Introducing Collaborative Filtering into an Agent-Based Travel Support System

Mateusz Kruszyk
Cognifide, Poznań, Poland

Maria Ganzha, Maciej Gawinecki, Marcin Paprzycki
System Research Institute, Polish Academy of Sciences, Poland

Abstract

Our recent work is devoted to the development of an agent-based travel support system in which personalized information is delivered to the user. Thus far we have focused our attention on personalization based on behavior of a given user. In this note we conceptualize how collaborative content filtering can be introduced into our system.

1. Introduction

The *Travel Support System* (*TSS*) is an academic project aiming at developing an agent-based system to provide travelers with personalized information (see, [3]). This work was inspired by the following scenario. *Hungry foreign tourist arrives to an unknown city and seeks a nice restaurant serving cuisine that she likes. Internet, searched for advice about restaurants in the neighborhood, "recommends" mainly establishments serving steaks, not "knowing" that the tourist is a fanatic vegetarian.* This scenario determines the following functionalities of the system (this list is not exhaustive, but points to features that are pertinent to this note):

- *Content delivery*. Content should be delivered in a browser-processable form, i.e. HTML, WML etc. and match the query. No other assumptions about content delivery methods should be made [5].
- *Content personalization*. Delivered content should be personalized according to the *user-profile* to avoid situations like the one presented above.
- *Adaptation of personalization*. User habits can change, therefore her profile should be adapted on the basis of her activities recorded by the system.

In fact, these functionalities are realized only by a part of the *TSS* depicted in Figure 1, called the *Content Delivery Subsystem* and are considered here, while the remaining parts of the *TSS*, responsible for data collection and management, have been described in [3]. This latter work (and references collected there) should be consulted for the remaining details concerning the *TSS*.

In the proposed system we use RDF to semantically demarcate content, (to meet the requirements of Semantic Web applications [19, 20]), while the Jena framework [10] is utilized to persist ontologically demarcated data. When conceptualizing the system, the Model-View-Controller design pattern [18] was applied for clear separation between pure data (model) and its visual representation (view). Let us illustrate the work of the system through a simple scenario.

The user is looking for a Vietnamese restaurant in Poznań. To find it she fills-out a web-form and sends it as a request to the system. The *Proxy Agent* (*PrA*), acting as a gateway between the non-agent user environment and the agent-based system (see [6] for details), receives the HTTP request and forwards it to the system, precisely, to the *Session Handling Agent* (*SHA*). The latter plays the role of the Controller in the MVC and thus delegates the user *Personal Agent* (*PA*) to realize the incoming request. The *PA* prepares an answer (here called accordingly model of data) by: (1) asking the *Restaurant Service Agent* about restaurants matching the template given by the user in her request, and (2) filtering returned restaurants according to her profile. Feature-based filtering technique is used at this stage. As a result, the *PA* returns sorted list of recommendations to the *SHA*, which requests that the *View Transforming Agent* (*VTA*) provides a view of a given model in a form of a document (HTML/WML/TXT), that can be rendered by the user device. Generated view is returned to the *SHA* and then forwarded back to

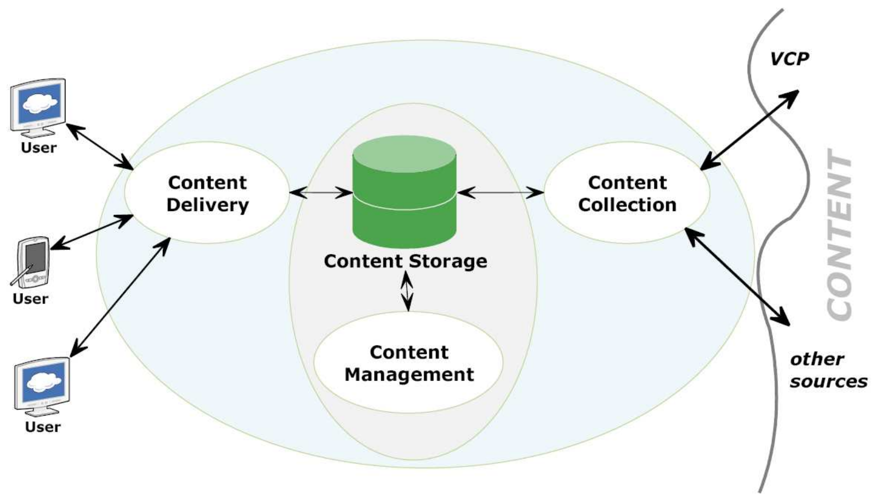

Figure 1. Travel Support System general architecture.

