

# A Modular Hybrid Recommender Engine

Marco Tiemann

Steffen Pauws

Fabio Vignoli

*Philips Research Europe, High Tech Campus 34, 5656 AE Eindhoven, The Netherlands*

## Abstract

We present an architecture for a user goal-adaptable hybrid recommender engine that flexibly and dynamically integrates heterogeneous data sources and algorithms to improve the user experience. To this end, we identify three main goals which recommender engines should support and introduce a modular and configurable engine that helps meet these goals. We illustrate practical benefits of the recommender engine architecture with implementation examples from a personalized music service.

## 1 Introduction

Recommender systems predict a user's interest in one or several items based on data in a user model or user profile. Various system functionalities can be realized with such personalized interest predictions: they can be used for "classic" recommender system applications such as creating ranked recommendation lists from a large item pool [11] or filtering out uninteresting messages from a stream of incoming newsgroup messages [6]. Today, recommender systems also enable a range of other systems and services such as personalized museum tours and personalized media channels.

### 1.1 An Enabling Architecture for Recommender Systems

While commercially deployed recommender systems often apply established methods such as item-based collaborative filtering algorithms [9], there is still significant scientific and commercial interest in further improving the accuracy, computational efficiency and usefulness of recommender systems. We are examining hybrid recommender systems. Hybrid recommender systems integrate data that are conventionally not used as part of a single recommender algorithm or that combine different recommender algorithms that use the same input data [2]. They are used to address problems inherent to particular types of recommender systems, such as the *cold start problem* in collaborative filtering, or to reduce prediction errors [1].

Next to research concerned with algorithmic improvements, novel applications of recommender systems play an important role both in academic and commercial research. Recommender systems can for example be used to propose personalized sightseeing tours or to suggest specific car configurations based on user profile data. In such applications, the user goals that need to be addressed may be better served if predicted preferences are modified to better suit the specific user goals. It may for instance be desirable to consider properties of the proposed set of items, such as distances between sights to visit on a sightseeing tour. We are investigating whether such novel applications that support such specific user goals can support users better via specifically tailored recommender systems.

In this paper, we present a system architecture for a recommender engine that allows us to easily implement, combine, evaluate and deploy recommender systems that facilitate hybrid recommender systems and can flexibly support novel recommendation-based applications. We use the term *recommender engine* to denote a system that encapsulates one or more recommender algorithm implementations. We elaborate on the main requirements that motivate the system development and describe the developed recommender engine architecture, focusing on its flexible modular design. We demonstrate how solutions can be implemented using the recommender engine both on an abstract level, in a practical application and in an online end user trial. We conclude with an outlook towards future development opportunities.

## 1.2 Related Work

Numerous research publications treat hybrid recommender systems and examine applications based on them. Burke [2] provides a categorization of hybrid recommendation approaches and an overview of related publications. User goals when interacting with recommender systems have been investigated both from a theoretical point of view and in practical recommender system implementations. Herlocker et al. [5] identify ten types of relevant user goals. Setten et al. [10] introduce a generic system architecture that tries to support different user goals via specifically tailored algorithm combinations. Few research publications address the design and integration of recommender engines in detail. Among the published work, the generic architecture described in [10] is most closely related to the work described in this paper. Gstrein et al. [4] describe a recommender engine for the music domain that integrates heterogeneous data sources and relies on instance-based lazy learning techniques.

## 2 Requirements

We can define several requirements for a recommender engine that can support various types of hybrid recommendation algorithms and that can facilitate novel recommendation applications. We elaborate on three core requirements: integrating heterogeneous data, integrating recommender algorithms, and supporting different user goals.

### 2.1 Integrating Heterogeneous Data

Recommender systems are commonly differentiated based on the data they use to compute interest predictions. Frequently, knowledge-based, content-based and collaborative filtering types of recommender systems are differentiated in this way. Hybrid recommender systems can be used to combine recommender systems that use different data sources [2]. In particular, they are often used to combine content data with user profile data of a multitude of users. In the music domain for example, such content data may consist of catalogue metadata, manually annotated descriptors and/or descriptive features extracted from audio data; user profile data may consist of user characteristics, content-related data provided explicitly by users (such as item ratings), and/or implicitly gathered data such as recorded user-system interactions. Such data integrations can be carried out in two ways: first, by integrating heterogeneous data with a single algorithm [8]; second, by using specialized algorithms for each data source and combining their output. We are especially interested in the latter approach, because it is inherently flexible regarding which algorithms can be employed, which data sources can be integrated and which methods can be used to combine them [2]. It also allows us to integrate third party recommendation services – including ones using hybrid algorithms that integrate heterogeneous data with a single algorithm – into the recommender engine, which may in the future become an important benefit in service-based, networked infrastructures.

