Reducing the Complexity of Recommender Systems Development

M. Elammari and R. Elfrjany

Abstract—**At present, recommender systems are emerging as a growing application and research field in several domains of computing research, from artificial intelligence to information systems. In the past, these systems have been primarily used to reduce information overload and to identify the items that are of interest to the user more precisely. Recommender systems development is a complex task, on the other hand, abstraction and modularity are powerful concepts for handling the complexity of software development, especially if the problem domain is particularly complex, changeable, or large scale. This paper presents our attempt to reduce the complexity of recommender systems development via using software architecture concepts as well as multi agent system.**

Index Terms—**Recommender systems (RS), multi-agent system (MAS), software architecture.**

I. Introduction

Since the emergence of the first paper on collaborative filtering in the mid-1990s, recommender systems (RS) became one of the most active research areas, and much research has been done resulting in numerous recommendation algorithms including the most basic − content-based filtering, collaborative filtering, and knowledge-based approach − and more advanced techniques, such as case base reasoning technique. As all currently available recommender systems have strengths and weaknesses, numerous research studies have attempted to develop techniques that would overcome the existing limitations by combining available techniques in different ways [1], [2], [3] .

In this paper, we present a powerful tool to reduce the complexity in recommender systems development via the following:

- Using software architecture concepts that provide a useful abstraction [4], as strategy to reduce the complexity of developing the recommender systems.
- Partitioning the overall problem into a number of simpler components, which would be easier to develop and maintain, by using an agent-based approach.

The paper is organized as follows: Section 2 presents a brief background of recommender systems and agent technology. Section 3 introduces related work. Section 4 describes our contribution in develop recommender systems by using multi-agent systems (MAS). Section 5 presents a case study to demonstrate the practical implementation of the proposed work. Finally, section 6 is the conclusions of this work.

Manuscript received July 25, 2012; revised September 4, 2012.

Mohamed Elammari is with the Department of Software Engineering at the University of Benghazi.

Rabeia Elfrjany is with the currently working towards Ph.D. degree in software engineering.

II. Background

A. Recommender Systems

Each recommender system consists of three basic components. These components are [5], [6], [7]:

- Items to be recommended: such as books, movies, music, courses, web pages...etc.
- Target consumer preference profile: this profile is created after the user preferences are identified through various techniques; the process is also called user modeling.
- The recommender algorithm - also called recommender methods or techniques: this component is the mechanism that generates recommendations.

There are many approaches (methods, approaches or techniques) in recommender systems field, each of them has strengths and weakness points so many researchers attempt to solve the various limitations of current recommender systems by combining some approaches in hybrid systems.

The basic recommendation approaches are content-based filtering (CBF), collaborative filtering (CF), and knowledge-based approach. Collaborative Filtering Approach attempts to simulate the collaboration between the users aimed at sharing recommendations. Thus, it recommends items to the consumer based on the matches between the gathered data about his/her preferences and those of other users in the system. Most of collaborative systems apply the nearest neighbor model to compute the recommendations [8], Examples of Collaborative recommender systems are Eigentaste [9], and GroupLens [10]. Content-Based Filtering approach generates recommendations based on the correlation between the items' content and user's preferences; in other words, these systems recommend items that are similar to previous user preferences [11]. Examples of content-based recommender systems are described by Syskill and Webert [12], and a recommender system for the dspace open repository platform is given in [13].

The knowledge-based approach exploits its knowledge base of the items domain for generating recommendations to a user, and then reasoning about what items meet user's requirements [3]. An Example of knowledge-based recommender systems is Entree [14]. Each of the previous approaches has its own shortcomings, when considered individually, but some solve many shortcomings found in the other techniques; for example, pure collaborative systems solve all content-based system's problems, and content-based systems solve all collaborative system's problems. Hence, to utilize these advantages, many researchers have combined two or more recommendation approaches to reduce the problems associated with individual techniques [8], [15], [16].

Burke [15] classified hybrid recommender approach into four groups: Weighted, Mixed, Switching and Feature Combination. Examples of hybrid recommender systems are WEBSELL [6], and hybrid recommender systems for electronic commerce [7].

B. *Agent Technology*

Recently, the agent technology has become one of the hottest research topics, promising to develop flexible and intelligent systems; thus there are a number of literature sources where researchers in mainstream computer science broadly discuss the agent technology. The majority of publications that relate to agent technology categorized agents into computational, biological, and robotic agents, and further subdivide computational agents into software agents and artificial life agents [17], [18]. agent technology incorporates structures that enable representing knowledge, achieving goals, interacting with the environments, and responding to unexpected changes, which, occur in its environment [19]. These structures could provide a significant advantage if used in the recommender systems' field, where they can make the recommender systems capable of taking personal preferences into account, intelligently aggregate opinions and relationships from heterogeneous sources and data, make systems scalable, protect privacy, create open systems, and provide recommendations with minimal user involvement [20]. Multi-agent system can be seen as "loosely coupled network of agents that work together as a society aiming at solving problems that would generally be beyond the reach of any individual agent" [21]. There are many benefits of using multi-agent system for developing software systems, such as those given in [22], [23], [24].