the user through the *PrA*. Let us also note that besides realization of current user request the *SHA* logs: (a) user request, (b) system response, and (c) user reaction to this response (both the implicit feedback and, if provided, the explicit feedback) and forwards logged information to the *Profile Managing Agent* (*PMA*). The latter is responsible for initializing and learning / adapting user profile on the basis of this data. It also provides the user profile for the *PA*.

2. Content personalization

In our earlier work [7, 8] we have considered content personalization based on user profile represented as an overlay "on top" of the travel ontology. Specifically, we have extended classical results presented in [2] to data stored in an ontologically demarcated format. Furthermore, we have adapted techniques based on feature-based filtering for the data model used in our system. Specifically, only objects (here, restaurants), whose features cover features of objects that user has seen, clicked and looked for, are recommended. To achieve this user feedback (both implicit—formulated queries, clicked recommendations etc., and explicit—rated object) is stored in the *History model* and processed to learn and adapt user profiles. Each user profile in normalized against the population behavior. For example, if the user often clicks on Chinese restaurants, while the whole population she belongs to behaves similarly, than Chinese restaurant will not be rated as significant. Finally, to solve the cold start problem we have created six culinary-behavior stereotypes to be used when no initial data about the user is available [8].

2.1. Collaborative filtering

Let us now discuss how collaborative filtering can be introduced into our system. Collaborative filtering should be understood as any situation in which recommendation depends not only on the profile of a given user, but involves also "advice" from other entities. Some well-known reasons for considering collaborative recommendation techniques are: (1) can be used to solve the cold start problem [14], (2) often is better in providing user with the correct recommendation (especially when a limited number of individual data points are available) [14], and (3) in this way we are completing the pyramid of recommendations, where we can represent interests of individuals, groups at various levels, and the population as a whole [17]. There are multiple ways in which "suggestions from others" can be utilized (observe, for instance, how Amazon.com suggests CD's on the basis buyer behavior clustering). In this work we utilize collaborative filtering based on the metaphor: I consider valuable what is recommended to me by people whom I trust. This technique is often named *trust-based recommendations* and to instantiate it we utilize results presented in [13].

Obviously, the idea of collaborative filtering is very well suited for agent systems. Let us envision a simple scenario, where a given agent, confronted with a request from its user is asking other agents about their recommendations. Furthermore, over time it develops a list of agents that it considers "trusted" (see, for instance, [15]). In this way trust becomes the relation that "selects" individuals that are collaborated with to provide user with recommendations. However, this approach will not work directly in our system (and it is our system that we would like to introduce collaborative filtering into; and thus have to take its design and functioning into account). As it has been discussed in [12], due to architectural considerations, *Personal Agents* do not "live" on user devices, but are part of the system infrastructure itself. As a result they are "alive" only when user is logged into the system. At the same time, in other agent systems [16], it is assumed that personal agents are available to be communicated with in a semi-permanent fashion. In other words, there in a typical scenario it is assumed that agents are mostly available and unavailable only sporadically, while in our system the situation is exactly the opposite: agent are mostly unavailable, while being available only periodically.

Therefore we had to design a different approach to collaborative content filtering. First, let us observe that since all *PA*s are instantiated within the system each one of them "knows" which other agents that currently co-exist with it (in a JADE-based system [9] this service is provided by the *RMA* agent). Therefore, as long as at least one other *PA* is instantiated, it is possible to ask it for a recommendation. However, notice a potential problem when k agents start asking for recommendation from all $(k-1)$ other agents. Each such question results in $2(k-1)$ messages (containing recommendations). Furthermore, each response may consist of a considerable number of suggestions, and thus become a large message. Overall, asking for recommendations from all available agents could be detrimental to the performance of the whole system. Therefore, we have decided that a request for a recommendation can be send only to a specific maximum number of agents (k_{max}). This number has to be established experimentally (similarly to the experiments reported in [1]), on the one hand maximizing it, on the other keeping system responsive by limiting the total number of messages.

