An Architecture for a General Purpose Multi-Algorithm Recommender System

Jose C. Cortizo, Francisco M. Carrero and Borja Monsalve
BrainSins
http://www.brainsins.com
Madrid, Spain
{josecarlos.cortizo, francisco.carrero, borja.monsalve}@brainsins.com

ABSTRACT
Although the actual state-of-the-art on Recommender Systems is good enough to allow recommendations and personalization along many application fields, developing a general purpose multi-algorithm recommender system is a tough task. In this paper we present the main challenges involved on developing such system and a system's architecture that allows us to face this challenges.

Categories and Subject Descriptors
H.4 [**Information Systems Applications**]: Miscellaneous

General Terms
Design, Algorithms

Keywords
General purpose recommendations, System architecture, API, Multi-algorithm

1. INTRODUCTION
There is a lot of literature on Recommender Systems for specific online domains like social software items [4], music [1], queries (in search engines) [8], news [6], e-commerce [7] or even for non online domains such as [5]. Those recommender systems employ specific techniques for specific domains in order to produce the most accurate systems for each single domain.

We wanted to integrate a recommender system in Wipley[1], our social network for videogamers launched by the end of 2009. With this purpose, we worked on a recommender system for videogames using a collaborative filtering approach with multidimensional and contextual features to fit this particular domain. After that, we wanted to improve the recommender with a content-based recommender system for those games with none or few ratings and a social-based recommender system [3]. We also needed to adapt the resulting system to other domains, beginning with image recommendations to be integrated in FlickrBabel[2], our multilingual multimedia search engine. And, finally, we decided to use a "Software as a Service" (SaaS) model to separete rommendations from the rest of the platform.

Although there exist several commercial approaches to general SaaS recommender systems, like Strands[3] or Directed-Edge[4] there exist no literature focusing on the system's aspects of general recommender systems. In this paper we describe our experience developing our general purpose multi-algorithm recommender system, which is currently being used to personalize our products and services at BrainSins and will be used as experimental platform to compare and evaluate our further research on recommender systems.

In the next section we describe the main challenges we found in order to develop a general purpose multi-algorithm recommender system. In section 3 we describe the general architecture of the system focusing on the elements that allowed us to solve the main issues, and in section 4 we focus on our recommender system API, which enables all our products and services to interoperate with the recommender system. Section 5 describes the next research works we will face, and section 6 concludes the paper.

2. MAIN CHALLENGES FOR A GENERAL PURPOSE MULTI-ALGORITHM RECOMMENDER SYSTEM
When designing a general purpose multi-algorithm recommender system, we found several challenges that had to be addressed in order to develop a useful system.

- Interoperability: Recommender systems are usually created to access a specific database that uses a well-known data structure. However, since our system had to offer the possibility of being integrated into several different existing platforms, we had to deal with the problem of accessing sources with different data structures.

[1]http://www.wipley.es

[2]http://www.flickrbabel.com
[3]http://recommender.strands.com
[4]http://www.directededge.com

PRSAT 2010. Copyright is held by the author/owner(s). Workshop on the Practical Use of Recommender Systems, Algorithms and Technologies (PRSAT 2010), held in conjunction with RecSys 2010. September 30, 2010, Barcelona, Spain.

- Configurability and easy of use: As one of the main goals of this recommender system was to be able to manage several recommender algorithms to provide a particular recommendation, the system had to be highly configurable. In these cases usability may become a problem difficult to solve.

- Performance: Recommendations must be served in real time, but users do not tolerate an increase in page downloading time. However, when a system is designed with a high degree of configurability, there are always some issues that slow down performance. In our particular case, the new system had to be as effective as the recommender we already had on production.

- Disk usage: A highly configurable system also presents disk space problems, due to the fact that data representation cannot be optimized.

- Scalability: When applied to the web, the number of items to be recommended and the number of users to receive those recommendations often grow exponentially. Therefore, a recommender system needs to be scalable in order to grow at the same rate. However, being scalable also means that the system should somehow hide the way it scales, so there should be no need to code or rewrite configuration parameters in the products and services that access the recommender system.