III. Related Works

The available literature in the field of recommender systems [5], [7], [25], [26], [27] reveals many studies conducted in this domain. However, the studies vary in nature and some attempt to improve one of the recommender system components at the expense of other components. Alternatively, they focus on the use of new methods to be integrated into recommender systems, without attempting to solve the existing problems or consider quality of recommendation in their works. Others focus on the improvement in the recommendation algorithms. Chaptini [5] investigated the use of discrete choice models as a radically new technique for giving personalized recommendations. Moreover, he presented a software package that allows the adaptation of generalized discrete choice models to the recommendation task. The work focused on user modeling, and was not intended to overcome the weaknesses of any recommender techniques. As a test case for investigating the effectiveness of the work, he explored the application of discrete choice as a solution to the problem of recommending academic courses to students. Sotomayor et al. [25] investigated integrating singular value decomposition (SVD) with collaborative filtering approach to enhance CF approach by reducing the dimensionality of recommender systems databases. The work was aimed at supporting and facilitating a demanding customer in the task of searching for items to purchase by bundling a set of products and creating a feedback list. The system allows users to identify their own recommendation list and it personalizes the shopping experience by aggregating new items that were unknown to them. Marivate et al. [26] took MAS approach to solve the problem of recommending training courses to engineering professionals; the system reduces the need for users to search a wide variety of sources of information as well as dispensing with any need to monitor these sources for changes and updates. The recommendation system was built as a proof of concept and is limited to the electrical and mechanical engineering disciplines. Collaborative filtering recommendation was implemented using intelligent agents.

The agents work together in recommending meaningful training courses and updating the course information. The system used individual user profiles and keywords from courses to rank courses. The limitation of this system is in the use of the supervised learning neural network for ranking. Thus if a new discipline was to be added, the neural network would need to be retrained with new survey data. Mukherjee et al. [27] built software agent that employed user modeling techniques to facilitate information access and management tasks. This system focused on personalized recommendation based on user preferences that were explicitly specified or inferred from interactions. The system generates recommendations from stored preferences by using both voting schemes and text-based learning. The authors have developed a movie recommender system that caters to the interests of users to demonstrate the applicability of their technique. The interactive agent in this system learns a user model by gaining feedback about the recommended movies from the user. Tran and Cohen [7] presented architecture for designing a hybrid recommender system that combined collaborative filtering and the knowledge-based approach. Their architecture used an interactive interface agent −"Interface agents are computer programs that employ Artificial Intelligence techniques in order to provide assistance to a user dealing with a particular computer application" [19, p. 258] − to switch between the two approaches; depending on the current service. The agent also coordinates the operations of the two subsystems to make the best possible recommendations to users. As any knowledge-based systems, this system suffers from the drawback of requiring knowledge engineering, with all of its attendant difficulties.

IV. Proposed Approach

Based on the core components of recommender systems (see Section 2. 1), and on the advantages of using multi-agent technologies in software development (as given in [22], [23], [24]), our study presents a general architecture supported by agent technology, using switching hybrid method to switch between three individual recommendation algorithms − collaborative filtering, content-based and knowledge-based − to reduce the shortcomings of these algorithms.

Fig. 1 illustrates the proposed architecture and presents its key components:

- Preferences gathering
- Recommendation generator
- Items collector

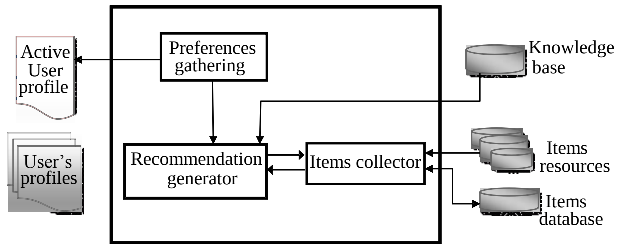


Fig. 1. A general architecture for building recommender systems.

The generic functions of each component in the presented architecture are supported by agent technology. Each agent is responsible for a relatively simple task, but cooperatively they present the powerful recommendation. Fig. 2 shows the conceptual overview of the presented architecture to illustrate its component configuration and interactions in detail.

