following subsections we detail the functionalities of the individual components. The system has been integrated with a context management framework that is based on a service-oriented architecture. To this end, our discussion refers to a specific *instance* of each of the components and in practice there can be multiple instances running.

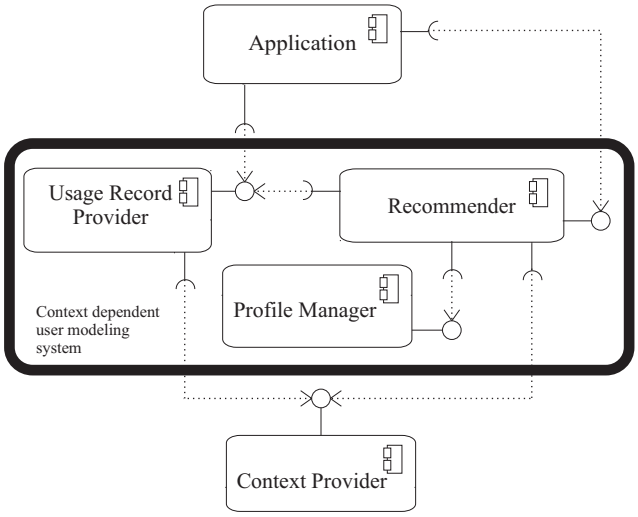

Fig. 1. A component diagram describing the context dependent user modeling system (the box in the middle) and its dependencies to external components.

3.1 Usage Record Provider

The first component that we describe is the *Usage Record Provider*, which is a repository that stores information on the behavior of a user and the context in which the behavior takes place. For example, the Multimedia Infotainer stores the selected modality and the corresponding situation of the user. The information that is stored in the Usage Record Provider is then used by the *Recommender* (see Fig. 1 and Sect. 3.2) to learn and update user models.

The entries that the system logs are called *Usage Records*. An example of a Usage Record is shown in Fig. 2. All context information is contained within the XML tag `contextElement`. For privacy reasons, we have removed and obfuscated some of the context parameters in the example.

In our setting, the context information comes from components that are called *Context Providers*. More details on Context Providers and the used context model can be found at [14] and on the website of the MobiLife project[3]. See the list of software components on the website for a number of concrete examples of Context Providers, as well as descriptions of what kind of context information they provide.

The Usage Record Provider also acts as an entry point for applications and services into the system. To get useful information from the system, the applications must send usage information to the Usage Record Provider. The sent information should contain at least unique identifiers for the user and for the action the user performs. When the Usage Record Provider receives usage information, it checks whether it can enrich the context associated to the usage

[3] `http://www.ist-mobilife.org/`

```
<usageRecord
        action="update"
        actor="355023003598706"
        application="Buddy"
        feedback="0.0"
        initiationType="manual"
        recommendationType="action"
        timestamp="2005-12-13T07:51:34.000Z">
    <contextElement>
        <parameter name=”location”>
            <parameter name=”cluster” value=”1”>
                <parameter name="mcc" value="204"/>
                <parameter name="mnc" value="815"/>
                <parameter name="cellid" value="45960"/>
                <parameter name="latitude" value="32.232845916875"/>
                <parameter name="longitude" value="9.8896758573835"/>
            </parameter>
        </parameter>
    </contextElement>
</usageRecord>
```

Fig. 2. An example of a Usage Record.

information. This is done by contacting those Context Providers that are known to contain parameters that are not yet part of the received usage information. This step is especially useful for mobile applications: both the Usage Record Provider and the Context Providers typically reside on a server, reducing the communication costs of the mobile client.

3.2 Recommender

The *Recommender* is responsible for performing all tasks related to user modeling: learning and updating user models and making inferences with the learned models. Within the component, the functionalities have been divided into three kinds of modules: Finders, Mappers and Reasoners.

Finders. The Finders are responsible for finding components and for the interactions that take place between the Recommender and other components. There is a Finder for each component with which the Recommender needs to interact. The reason for separating the interactions with external components is that it offers us more flexibility as the Recommender is not tied to a particular implementation of the external components. Thus, we can modify the external components at any stage and the only thing we need to change in the Recommender is the corresponding Finder.

Mappers. The Mappers are responsible for mapping structured information into a non-structured (flat) form and vice versa. Context and behavior information

are often structured (for example, nested context parameters as illustrated in Fig. 2, or references to a tree-like or graph-like ontology of user behavior), whereas common machine learning algorithms can only handle flat data (for example, numeric vectors or name-value pairs).

The requirements of the mappers are two-fold. First, the data formats must be made compatible. Second, the translated data should contain information that makes it as efficient as possible to learn and apply the context-dependent user model. While the first requirement could be handled in an application-independent manner, this is not the case for the second requirement. A good mapping may depend on the application: for example, an application-specific ontology of service categories can be used to map fine-grained identifiers of individual services into higher-level identifiers of service categories; this way the system can make useful predictions with considerably less training data and the predictions generalize to new situations. Furthermore, a good mapping may also depend on the specific machine learning algorithm.