### 2.2 Integrating Recommender Algorithms

Hybrid recommender systems do not necessarily integrate heterogeneous data sources – it is also possible to combine recommender systems that predict interest based on the same input data. For such approaches, one can distinguish between approaches combining a) distinct algorithms, b) differently parameterized instances of the same algorithms and c) instances of algorithms that are trained on different training (sub-)sets. In addition, approaches can also be distinguished by the method applied to combine outputs. Machine learning techniques that can be characterized as above belong to the family of ensemble learning methods; applying them in recommender systems can reduce the average prediction error as has been shown elsewhere [1, 13].

### 2.3 Supporting Different User Goals

Herlocker et al. [5] describe different goals users may pursue when using a recommender system. We are examining approaches to tailor recommender systems to better support specific user goals. In order to do this, we identify specific properties of the output of a recommender system. We can then use these properties to describe a particular *recommendation task* that has to be serviced by a recommender system in more detail. We consider the following three properties to be especially relevant for applications with which we are currently concerned:

- *Expectedness*: Recommender systems can facilitate the serendipitous discovery of unexpected yet interest-relevant items [11]. We assume that such recommendations are not equally desirable for all types of recommendation functionalities. They may for example be of central importance for users searching for unknown yet interesting items such as a new book to read, but they may not be desirable for recommendation functionalities with reminder or trust-building purposes.
- *Diversity*: Recommender systems are often used to select item subsets. Depending on the user goal, it may be beneficial to enforce particular properties for such subsets via trade-offs between item space coverage and predicted interest [12, 15].
- *Error sensitivity*: Likewise depending on the user goal, the number and type of errors that a recommender system makes may be important factors [5]. Proposing false positive items may not be detrimental to the user experience in certain applications as long as suitable candidate items are proposed as well, while it may have a strongly negative impact on the user experience in other applications (see Section 4.3). Inversely, omitting true positive items may reduce the usefulness of a recommendation functionality, for example in legal settings.

### 3 Architecture

The recommender engine architecture described in this section has been designed to support both the data and algorithm integration that is required to support various types of hybrid recommender systems, and to allow us to tailor the recommender engine output to better support user goals via recommendation tasks.

#### 3.1 Design Concepts

The recommender engine architecture is based on four main design concepts which support the stated requirements:

- *Modular architecture*: The recommender engine can integrate various algorithms in an extensible modular architecture. Each algorithm is realized as an independent component which implements a set of mandatory core functions and can implement additional optional functions.
- *Dynamic component coupling*: Recommender engine components can be combined to enable various training and prediction methods. We use a dynamic configuration mechanism with which the system can create, assemble and configure components at run-time.
- *On-line learning and prediction support*: Ideally, a recommender system should learn from newly available data immediately and on-line so that all relevant and available data is used to predict for a user. The architecture provides means to update all components dynamically as new data become available.
- *Simple integration of domain-specific components*: Core components of a recommender engine can be used across a variety of application domains. The recommender engine provides support for pluggable domain data definitions and data access components.

#### 3.2 System Overview

The recommender engine architecture consists of four main elements:

- One or more *event observers* receive and evaluate data related to users, content or relevant other data and requests that trigger predictions or updates of the recommender engine. Events related to users can for example consist of explicitly provided ratings for content items, or observed usage data such as selecting an item in a menu. Events are analyzed by event observers and can be used to initiate processes such as updating stored user data, updating data models, and generating or validating recommendations for a user. When a prediction or update action is initiated, an event observer assembles a recommendation task for that request.
- A *recommender engine wrapper* processes requests. It analyzes a request and selects a combination of components to service it by predicting interest or by updating components. It provides access to available user data, user models, item data and other supported types of data.

- *Scorer* components implement a recommender algorithm to influence interest predictions.
- *Merger* components implement combination methods that unify the output of several scorer components into a single result.

An overview of the recommender engine architecture is provided in Figure 1 a). The following sections describe the core components in more detail.