Second, in a trust-based recommender, recommendations are weighted according to the "trust" in their providers. Observe that in the standard scenario the "advisory panel" is assumed to be mostly available. It is thus enough to collaborate with a small number of trusted peers and store information about them and their recommendations. In our system it may be often the case that no "known/trusted peer" is available. Obviously, we could assume that when needed such peers could be instantiated by the system (recall that they are all entities within the system) to respond to the query (and then disposed off), but this solution seems to be a serious waste of resources. Envision that for each user query about 20-30 agents are instantiated and then immediately killed. It would be also possible to provide the requester with user profiles of its trusted peers, but this suggestion is highly questionable due to the necessary privacy of user information (note that some users may not want to be a part of collaborative advising). Another possible approach would be to store information about all agents that were ever interacted with and thus increase potential of finding in the system active agents that we know at least something "positive" (or "negative," if we were to extend our filtering model to include "counter-advisers") about. However, if we assume that for each agent we store only the trust value. Then for n agents, each agent would store their $n-1$ ID's and $n-1$ trust values; overall, $O(n^2)$ values. While not a problem for a few hundred agents, this soon grows to a very large number. Therefore, we decided to limit the size of the "advisory panel" (again, to be established experimentally).

Finally, to support needs of collaborative filtering, we have decided to introduce the *Collaborative Filtering* (*CF*) agent and place it between the *SHA* and the *PA*. Its role is to know (a) *PA*s available in the system, (b) how many agents can be queried, (c) who are the advisory agents; as well as (d) manage trust information, (e) query agents, and (f) collate their responses.

As a result the following scenario materializes. When a given *PA* wants to ask other agents about their recommendation, it sends a request to the *CF* agent, which checks if any member of the advisory panel is available (e.g. 5 of them) and contacts them. Then the remaining available quota of agents (e.g. 25, for $k_{max} = 30$;) is selected randomly from the pool of agents available in the system (e.g. 70). Finally, the *PA* is also contacted back and its recommendations taken into account. All responses are then collated and sent back to the *PA* which follows the above described scenario to present them to the user (see Figure 1). Let us recall from [4], that the suitability of a given object (its *temperature*) is represented as a real number from the interval $[0, x]$ (where x is a constant; typically $x = 1$). Therefore, each agent presents a list of recommendations and their temperatures. Then, recommendations of trusted agents are combined as follows (see [13]); for each object i appearing in the set of all responses from all trusted advisers:

$$REC_i^T = \frac{\sum_{j=1}^{m_i} temp_{i,j} * T_j}{\sum_{j=1}^{m_i} T_j}, \qquad (1)$$

where T_j is the trust value assigned to a given (j-th) agent, while $temp_{i,j}$ specifies the temperature that agent j associates with object i. The procedure for the remaining agents is different because we have no trust value associated with them. Modifying the above approach

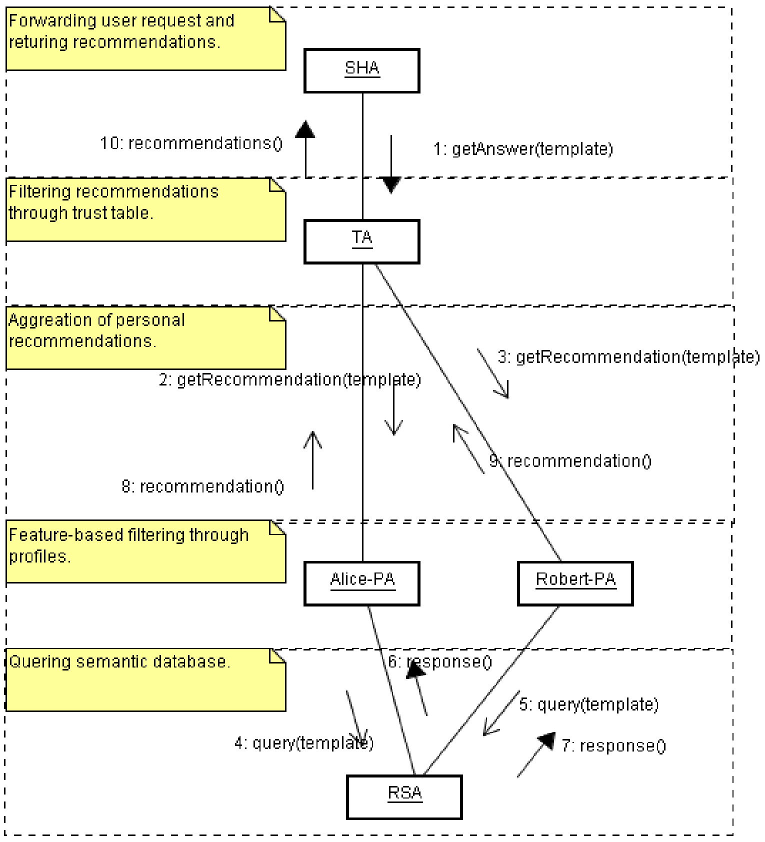
Forwarding user request and returing recommendations.
SHA
10: recommendations()
1: getAnswer(template)
Filtering recommendations through trust table.
TA
Aggregation of personal recommendations.
3: getRecommendation(template)
2: getRecommendation(template)
9: recommendation()
8: recommendation()
Feature-based filtering through profiles.
Alice-PA
Robert-PA
Quering semantic database.
6: response()
5: query(template)
4: query(template)
7: response()
RSA