After analyzing possible solutions for those challenges and start designing and developing the system, we thought that a key fact to achieve most of them was to conveniently separate recommendation process from the interface with the clients, so interoperability became our first focus. Our goal was to design a way to easily integrate the recommender system while keeping recommender complexity hidden from its clients, so we designed a REST API. REST APIs are easy to use and to integrate in any product and service on the web, and many developers are familiar to it. Furthermore, REST API helps to hide the complexity of the recommender system, and allows to transform the REST petitions into more complex data structures needed to maintain the performance and scalability of the recommender system.

To face configuration and performance issues we designed a back-end architecture that enables us to define recommendation algorithms as software modules that can be adapted to any domain required by clients. The REST API also allowed us to introduce configurability elements as optional parameters in the petition.

3. GENERAL ARCHITECTURE

The input to our system are API requests, which can be classified as online or batch:

- Online requests, which must be handled in real time. Their processing can't be delayed, because users are waiting for a response. They are also utilized to update the profile for new users and begin to provide them with recommendations.

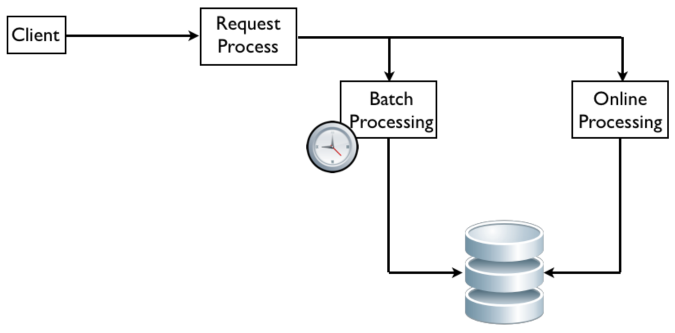


Figure 1: The request processor evaluates if a certain API request needs to be run online or it can be batched.

- Batch requests, which may be stored and processed only at given time periods. Now requests processing can be delayed and attended when the system is not at full capacity. These requests are used to upload the initial data from a client and also to update information concerning users with a wide user profile.

Figure 1 shows how API requests are being processed. Each API request generates an HTTP request to a certain endpoint where the Request Processor evaluates it and determines whether it must be processed at that moment or it can be delayed until more requests reach the system (for a more optimum processing) or until certain batch process is programmed to be run.

Requests can also be classified as update or retrieval:

- Retrieval requests just ask the system to return some kind of information, such as a recommendation.
- Update requests have the objective to update the profile of the source user.

When an update request begins to be processed there are two steps that must be taken to produce recommendations for the user. As we wanted to be able to process several types of recommendations (collaborative filtering, content based, social recommendations), the system had to be general enough to process data in several ways. So we defined those steps in a way that enabled the use of any possible recommender algorithm (see Fig. 2):

- Update user profile. This can be done by re-calculating simmilarity with other users, re-calculating trust or updating a content-based profile.
- Update user recommendations. This step uses the values obtained from the previous task as input for the recommendation algorithm and produces a new rank of recommendations for the user.

As a first approach we coded 3 different recommender algorithms:

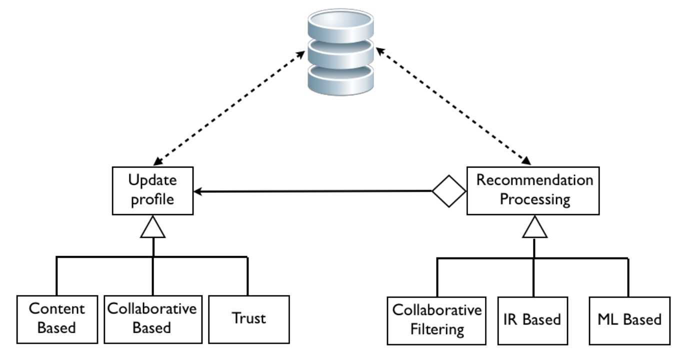


Figure 2: Main modules of the recommender system.