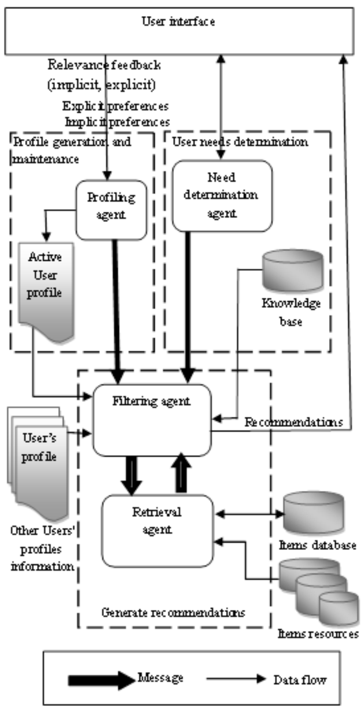
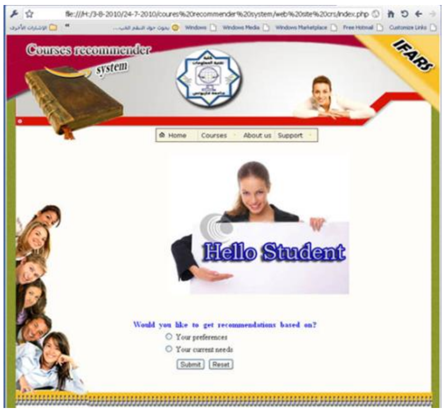


Fig. 2. Conceptual overview of the proposed architecture.

1) *Preferences gathering component*

This component provides the main graphical user interface (GUI). It functions as an intermediary between the user and the system, and is responsible for interaction with the users to collect their preferences and display the recommendation. Gathering user preferences is the first step in making the recommendation. If the system gathers sufficient information which enables creation of precise knowledge of user needs and preferences, the recommendations will be a closer match to the user requirements. A user interface design depends on the nature of the recommender system, but it should be designed to be easy to use; hence the designer should use suitable technology with good graphical capabilities, such as a Java Server Pages. Java Server Pages provide an efficient and simplified way to build dynamic graphical interface [25]. In the presented architecture, the graphical interface should select one of the following two subcomponents: (1) profile generation and maintenance component that will be activated when the user requires the system to produce recommendation based on his/her preferences; (2) user needs determination component that will be activated when the user requests recommendation based on current needs. An example of the scenario whereby the user requests a recommendation based on his/her current needs is *a user wants an item, such as book to give as present to his/her friend* [7]. To determine which component to employ, the system can simply ask the user if he/she wants the recommendation to be based on his/her preferences or not. The system can make the enquiry via the GUI. An example of such interface is shown in Fig. 3.



Would you like to get recommendations based on?
- Your preferences
- Your current needs

Submit Reset

Fig. 3. Determine user requirements.

a) *Profile generation and maintenance component*

This component (Fig. 4) is responsible for creating and updating an active user profile. It contains a *profiling agent (PA)* that acts on behalf of the user to gather his/her preferences, as well as to build and update user profile. The *profiling agent* can explicitly ask the user about his/her preferences, or ask the user to rate a number of items that he/she knew before.

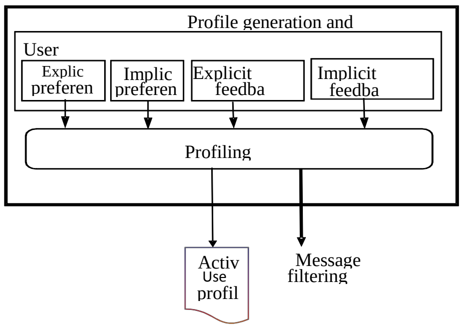


Fig. 4. Profile generation and maintenance component.

The *profiling agent* gathers this information and uses it in addition to observing the user behavior through, for example, following the URL to capture his/her interests. This agent will use proven AI techniques, such as learning technique to learn user preferences. It creates an active user profile if the user is new (the *profiling agent* can determine if the user is using the system for the first time during a log-in process) via graphical user interface, such interface can be like one shown in Fig. 5.

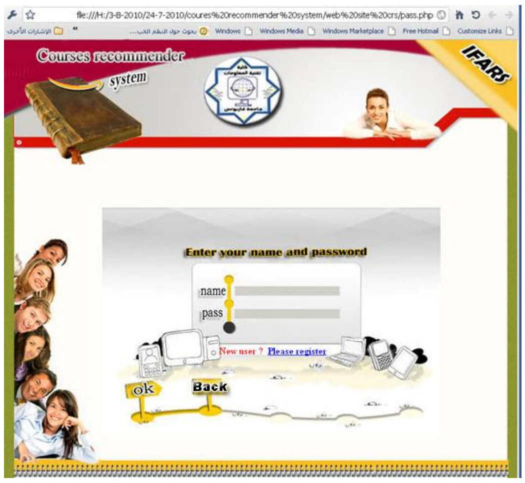

Courses recommender system
IFARS
Enter your name and password
name
pass
New user ? Please register
ok
Back

Fig. 5. Determine if the user is new or a registered user.