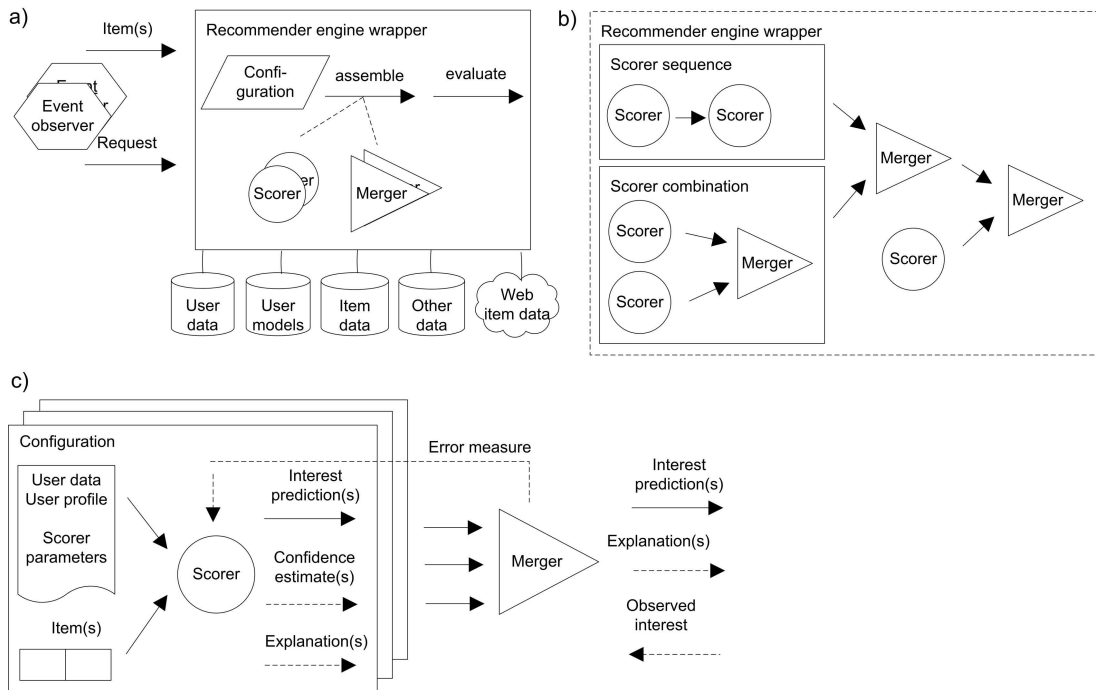


Figure 1: Conceptual overview of the recommender engine architecture; a) illustrates the general system architecture, b) shows example component configurations, c) describes scorer and merger component inputs, outputs, and interactions

### 3.3 Scorer Components

*Scorer* components are the basic building blocks of the recommender engine. A scorer encapsulates a recommender algorithm (such as a collaborative filtering algorithm), an algorithm that is used to modify a result for example a diversity metric [12] or generally any method that can contribute to a recommendation output (for instance ratings provided by the user [10]). *Composite scorers* are a special class of scorers that encapsulate a sequence of scorers or a combination of scorers with a *merger*. They can be nested within larger configurations. Composite scorers can also implement dynamic updating mechanisms that can duplicate, reconfigure, add and remove contained components. Figure 1 b) shows an example engine configuration with a scorer sequence and a nested scorer-merger combination.

Figure 1 c) describes a scorer as a black box component comparable to [10]. The figure shows required inputs and outputs (depicted with solid arrows) and examples for optional ones (depicted with dashed arrows). A scorer is activated and configured via a scorer configuration, with which algorithm parameters can be set and user data can be provided to a scorer. A scorer predicts values for input items and can provide additional data such as confidence estimates or explanations on how the scorer arrived at a given prediction. A scorer can optionally receive observed prediction error values to dynamically adapt configuration parameters.

### 3.4 Merger Components

*Merger* components combine the output of one or more scorers into a unified result. Mergers can implement basic combination methods such as (weighted) averaging, majority voting, or switching between algorithms [2] as well as more complex mechanisms such as learning algorithms trained on the output of

scorer components as they are used in stacked generalization ensembles. Figure 1 c) describes a generic merger component with mandatory (solid arrows) and optional (dashed arrows) inputs and outputs. Mergers are activated and configured via merger configurations that are used analogously to scorer configurations described in Section 3.3 (not depicted in Figure 1 c).