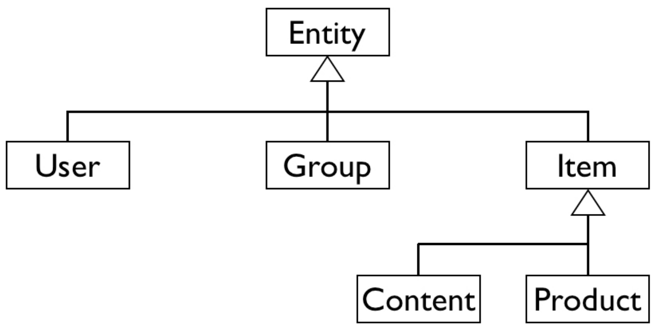


Figure 3: Entities describes every possible data used by the recommender algorithms.

- Collaborative filtering. A standard method that produces collaborative recommendations when using a collaborative similarity measure, or social recommendations when using trust.
- Information retrieval based recommender used for content based recommendations.
- Machine learning based recommender also used for content based recommendations.

From the data point of view, the system only stores entities and relationships among entities. Each possible data point (user, group, content or product), is represented by an entity (see Fig. 3). Relationships (see Fig. 4) among entities represent actions (ratings, comments, reviews) or behaviors (buy a product, pageview, joining a group). This data representation allows enough degree of data abstraction for the interoperability, and it also stands near enough to data representation used by the different recommender algorithms.

4. INTEROPERABILITY: A GENERAL API

A Representational State Transfer (REST) API [2] is a style of software architecture for distributed systems like the Web where clients initiate requests and the servers process requests and return appropriate responses. Requests and responses are built around the transfer of resources. REST is described in the context of HTTP, although it can be used in other contexts.

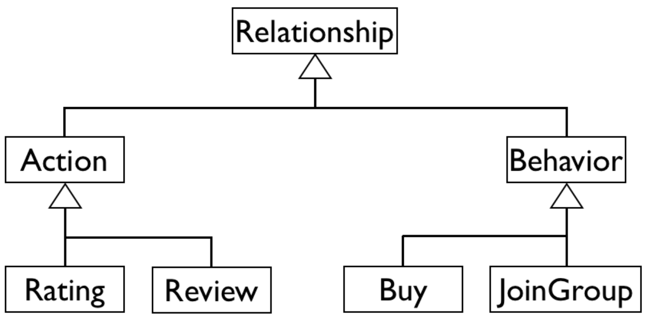
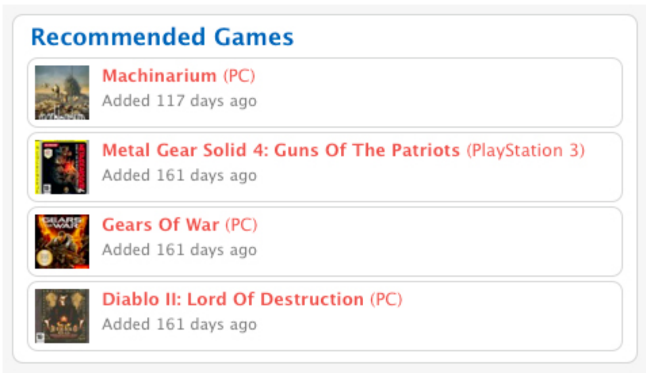


Figure 4: Relationships describes actions and behaviors.


Recommended Games

Machinarium (PC)
Added 117 days ago

Metal Gear Solid 4: Guns Of The Patriots (PlayStation 3)
Added 161 days ago

Gears Of War (PC)
Added 161 days ago

Diablo II: Lord Of Destruction (PC)
Added 161 days ago

Figure 5: Integration of the recommendations in Wipley, our social network for videogamers.

A client request is described by an HTTP request to a certain resource. For instance, for one of our domains, we may want videogames recommendations for a certain user (user id=2). With our REST API, the client must make an HTTP request to a certain URI[5]. The first part of the URI describes the endpoint where our API server is running[6]. Next part shows that a recommendation (and the recommendation's type) is requested ('/recomm') for a certain entity type ('/player') considering a relation of ownership with another entity type ('/has/videogame'). The user id is '2' and the response format is '.xml'. The server must return an XML file containing the recommendations.

Figure 5 shows our integration on Wipley, our social network for videogamers, where XML responses are processed with PHP in order to generate a more visual interface.