If the user is a registered user, the profiling agent use the gathered information to add new preferences to the existing user profile and delete preferences that had changed; in this way the agent adapts to changes in user preferences over the time. The profiling agent transfers active user preferences to the filtering agent to inform it that there is a new recommendation request. After the user receives the recommendation, he/she can give feedback to the system. This feedback can be explicit, such as rating the items (e.g. a system can provide the user with list of items and request his/her opinion by rating each item on the scale from 1 to 5), or implicit, such as purchasing the product or placing it in a purchases basket. *Profiling agent* uses suitable implicit feedback methods to learn user preferences and update user profile.

The active user profile can be based on content analysis approach as described by [28], whereby the profile contains information about the content of items of interest. By applying this approach, the architecture can use CF and CBF as a hybrid approach; where it gives the system ability to compare user profiles to recommend items that rated highly by users with similar profiles if CF approach applied; or to recommend items that similar to those stored in the user profile if CBF approach applied.

b) User needs determination component

This component (Fig. 6) takes responsibility of gathering user's current needs. It comprises of a *need determination agent (NDA)*, which has capability to interact with the user and gather his/her, requirements. The *NDA* can do that via queries (the developer chooses an appropriate way to show the queries, either as lists or questions). After the user requirements are gathered, the *NDA* transfers them to the *filtering agent*.

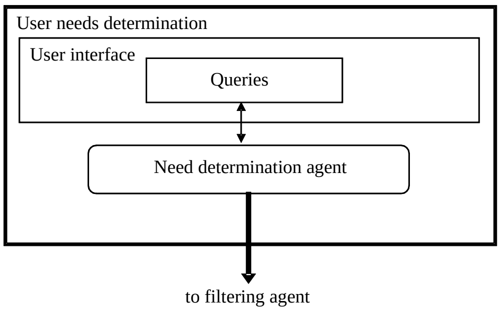


Fig. 6. Needs determination component

A. Recommendation Generator Component

The responsibility of this component is to generate the recommendations and translate them to the user interface. Fig. 7 illustrates Recommendation Generator component. Within this component, there is a filtering agent (FA), responsible for producing recommendations by applying a switching hybridization method. According to its up-to-date knowledge about the active user, items, other users in the system, and the context (i.e. it considers whether the user is a new or a registered user in the system, if received message came from *NDA* or *PA*, etc.), it switches between the CF, CBF and KB approaches.

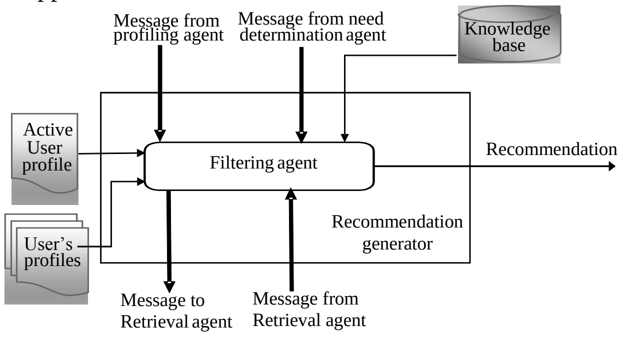

Message from profiling agent
Message from need determination agent
Knowledge base
Active User profile
Filtering agent
Recommendation
User's profiles
Recommendation generator
Message to Retrieval agent
Message from Retrieval agent

Fig. 7. Recommendation generator component

The filtering agent needs knowledge about when each approach is best suited to provide recommendation in order make a decision about which approach is best suited to produce recommendation. The filtering agent incorporates situation–action rules to switch between recommendation approaches. By following these rules, the presented architecture aims to avoid the disadvantages found in some existing recommender systems. As illustrated in Section 2.1.2, the systems based on individual algorithm have number of disadvantages, and the presented work aims to avoid some of them. Table I illustrates the three algorithms used in this work together with their disadvantages. This table shows only the disadvantages that this study solved.