### 3.5 Recommender Engine Wrapper

A recommender engine wrapper processes a prediction request by selecting or assembling a configuration encapsulating scorers and mergers. It defines which components are to be used with which parameters as well as how they are to be composed. It also controls the order in which components are called. This is relevant when updating components with additional user or item data, where components are updated in the reverse order of processing for prediction (see Figure 1 b) for an example of the order of processing for prediction). This allows mergers to request item predictions from scorers before they are updated with additional data, so that a model learned by a merger component can be updated before the contained scorers themselves are updated. This allows mergers to evaluate the performance of scorers using these data samples.

## 4 Example Application

We are using a Java reference implementation of the described recommender engine architecture for evaluation and demonstration purposes in several application contexts. In this section, we illustrate how the recommender engine can be used in the xStream personalized music recommendation service (abbreviated as *xStream* in this paper) that is used in large scale user evaluations of personalized music recommendation and personalized music delivery services within Philips Research.

### 4.1 The xStream Personalized Music Service

xStream is a web-based service that allows experiment participants to browse and listen to a centrally stored 80 000 song music collection. In addition, it provides access to music channels that are presented similarly to online radio channels, but additionally allow participants to pause or skip playback of a song. The system is used to gather usage data of several hundred participants. Various search and recommendation functions can be enabled and used to carry out user experiments (such as [3]).

The recommender engine is loosely integrated into the xStream system as depicted in Figure 2 a). Event observers monitor incoming data so that the recommender engine can be directly triggered to predict item interests for recommendation functionalities and so that all user activities that are being received by the system can immediately be evaluated by the recommender engine. Depending on which events are evaluated by the event observers, a suitable recommendation task is created and used to call the recommender engine.

### 4.2 Generating Recommendation Lists

It can be argued that recommender systems are most frequently employed to generate ranked top- $n$  lists of recommended items [5]. Such top- $n$  recommendation lists can address various user goals, and specific recommendation strategies can be employed to better address such specific user goals. In xStream, recommendation lists can be provided to support two types of user goals: first, to encourage content exploration and discovery (for instance on a welcome page). Recommender systems can support exploration and discovery by employing diversity measures to influence the ranking of a generated recommendation list [15]. Second, recommendation lists can be provided to increase the efficiency and/or effectiveness of creating or extending a play list by recommending items to add to it. For such recommendation lists, we assume that coherence with the songs in the play list is desired by users. To achieve this, predicted interest is modified by determining the average similarity of songs to be predicted to the songs in the play list to be augmented, using a similarity measure described in [14].

Figure 2 b) depicts an example recommender engine configuration that facilitates both recommendation tasks described above. The configuration consists of a sequence of two scorers. The first implemented scorer computes recommendations using either a content-based recommender algorithm or a collaborative filtering algorithm to predict item interest. Descriptions of the recommender algorithms used can be found