Examples of possible mappings include the following: (i) Flattening structured information; e.g., translating the Usage Record of Fig. 2 into (cluster = $1, \mathrm{mcc} = 204, \ldots, \mathrm{longitude} = 9.89$). (ii) Feature selection, i.e., choosing a subset of input data; e.g., translating the Usage Record of Fig. 2 into (cluster = 1). (iii) Extracting higher-level features in an application-specific manner; for example, using an external ontology of context or behavior information. (iv) Technical low-level translations as required by the specific machine learning algorithms; for example, discretizing real-valued data to integral values, or mapping class-valued data to Boolean vectors.

Reasoners. The Reasoners are responsible for encapsulating implementations of individual machine learning algorithms into the Recommender. Thus, the Reasoners are the part of the architecture where the actual learning and inference is done. The Reasoners interact closely with the Mappers: Before usage records are used for learning, the Mappers map the context and behavior information into vectors that are given to the Reasoners. Similarly, when the Reasoners are used for inference, the Mappers take the results of the inference and map them so that the applications can understand the results.

To have a clear separation between learning and inference, we have specified separate interfaces for the components that offer learning functionalities and for the components that offer inference functionalities. The separation of interfaces is crucial as in a mobile device we seldom have enough resources to run the learning phase. However, once we have learned a model, it may be possible to run the inference stage even on the phone. Another advantage of the clearly separated interfaces is that this further improves the reusability of the implementations of individual algorithms.

Once a Reasoner has learned a new model, we store the models into a Profile Manager (see Sect. 3.3). The motivation for this is improved scalability and better distribution of functionalities. Furthermore, by storing the models into a Profile Manager, the user models become part of the user profile and thus all relevant preference-related data is stored in a single place.

In the current implementation of the Recommender, we have support for tree augmented naïve Bayesian classifiers (TAN; see, e.g, [15]) and for rule-based reasoning. The algorithm that is used for learning the TAN classifiers is described in [15] and as the inference method we use so-called quasi-Bayesian inference [16]. Rule learning, on the other hand, has been implemented using the Ripper algorithm [17] and the inference utilizes RuleML-based [18] rule-engines.

3.3 Profile Manager

The third component of the system is the *Profile Manager*. A detailed description of the Profile Manager can be found at [19]; we focus on the functionalities that are relevant for the user modeling system.

The Profile Manager is responsible for managing profiles of different entities such as users and groups. Each entity has a separate profile that is further divided into views that contain data specific to a particular application or a set of applications. In terms of the user modeling system, the role of the Profile Manager is to act as a persistent storage for user models. Whenever new models are learned, the models are sent to the Profile Manager, which stores them in the appropriate user profile. When an application requests for recommendations, the Recommender fetches suitable models from the Profile Manager.

The entity for whom the recommendations are requested uniquely specifies the profile to use; the application specifies the appropriate view within the profile. Since the user modeling system has been designed to be usable in a wide variety of scenarios and potentially with a large number of users, we need to properly index the models that are stored in the Profile Manager. To this end, we use qualifiers, which are collections of (name, value) pairs. Thus, the qualifiers specify a set of index terms for each model, which facilitates finding the appropriate models. As an example, when we store a newly created model, we use the names of context parameters to specify a set of qualifier constraints. When we then want to infer on the models, we can first check what context information is available and to fetch only those models than can be used with the available context information.

3.4 Evaluation of Recommendations

In order to evaluate the quality of the recommendations that our system produces and to ensure that the system works properly, we selected two datasets from the UCI machine learning repository [20]. We exported the data into our system by mapping the data records into usage records. After this, we used leave-one-out cross-validation to test the classification accuracy of the system. In the experiments, we used the TAN, and as the system is able to make probabilistic predictions, we also measured the logarithmic loss ($-\log p_i$, where p_i is the probability assigned to the correct class).

The data sets and the experiment results are summarized in Table 1. The first data set, Vote, consisted of 435 records. The data set contains many records with missing values. In this experiment, we considered only the part of the data set with complete information, leaving us with 232 records. The classification

task is a binary task and our system was able to achieve a 94.4 percent accuracy on the data set. The second data, Zoo, consisted of 101 records, each of which had 17 attributes. However, since one of the attributes was a unique identifier, we removed it. On this data set, our system achieved an accuracy of 98 percent.

Table 1. Summary of the used datasets and results of the experiments.