TABLE I: RECOMMENDATION ALGORITHMS AND THEIR DISADVANTAGES

Disadvantages	CBF	CF	KB
New system	Yes	Yes	No
New user	Yes	Yes	No
New item	No	Yes	No
Unusual user	No	Yes	No
Sparsity problem	No	Yes	No
Over specialization	Yes	No	No

In this study, some situation–action rules are proposed. Filtering agent should follow these rules to switch between recommendation approaches. Furthermore, the agent can learn, so that it can add other situation–actions, because "An agent's behavior can be based on both its own experience and the built-in knowledge used in constructing the agent for the particular environment in which it operates." [29, p. 35]. The proposed rules are based on the advantages and disadvantages of each recommendation approach, known weakness points in these approaches, and when one recommendation approach can compensate for the shortcomings of another approach. The rules that the filtering agent will follow to decide which approach will be selected to produce recommendation are illustrated in Fig. 8.

```
a. If received message from NDA or PA Then
      send message to retrieval agent (RA) to request available items.
b. If the message came from NDA Then apply KB approach
      Else /* if the message came from PA */
         If new user Then apply KB approach
            Else Compare available items list with active user profile and other users' profiles
   If there is item found in available items list and not found in users' profiles Then apply CBF approach.
      Else Apply CF approach.
c. If CF approach fails to produce recommendations Then switch to apply CBF approach.
d. Match active user profile with recommendation list, if there is an item found in both Then remove it from recommendation list.
e. Match the recommendation list with available items, If there is an item found in the recommendation list and not found in available items list Then remove it from the recommendation list.
```

Fig. 8. Situation–action rules that applied by filtering agent.

When the filtering agent is activated, it sends message to the retrieval agent to request the available items, and then checks the message source; if the source is the need determination agent, this implies that the user wants recommendations not to be based on his/her preferences. In this case, the filtering agent applies the KB approach to produce recommendations, where this agent generates query to the knowledge base based on the user requirement that it received from the need determination agent or the profiling agent. When the query is completed, the filtering agent selects items that satisfy user needs to recommend them, and then it matches the recommendations list with the items that it received from the retrieval agent to remove unavailable items from the recommendation list. It subsequently transfers the recommendation list to the user interface. If the message source was the profiling agent, the filtering agent checks if the user is a new user or a registered user. For a new user, the FA will apply the KB approach; in this way the system avoids the new user problem (see [26] for more details about recommendation algorithms' problems), and if the system is newly developed, it helps to avoid new system problem. If the user is a registered user, the FA compares his/her active user profile and other users' profiles with the available items. If it found item in the available items list and the item is not found in any user profiles, the FA will apply the CBF approach, thus the system avoids new item problem. When the filtering agent applies the CBF, it does matching between the active user profile and the available items to produce the recommendations, where it recommends items that are similar to the items that the user preferred in the past. Then it transfers the recommendation list to the user interface. If the filtering agent does not find the new item, it applies the CF approach. By applying the CF approach, the system generates cross-genre recommendation (i.e. it recommends items not similar to those the user has already seen before). In this way the system will avoid over specialization problem. When the filtering agent applies the CF approach, it does matching between the active user profile and the other users' profiles to produce the recommendations based on the similarity between the active user and the other users in the system database. After generating the recommendation, the filtering agent matches the recommendations list with the active user profile to remove the items that the user has known before from the recommendation list. Then it matches the recommendations list with the items that it received from the retrieval agent to remove unavailable items from the recommendation list and transfers the recommendation list to the user interface. When the filtering agent fails to produce the recommendation by applying the CF approach, because the number of other users' profiles insufficient to apply the CF approach, or there is no user with a profile similar to that of the active user, it switches to apply the CBF approach, hence the system avoids new system problem and gray sheep problem.

The knowledge base contains knowledge on how well a specific item will satisfy the user's needs. The developer should have a good understanding about the recommendation domain and should use appropriate methods for its development and updates, such as production rules or frames.

B. *Items Collector Component*

This component (Fig. 9) is responsible for collecting the items _ and their features _ related to recommendation domain, from the different resources. Within this component, there is a Retrieval agent that takes the responsibility for searches in the resources to retrieve the items. These resources may be websites or internal database or both, according to the nature of the recommender system.

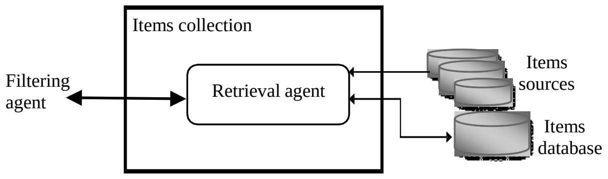


Fig. 9. Items collector component.

The retrieval agent subsequently stores the available items and their features in the items database. This ensures that the system is updated with new items in the recommendation domain. The retrieval agent retrieves the items continuously at offline stage (i.e. not during generating the recommendations); in this way, the recommendation computation time will be reduced. When the retrieval agent receives the message from the filtering agent, it extracts the items from the items database and sends list of available items to the filtering agent. The retrieval agent should be mobile agent. There are several benefits to using a mobile agent, such as reduction in the communication bandwidth, load balancing and overcoming latency of the network [30].