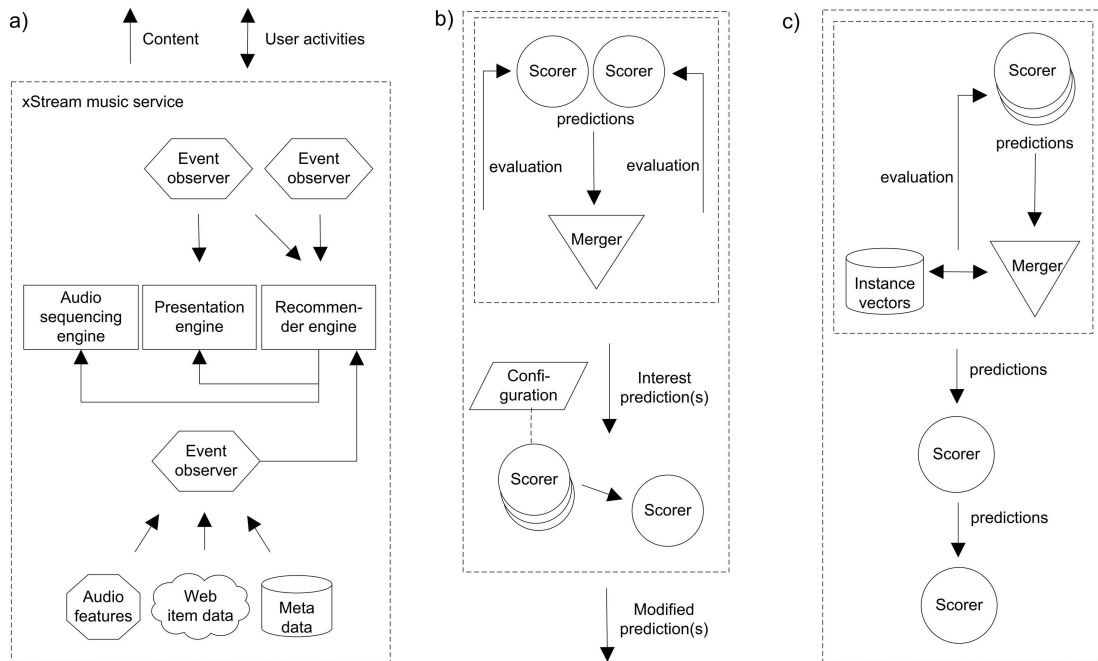


Figure 2: The xStream application example; a) describes the integration of the recommender engine in the system, b) describes a recommender engine configuration for recommendation list generation, c) depicts a configuration for personalized music channel creation and adaptation

in [13]. The performance of these algorithms for each user is continually evaluated. The better performing algorithm of the two is selected to predict item interest (see also [15]). These predictions are then modified, using either a diversity measure or similarity measure depending on the anticipated user goal as described above. Finally, a top- $n$  recommendation list is created from the modified results and is presented to the user.

### 4.3 Song Selection for Personal Music Channels

The xStream system is also being used to demonstrate hybrid recommender systems for creating personalized music channels, where music that is played back in a radio-like channel is selected based on various types of user data (such as ratings, listening behavior, personality profile) and music content data (such as catalogue meta data, descriptive features extracted from audio data, artist and album descriptions mined from various web resources). We provide a configuration used as one of our demonstration systems as an example of such a system. It is depicted in Figure 2 c). The configuration is based on earlier work described elsewhere [13]. It combines collaborative filtering implementations (derived from the main types described in [1]), content-based recommender algorithms using catalogue meta data, content-based recommender algorithms using extracted audio features, and scorers that return interest predictions using a user's artist, album and song ratings. Results are aggregated using a merger which employs a  $k$ -nn classifier as merger that uses output predictions and confidence values of recommender algorithms as a vectors of training examples to predict a user's interest in items (see also [13]).

A combination of several recommender algorithms such as the one described above coupled with an additional classifier used to merge results may not be feasible for computing recommendations on-line, meaning that it will be used to predict interests during scheduled update cycles. In order to still adapt to user activities in a timely fashion, we use a second recommender algorithm that can be used on-line to modify the predictions made using the hybrid recommender.

Finally, the presentation of recommendations in a personalized media channel differs from a presentation as a recommendation list. Instead of presenting a ranked list of items, a subset of likely interesting items that may also include items with known high interest is selected and must be arranged into a sequence for which the between-item coherence is likely more relevant than the predicted or stated interest of a user. To support this, we reorder items from a top- $n$  subset of the item set using a play list generation algorithm that uses content-based features to group songs into a sequence that is perceived as more pleasing (see [7]).

## 4.4 User Evaluation of Recommender Algorithms

xStream is also being used as a research instrument to carry out comparative user evaluations of recommender algorithms for personal music channels such as the one described above. In a currently ongoing user experiment, participants listen to personal music channels during their daily work routine. While they are using their personal music channel, they are periodically asked to fill in questionnaires about their perception of various properties of the music selected for them. The experiment is conducted in three iterations with a duration of 8 weeks per iteration. In each of these iterations, participants interact with four different personal music channels generated using four different recommender algorithms. Algorithms for the second and third iteration of the user evaluation are selected and adapted based on feedback from previous iterations. All recommender algorithms used in the user evaluation have been implemented with the recommender engine. Apart from a random baseline algorithm and two recommender algorithms which represent the current state of the art, all evaluated recommender algorithms are hybrid algorithms composed of multiple component algorithms. The recommender engine enables such complex user evaluations and simplifies their technical realization. The key benefits are:

- Various hybrid recommender systems can be implemented easily and quickly. New algorithm variants are implemented and tested between iterations of the ongoing user evaluation despite the very brief time windows available for doing so.
- User and usage data can be used to update recommender systems directly as they enter the system. All evaluated recommender algorithms can hence be updated on-line as additional usage data and user ratings enter the system.
- Components and data that are shared by recommender system components are not duplicated in the recommender engine. This allows us to carry out experiments with several recommender algorithms on modestly powerful hardware (a single Xeon 5140 dual-core processor, 2 GB RAM server), while servicing a participant population of up to 200 concurrent users.
- Functionalities for switching between algorithms depending on the experimental design are readily available. The system also ensures that every recommender algorithm only has access to the subset of user data that was gathered while a user was using that particular algorithm.