Figure 2. Trust-based filtering.

slightly we combine their recommendations as follows:

$$REC_i^{NT} = \frac{\sum_{j=1}^{m_i} temp_{i,j}}{m_i} \qquad (2)$$

In this way we obtain separate weighted average of the strength of recommendations put forward by trusted and other agents. Furthermore, these responses have to be combined with that obtained from the *PA* (REC_i^{PA}). Note that we have to try to prevent a group of agents from colluding to promote a given object (e.g. a restaurant). Therefore we suggest that the recommendation about object *i* be calculated as follows:

$$REC_i = \alpha * REC_i^{PA} + \beta * REC_i^{T} + \gamma * REC_i^{NT}, \qquad (3)$$

where α, β and γ are scaling constants that establish strength of each of the three components, while $\alpha+\beta+\gamma = 1$. For instance, to assure that colluding agents cannot influence the "vote" it is enough to specify $\alpha = 0.6$, $\beta = 0.3$ and $\gamma = 0.1$. Note that if a group of not-trusted agents colludes with trusted agents they can achieve their goal. However, if the user is then "unhappy" and expresses negative opinion about their recommendation, then the trust value of trusted agents that made such a recommendation decreases (possibly even considerably) and subsequent collusions will remove it/them from the pool of advisers. The resulting set of recommendations is then sorted and presented to the user starting from the most highly recommended. Note that in the case when no trusted agents are logged into the system and cannot be asked for their opinion, or in the case of only trusted agents being asked about their opinion, one of the recommendations: $T\_Recommendation_i$ or $NT\_Recommendation_i$, will be 0 for all *i*. This does not change the way that our system works and using the proposed formula we can still present user with a ranked set of recommended objects.

At this stage the user provides the system with an explicit or implicit feedback (or both) about the recommended objects (e.g. restaurants). This feedback will be used to adjust user profile (as described in [4]) and to adjust trust values of known peers. The new trust value is calculated on the basis of the following function [13]:

$$T_{n+1}(x) = \phi T_n(x) + (1-\phi) r_{real} \qquad (4)$$

where $T_{n+1}(x)$ and $T_n(x)$ are the new and the old values of trust of a given agent in agent x; ϕ is a parameter of the system that manages the evolution dynamics of trust (proposed by Jonker and Treuf in [11]); and r_{real} is a real interest of user in the object (evaluated on the basis of feedback [13]).

Observe that in the proposed system, we utilize a specific limited number of trusted peers; consisting of agents with currently highest values of trust. Let us now note that function (4) can be used to evaluate trust not only of agents-advisers, but also of unknown agents (their initial trust value is $T_n(x) = 0$). As a result, if the best scoring unknown agent (Y) scored higher than the lowest scoring member of the panel (Z), then Y will replace Z in the panel. In this way, by introducing new agents to the group of advisers, we can naturally adapt its composition to users' changing profile. However, to avoid drastic changes that may not be warranted, each time only a single replacement is made.