V. Case Study

We present a case study to demonstrate the practical implementation of the proposed work. A recommender system for course selection is built to demonstrate how the proposed recommender system will be adapted to the structure of the implemented architecture. The course recommender system helps students to decide which courses to take. The system can switch between three recommendation approaches (CF, CBF and KB) for recommending courses to students. The recommended courses can be similar to other students' preferred courses, or similar to the courses that the student has enrolled before. Furthermore, the suggested courses can be a result of the extraction of the system knowledge domain, which recommends courses that the student may have interest in. A number of characteristics for the course recommender system in this case study are summarized below:

- User profile: each student registered in the system has a profile that contains his/her preferences and all the information related to the courses to which the student has enrolled in the past.
- Course resources: this case study assumes that the resource is an internal database that contains the courses that the university is offering. Additional information about these courses; such as their description and availability is also included.
- Domain knowledge base: It contains knowledge about how the specific course meets the student needs and it should be represented by rules or ontology.

The Jason platform is used to implement the simple course recommender system. The reasons for selecting Jason are summarized below [31], [1]:

- It enables agents to communicate and coordinate with each other in a high-level way.
- It uses Saci, which is an infrastructure that provides (KQML-based) communication, as well as the ability to run MAS distributed over a network.
- It has extensibility ability by means of user defined "internal actions."
- It enables running test scenarios (i.e. can simulate a real environment) before deploying the system.

The proposed agents and their tasks for course recommender system are shown in table II.

TABLE II: AGENTS AND THEIR TASKS

Agents	Purpose
Profiling agent	• Gathering the student preferences • Gathering the relevance feedback • Building and updating the active student profile
Need determination agent	• Gathering the student's current needs
Filtering agent	• Produce the recommendations • Remove the courses that are not offered in the current semester from the recommendation list • Remove the courses that the student has studied before from the recommendation list • Translate the recommendation to the GUI
Retrieval agent	• Retrieves the courses that are offered in the current semester from the university courses database. • Stores the available courses in the recommender system database

This part of the case study provides the implementation of some of the agents' functionalities. Where, it is assumed that there is a student already registered in the system and that he/she requires course recommendations based on his/her preferences.

A. Some of Implementation Details

When running the MAS system, the courses_RecommenderSystem.mas2j file will be executed first to create the environment to execute the MAS. This file is a multi-agent system definition file that defines the agents in the system, as well as various parameters for the multi-agent system execution. In this file, four agents are defined, which comprise the system: login agent, profilingAgent, filteringAgent, and retrievalAgent, in addition to the identification of the communication infrastructure. The source code of the courses_recommenderSystem.mas2j is illustrated in Fig. 10.

```
MAS courses_RecommenderSystem {
// The Multi Agent System in  Case Study Courses Recommender
    System
  infrastructure:
     Centralised
  agents:
     login         beliefBaseClass jason.bb.TextPersistentBB;
    filteringAgent  ;
     profilingAgent ;
     retrievalAgent                              beliefBaseClass
    jason.bb.JDBCPersistentBB(
// parameters to connection to database
          "org.hsqldb.jdbcDriver", //driver for HSQLDB
          "jdbc:hsqldb:coursestore", // URL connection for the DB
           "sa", // user
           "", // password
"[course(7),course_lecturer(2),lecturer(2),department(2),topic(2),
    course_topic(2), course_place(2)]"); // List of the beliefs that
    are mapped into tables of the database
}
```

Fig. 10. Source code for courses_recommenderSystem.mas2j

The Jason used to implement the proposed course recommender system is a plug in inside the jEdit platform, which is a cross platform programmer's text editor written in Java (for more information about jEdit see [1]).

To simplify the implementation of the proposed MAS, a login agent has been proposed. This agent checks the registered students database to check whether the student using the system is registered in the database or not, and if the student profile exists, profiling agent for this student will be created. The proposed Login agent uses a customized belief base provided by Jason to create a file called "login.bb" to store its beliefs.