By using the described recommender engine, we are able to carry out complex and realistic user evaluations such as the one described above, and we can quickly adapt solution approaches for upcoming iterations of experiments. Preliminary analyses of data gathered in the user evaluation indicate that, compared to two state of the art non-hybrid recommender systems, the first evaluated hybrid recommender system proposed items that were rated as liked more frequently by participants, and that, on average, using the personal music channel generated by the hybrid recommender system was liked more. Final results of the user evaluation will be made available in a forthcoming publication.

## 5 Conclusion

The run-time configurable recommender engine described in this paper is currently being used to carry out the described and other user experiments concerned with user profiling, recommender algorithms and hybrid recommender algorithms. In the context of these evaluations the modular architecture can be used directly to implement various combinations of algorithms for recommendation and related tasks, to dynamically select appropriate combinations at run-time, and to implement and carry out comparative studies in which participants are dynamically assigned to different recommender engine configurations. In addition, the simple component-based architecture allows researchers to quickly implement and rapidly prototype and evaluate novel approaches that may improve specific recommendation functionalities or recommender systems in general.

Future work using the recommender engine will focus on examining a number of research questions related to hybrid recommendation and the usefulness of recommendation tasks – a number of examples that illustrate the focus of this ongoing work have been provided in this paper. Future extensions of the recommender engine architecture may focus on the integration of third-party recommender systems using available interfaces and specialized recommender system query languages, an extension of the engine architecture that

formalizes mechanisms to dynamically create component configurations for specific user goals by evaluating a large number of candidate components and component configurations automatically, and scenarios where recommender engines are used in distributed environments such as in peer-to-peer systems.

## References

- [1] R. Bell and Y. Koren. Lessons from the netflix prize challenge. *SIGKDD Explorations*, 9:75–79, 2007.
- [2] R. Burke. Hybrid web recommender systems. In *The Adaptive Web*, pages 377–408. Springer, Berlin, 2007.
- [3] G. Dunn and B. Ruyter. Relating personality to reported music preferences and listening behavior. In *Proceedings of the ICMPC10 International Conference on Music Perception and Cognition*, pages 552–561, 2008.
- [4] E. Gstrein, F. Kleedorfer, R. Mayer, C. Schmotzer, G. Widmer, O. Holle, and S. Miksch. Adaptive personalization: A multi-dimensional approach to boosting a large scale mobile music portal. In *Fifth Open Workshop on MUSICNETWORK: Integration of Music in Multimedia Applications*, Vienna, Austria, 2005.
- [5] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.
- [6] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.
- [7] S. Pauws, W. Verhaegh, and M. Vossen. Music playlist generation by adapted simulated annealing. *Information Sciences*, 178(3):647–662, 2008.
- [8] A. Popescul, L. Ungar, D. Pennock, and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *17th Conference on Uncertainty in Artificial Intelligence*, pages 437–444. ACM Press, 2001.
- [9] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, Hong Kong, Hong Kong, 2001. ACM.
- [10] M. Setten, M. Veenstra, A. Nijholt, and B. Dijk. Prediction strategies in a tv recommender system - method and experiments. In *Proceedings of the IADIS International Conference on WWW and Internet 2003*, Algarve, 2003.
- [11] U. Shardanand and P. Maes. Social information filtering: Algorithms for automatic word of mouth. In I. Katz and R. Mack, editors, *Human Factors in Computing Systems. CHI95 Conference on Human Factors in Computing Systems*, pages 210–217, Denver, Colorado, USA, 1995. ACM Press.
- [12] B. Smyth and P. McClave. Similarity vs. diversity. In D. Aha and I. Watson, editors, *Proceedings of the 4th International Conference on Case-Based Reasoning, ICCBR 2001*, pages 347–361, Vancouver, BC, Canada, 2001. Springer.
- [13] M. Tiemann, S. Pauws, and F. Vignoli. Ensemble learning for hybrid music recommendation. In *Proceedings of the 8th International Symposium on Music Information Retrieval*, pages 179–180, Vienna, Austria, 2007.
- [14] F. Vignoli and S. Pauws. A music retrieval system based on user-driven similarity and its evaluation. In *2005 International Symposium on Music Information Retrieval*, pages 272–279, London, 2005.
- [15] C. Ziegler, S. McNee, J. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 20–32, Chiba, Japan, 2005. ACM.