3. Concluding remarks

In this note we have briefly (due to the lack of space) outlined how collaborative filtering proposed by Montaner in [13], can be adjusted to our agent-based travel support system to fit its design characteristics. Currently we are improving the presented method, so the each PA does not ask the RSA the same query about recommended objects (see Figure 2), but rather the RSA answers this query once, and then each PA uses this answer to filter out its own recommendation. Furthermore, we consider utilization of the threshold trust value to manage the composition of the advisoy panel.

References

[1] K. Chmiel, D. Tomiak, M. Gawinecki, P. Karczmarek, M. Szymczak, and M. Paprzycki. Testing the efficiency of jade agent platform. In *ISPDC '04: Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04)*, pages 49–56, Washington, DC, USA, 2004. IEEE CS.

[2] J. Fink and A. Kobsa. User modeling for personalized city tours. *Artif. Intell. Rev.*, 18(1):33–74, 2002.

[3] M. Ganzha, M. Gawinecki, M. Paprzycki, R. Gąsiorowski, S. Pisarek, and W. Hyska. *Semantic Web Technologies and eBusiness: Virtual Organization and Business Process Automation*, chapter Utilizing Semantic Web and Software Agents in a Travel Support System. Idea Publishing Group, 2006.

[4] M. Gawinecki. User modelling on a base of interaction with WWW system. Master's thesis, Deparment of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, 2005.

[5] M. Gawinecki, M. Gordon, P. Kaczmarek, and M. Paprzycki. The problem of agent-client communication on the internet. *Scalable Computing: Practice and Experience*, (6(1)):111–123, 2003.

[6] M. Gawinecki, M. Gordon, P. Kaczmarek, and M. Paprzycki. The problem of agent-client communication on the internet. *Parallel and Distributed Computing Practices*, 6(1):111–123, 2003.

[7] M. Gawinecki, M. Gordon, M. Paprzycki, and Z. Vetulani. Representing users in a travel support system. In H. K. *et. al.*, editor, *Proceedings of the ISDA 2005 Conference*, pages 393–398, Los Alamitos, CA, USA, 2005.

[8] M. Gawinecki, M. Kruszyk, and M. Paprzycki. Ontology-based stereotyping in a travel support system. In *Proc. of the XXI Fall Meeting of Polish Information Processing Society*, pages 73–85. PTI Press, 2006.

[9] JADE—Java Agent DEvelopment Framework. `http://jade.tilab.com/`, 2005. an Open Source platform for peer-to-peer agent based applications.

[10] Jena—A Semantic Web Framework for Java. `http://jena.sourceforge.net/`, 2005.

[11] C. M. Jonker and J. Treur. Formal analysis of models for the dynamics of trust based on experiences. In *MAAMAW '99: Proc. of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 221–231, London, UK, 1999. Springer-Verlag.

[12] M. Kruszyk, M. Paprzycki, and M. Ganzha. Pitfalls of agent system development on the basis of a travel support system. In W. Abramowicz, editor, *Proceedings of the BIS 2007 Conference*, 2007.

[13] M. Montaner, B. López, and J. L. de la Rosa. Developing trust in recommender agents. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 304–305, New York, NY, USA, 2002. ACM Press.

[14] M. Montaner, B. López, and J. L. de la Rosa. A taxonomy of recommender agents on the internet. *Artif. Intell. Rev.*, 19(4):285–330, 2003.

[15] M. Montaner, B. Lopez, E. del Acebo, S. Aciar, and I. Cuevas. *AgentCities Agent Technology Competition*, chapter IRES: On the Integration of Restaurant Services, pages 6–8. February 2003.

[16] S. Nesbitt. Collaborative Filtering on the Web: An agent-based Approach (Literature Review), 1997.

[17] C. E. Nistor, R. Oprea, M. Paprzycki, and G. Parakh. The role of a psychologist in e-commerce personalization. In *Proceedings of the 3rd European E-COMM-LINE 2002 Conference*, pages 227–231, 2002.

[18] V. Ramachandran. Design Patterns for Building Flexible and Maintainable J2EE Applications. `http://java.sun.com/developer/technicalArticles/J2EE/despat/`, styczeń 2002.

[19] Resource Description Framework (RDF). `http://www.w3.org/RDF/`, 2005.

[20] Semantic Web Activity Statement. `http://www.w3.org/2001/sw/Activity`, 2001.