Some of the source code of the login.asl is illustrated in Fig. 11.

```
// Agent Login in Case Study Courses Recommender System
/* beliefs */
// initial student data
//  student(UserName, Password, Name, Address, City, EMail)

student(          zinab,
                  "zinab",
                  "zinab elbadree",
                  "libya, 00218",
                  "benghazi",
                  "zsaad@yahoo.com").

student(          rabeia,
                  "elfrjany",
                  "Rabeia N. Elfrjany",
                  "libya, 00218",
                  "benghazi",
                  "rr1_456@yahoo.com").
student(          guest,
                  "",
                  "Guest",
                  "",
                  "",
                  "").
```

```
/* plans */
// student managing
+!kqml_received(S, askOne, add_student(UserName, Password,
 Name, Address, City, EMail), M)
:not student(UserName, _, _, _, _, _)
<-+student(UserName, Password, Name, Address, City, EMail);
.send(S, tell, ok, M);
.print("Add student: ", Name).
+!kqml_received(S, askOne, add_student(UserName, _, _, _, _, _),
 M)
:       student(UserName, _, _, _, _, _)
<-     .send(S, tell, error("Invalid username"), M);
.print("Invalid username").
+!kqml_received(S, askOne, student(UserName),M)
:      student(UserName, Password, Name, Address,City, EMail)
<-   .send(S, tell, student(Name, Address, City, EMail), M).
+!kqml_received(S, askOne, user_logon(UserName, Password), M)
:     student(UserName, Password, Name, _, _, _)
<-   .print("logon ok for ",UserName);
       // creates an agent for this user
.create_agent(UserName,"profilingAgent.asl",
 [beliefBaseClass("jason.bb.TextPersistentBB")]);
       .send(S, tell, student(Name), M).
+!kqml_received(S, askOne, user_logon(UserName, Password), M)
<-      .send(S, tell, error("Username or password invalid!"), M).
+!kqml_received(S, askOne, user_logout(UserName), M)
  <- .print(ok).
<-   .kill_agent(UserName);
     .print("Recommendation Request for ",UserName," was
 Achieved").
+!kqml_received(S, askAll, student_ids, M)
<-   .findall(Id,student(Id, _, _, _, _, _),ListStudent);
       .send(S, tell, ListStudent, M);
.print("All students ", ListStudent).
```

Fig. 11. Source code of the ogin.asl

VI. Conclusion

We presented a general architecture that can serve as a guide for new developers in the field of recommender systems, enabling them to build recommender systems according to their requirements. The main architecture goal is to enable building specific recommender system that can satisfy the following:

- Recommend items based on user preferences.
- Gather user preferences with least involvement from the user.
- Recommend items within the appropriate time limit for enabling the user to act on them immediately.

The developers using the architecture designed in this study to build their bespoke recommender system will benefit from the following:

- The architecture inherits the advantages of the CF, CBF and KB approaches.
- The architecture helps to avoid new system problem via the KB approach.
- The architecture enables the recommender system to recommend new items to the user (solves the new item problem) by using the CBF approach.
- The architecture enables the recommender system to produce cross-genre recommendation (i.e. it solves over-specialization problem) by applying the CF approach.
- By using the content approach to build a user profile, the architecture helps to solve sparisty problem.
- The architecture gives the recommender system ability to recommend useful items to a user with unusual preferences (thus solving the gray sheep problem) via the CBF approach.
- Item retrieval from the different sources at offline stage reduces the recommendation computation time.

There are further advantages in using multi-agent technology to design the presented architecture, which are summarized below:

- The agent's reactive ability gives a RS enough flexibility to adapt to the user's changing interests.
- The proactive information gathering ability creates an up-to date RS whit new items in the recommendation domain.
- Recommender system will be scalable, in view of the fact that they are inherently modular, using MAS technology. Thus, new agents can be easily added to the system when required.
- By providing an agent with a high-level goal to produce the recommendations, it will act autonomously to achieve its goal.

References

[1] R. Bordini, J. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in Agent Speak using Jason*. England: Wiley-Interscience, 2007.

[2] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intelligence*, England: MIT Press, 1999.

[3] S. Franklin and A. Graesser, "Is it an agent, or just a program?: A taxonomy for autonomous agents," in *Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, London, 1996, pp. 21 - 35.

[4] D. Garlan and M. Shaw, "An introduction to software architecture," *Advances in Software Engineering and Knowledge Engineering*, World Scientific Pub Co., Singapore, 1993, vol. II pp. 1–39.

[5] B. Chaptini, "Use of discrete choice models with recommender systems," PhD. Thesis, Dept. of. Civil and Environmental Engineering, Massachusetts Institute of Technology, 2005.

[6] P. Cunningham, R. Bergmann, S. Schmitt, R. Traphner, S. Breen, and B. Smyth, "WEBSELL: Intelligent sales assistants for the World Wide Web," in *Workshop at the Fourth International Conference on Case- Based Reasoning*, 2001.

[7] T. Tran and R. Cohen, "Hybrid Recommender Systems for Electronic Commerce," in *the 17th National Conference on Artificial Intelligence AAAI*, 2000.