In Wipley we have configured our system to serve different types of recommendations for a user, such as products, users and groups. They can be requested just modifying the URI. In order to update the information that the recommender system manages internally, we have designed a XML based specification, which allows to update information about any entity (data points, see Fig. 6) or action (relationships among entities, see Fig. 7).

[5]http://webservices.brainsins.com/api/recomm/player/has/videogame/2.xml
[6]http://webservices.brainsins.com/api

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<recsins version=”0.1”>
     <entity id=”product01”>
          <property name=”entityClass”>product</property>
          <property name=”productName”>Call of Duty:
Modern Warfare 2</property>
          <property
name=”productClass”>videogame</property>
          <property name=”price”>12.99</property>
     </entity>
</recsins>
```

Figure 6: Example XML containing a description of an entity (videogame).

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<recsins version=”0.1”>
     <action id=”action0000001”>
          <property name=”timestamp”>2010-01-01 T
10:43:21 UTC</property>
          <property name=”origin”>user01</property>
          <property name=”destination”>item01</property>
          <property name=”actionType”>rating</property>
          <property name=”actionValue”>3</property>
     </action>
</recsins>
```

Figure 7: Example XML containing a description of an action (user rating a videogame).

5. FUTURE WORK

We have developed this general purpose multi-algorithm recommender system and we have integrated it into Wipley with great results, given that videogames recommendations are considered by our users as one of the top features. Our future work is defined in several lines:

- Test the performance of the general recommender system in terms of running time and disk usage. Our first impressions make us think that generalizing the recommender system does not introduces computational overhead, since processing the REST requests represents only a 3-4% of the total running time of a recommendation, but we need to run a larger set of experiments to evaluate the final performance.

- Scalability. As the REST API give us a lot of independence of the clients from the implementation, we are actually re-coding the recommender algorithms through a map-reduce perspective using Apache Mahout[7], which will allow the platform to be really scalable.

- Experimental platform. One of our main goals is to obtain an experimental platform to test the performance of our recommender systems implementations. We are starting to measure several metrics related to web analytics such as CTR. We expect to obtain real feedback about what recommender algorithms and settings works better for different domains. We are also developing a web based backend which will allow us to define the experiments and measure several aspects related to the effectivity of the recommendations.

[7]http://lucene.apache.org/mahout/

There are no references in the recommender systems literature describing a general purpose recommender system being used to test several recommender algorithms within several domains with the aim to produce an extensive experimental comparison of recommender algorithms. We have developed a general recommender system which encapsulates several recommender algorithms (collaborative filtering, content based recommender, and social recommender) with the main purpose of producing an extensive comparison of recommender algorithms in a real environment. We have also integrated this general recommender system in a real social network that represents an experimental setup with real users in real conditions.

6. ACKNOWLEDGMENTS

The research described in this paper has been partially supported by the Madrid autonomous region, IV PRICIT, S-0505/TIC/0267.

7. REFERENCES

[1] O. Celma and P. Lamere. If you like the beatles you might like...: a tutorial on music recommendation. In *MM ’08: Proceeding of the 16th ACM international conference on Multimedia*, pages 1157–1158, New York, NY, USA, 2008. ACM.

[2] R. T. Fielding. *Architectural Styles and Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[3] J. Golbeck. Tutorial on using social trust for recommender systems. In *RecSys ’09: Proceedings of the third ACM conference on Recommender systems*, pages 425–426, New York, NY, USA, 2009. ACM.

[4] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *RecSys ’09: Proceedings of the third ACM conference on Recommender systems*, pages 53–60, New York, NY, USA, 2009. ACM.

[5] J. F. McCarthy. The challenges of recommending digital selves in physical spaces. In *RecSys ’07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 185–186, New York, NY, USA, 2007. ACM.

[6] O. Phelan, K. McCarthy, and B. Smyth. Using twitter to recommend real-time topical news. In *RecSys ’09: Proceedings of the third ACM conference on Recommender systems*, pages 385–388, New York, NY, USA, 2009. ACM.

[7] J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *EC ’99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, New York, NY, USA, 1999. ACM.

[8] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 1039–1040. ACM, 2006.