Data set	Records	Attributes	Classes	Accuracy	Avg. log-loss
Vote	232 (435)	16	2	94.4%	0.165
Zoo	101	16 (17)	7	98.0%	0.121

4 Conclusions and Future Work

In this paper, we have presented a generic component for context-dependent user modeling. The component has been designed with flexibility in mind and it eases the reuse and the evaluation of different machine learning methods. In the future, our goal is to implement additional machine learning methods, more specifically rule learning and neural networks, to the Recommender. We will also perform user acceptance studies in different settings and test the relevancy of different machine learning methods for these settings.

Acknowledgements

This work has been performed in the framework of the IST project IST-2004-511607 MobiLife, which is partly funded by the European Union. The authors acknowledge the contributions of their colleagues.

References

1. Kobsa, A.: Generic user modeling systems. User Modeling and User-Adapted Interaction **11**(1–2) (2001) 49–63
2. Horvitz, E., Breese, J., Heckerman, D., Hovel, D., Rommelse, K.: The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In: Proc. 14th Conference on Uncertainty in Artificial Intelligence (UAI), Morgan Kaufmann Publishers (1998) 256–265
3. Zukerman, I., Albrecht, D.W.: Preditive statistical models for user modeling. User Modeling and User-Adapted Interaction **11**(1–2) (2001) 5–18
4. Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji, N., Reiger, K., Shaffer, J., Wong, F.L.: SenSay: A context-aware mobile phone. In: Proc. 7th IEEE International Symposium on Wearable Computers (ISWC), IEEE (2003) 248–249

5. van Setten, M., Pokraev, S., Koolwaaij, J.: Context-aware recommendations in the mobile tourist application COMPASS. In: Proc. 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH). LNCS, Vol. 3173. Springer-Verlag (2004) 235–244
6. Henricksen, K., Indulska, J.: Personalizing context-aware applications. In: Proc. Workshop on Context-Aware Mobile Systems (CAMS). LNCS, Vol. 3762. Springer-Verlag (2005) 122–131
7. Sadeh, N.M., Gandon, F.L., Kwon, O.B.: Ambient intelligence: The MyCampus experience. CMU-ISRI-05-123, Carnegie Mellon University (2005)
8. Kofod-Petersen, A., Aamodt, A.: Case-based situation assesment in a mobile context-aware system. In: Proc. Artificial Intelligence in Mobile Systems (AIMS), Saarbrücken, Germany, Universität des Saarlandes (2003) 41–48
9. Ranganathan, A., Al-Muhtadi, J., Campbell, R.H.: Reasoning about uncertain contexts in pervasive computing environments. IEEE Pervasive Computing **3**(2) (2004) 62–70
10. Yau, S.S., Karim, F.: An adaptive middleware for context-sensitive communications for real-time applications in ubiquitous computing environments. Real-Time Systems **26**(1) (2004) 29–61
11. Horvitz, E., Koch, P., Sarin, R., Apacible, J., Subramani, M.: Bayesphone: Precomputation of context-sensitive policies for inquiry and action in mobile devices. In: Proc. 10th Conference on User Modeling (UM). LNCS, Vol. 3538. Springer-Verlag (2005) 251–260
12. Tapia, E.M., Intille, S.S., Larson, K.: Activity recognition in the home setting using simple and ubiquitous sensors. In Ferscha, A., Mattern, F., eds.: Proc. 2nd International Conference on Pervasive Computing. LNCS, Vol. 3001. Springer-Verlag (2004) 158–175
13. Oliver, N., Garg, A., Horvitz, E.: Layered representations for recognizing office activity. In: Proc. 4th IEEE International Conference on Multimodal Interaction (ICMI), IEEE (2002) 3–8
14. Floréen, P., Przybilski, M., Nurmi, P., Koolwaaij, J., Tarlano, A., Wagner, M., Luther, M., Bataille, F., Boussard, M., Mrohs, B., lau, S.: Towards a context management framework for mobilife. In: Proc. IST Mobile & Wireless Communications Summit. (2005)
15. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning **29** (1997) 131–163
16. Cozman, F.: A derivation of quasi-Bayesian theory. Technical Report CMU-RI-TR-97-37, Robotics Institute, Carnegie Mellon University (1997)
17. Cohen, W.W.: Fast effective rule induction. In: Proc. 12th International Conference on Machine Learning (ICML), Morgan Kaufmann (1995) 115–123
18. Boley, H., Tabet, S., Wagner, G.: Design rationale of RuleML: A markup language for semantic web rules. In: Proc. International Semantic Web Working Symposium (SWWS). (2001) 381–401
19. Coutand, O., Sutterer, M., Lau, S., Droegehorn, O., David, K.: User profile management for personalizing services in pervasive computing. In: Proc. 6th International Workshop on Applications and Services in Wireless Networks (ASWN). (2006)
20. Newman, D., Hettich, S., Blake, C., Merz, C.: UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Sciences (1998) `http://www.ics.uci.edu/~mlearn/MLRepository.html`