[8] J. Itmazi, "Flexible learning management system to support learning in the traditional and open universities," PhD Thesis, University of Granada, 2005.

[9] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval Journal*, vol. 4, no. 2, pp. 133-151, July 2001.

[10] P. Resnick, N. Iacovou, M. Suchak P. Bergstrom, and J. Riedl, "GroupLens: An open architecture for collaborative filtering of netnews," in *Proc. of the ACM Conference on Computer-Supported Cooperative Work*, Chapel Hill, NC, 1994, pp. 175–186.

[11] M. Montaner, "Collaborative recommender agents based on case-based reasoning and trust," PhD Thesis, Department of Electronics, Computer Science and Automatic Control, University of Girona, Girona, 2003.

[12] M. Pazzani, J. Muramatsu, and D. Billsus, "Syskill and Webert: Identifying interesting web sites," in *the Thirteenth National Conference on Artificial Intelligence*, Portland, 1996, vol. 1, pp. 54-61.

[13] D. Elliott, J. Rutherford, and J. Erickson, "A recommender system for the DSpace open repository platform," Tech. Rep., HP Laboratories, Bristol, 2008.

[14] R. Burke, "Knowledge-based recommender systems," *Encyclopedia of Library and Information Systems*, vol. 69, 2000.

[15] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, 2002, pp. 331-370

[16] G. Adomavicius and A. Tuzhilin, "Towards the next generation of recommender systems: A survey of the state-of-the-Art and possible extensions," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 734-749, 2005.

[17] J. Bradshaw, "An Introduction to Software Agents," in *Software agents*, AAAI Press, 1997, pp. 3-46.

[18] M. Raphael, "Knowledge base support for design and synthesis of multi-agent systems," MSC Thesis, School of Engineering and Management, Air Force Institute of Technology, 2000.

[19] R. O'Connor and E. Gaffney, "A Distributed framework for implementing a multi-agent assistant system," Tech. Rep. CA1198, Dublin City University, 1998.

[20] O. Niinivaara, "Agent-based recommender systems," Tech. Rep., Dept. of Computer Science, University of Helsinki, 2004.

[21] F. Gandon, "Distributed artificial intelligence and knowledge management: Ontologies and multi-agent systems for a corporate semantic web," Ph.D. dissertation, School of Sciences and Technologies, INRIA and University of Nice, Sophia Antipolis, 2002.

[22] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, 1997, pp. 345-383.

[23] L. Guo, D. Roberston, and Y. Chen-Burger, "Business process model based multi-agent system development," in *the Second International Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments*, Beijing, China, 2004 .

[24] A. Albashir, "Review of multi-agent systems and their tools," University of Liverpool, 2002. [Online]. Available: http://www.csc.liv.ac.uk/~ali/wp/MAS_review.pdf.

[25] R. Sotomayor, J. Carthy, and J. Dunnion, "The design and implementation of an intelligent online recommender system," in *Mixed-Initiative Problem-Solving Assistants, Fall Symposium Series. AAAI*, 2005.

[26] V. Marivate, G. Ssali, and T. Marwala, "An intelligent multi-agent recommender system for human capacity building," in *Proceedings of the 14th IEEE Mediterranean Electrotechnical Conference*, 2008, pp. 909 – 915.

[27] R. Mukherjee, N. Sajja, and S. Sen, "A Movie recommendation system - an application of voting theory in user modeling," in *User Modeling and User-Adapted Interaction*, vol. 13, pp. 5-33, 2003.

[28] M. Balabanovic and Y. Shoham, "Fab: Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, pp. 66 – 72, 1997.

[29] S. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*, New Jersey: Prentice-Hall, 1995.

[30] R. Nienaber, "A model for enhancing software project management using software agent technology," Ph.D. dissertation, University Of South Africa, 2008.

[31] R. Bordini and J. H ̈ubner, "BDI Agent Programming in AgentSpeak Using Jason," in *Computational Logic In Multi-Agent Systems*, London, UK, LNCS vol. 3900, 2006, pp. 143-164.

Mohamed Elammari received a B.Sc. and M.Sc. degrees in computer science from Acadia University in Nova Scotia, Canada and a Ph.D. degree in computer science from Carleton University in Ottawa, Canada. He is currently associate professor in the Department of Software Engineering at the University of Benghazi. His research interests include Software Engineering, Agent Systems, and e-government.

Rabeia Elfrjany received a B.Sc. and M.Sc. degrees in computer science from University of Benghazi in Benghazi, Libya. She is currently working towards Ph.D. degree in software engineering. Her general research interests include Software Engineering and Multi-agent systems.