Recommender agents

Prodan Andrei-Cristian
Master, IInd year, group 256
`prodan.cristian@gmail.com`

January 18, 2010

Abstract

This documents presents a multiagent system which represents a recommender system for a user. The idea is that the user connects to (has) an AssistentAgent. This agent will then collaborate with other types of agents (RecommenderAgent, InformationAgent) and an environment in order to provide the users with different recommendations, in a distributed manner. The software is implemented in the Ruby programming language, using only it's standard API. For this software we make expensive use of the DRb (Distributed Ruby) module.

1 Project description

This software implements the multiagent paradigm in a distributed manner. The software (which we will refer to by the name of ReCo) serves as a distributed environment for recommending movies to a user. There are mainly two functions of this system from the user perspective that is:

- get some recommendations for me (the user);
- set rating x for movie y on behalf of me (the user);

Inside the software this functions have the following signature.

```
get_recommendations(user_id)
```

```
set_recommendation(user_id, movie_id, rating)
```

The actual method for the recommendation is collaborative filtering. This technique makes use of triples like $< movie_{id}, rating_{id}, user_{id} >$ in order to compute recommendations for users. It relies of the fact that if two users have agreed upon certain preferences in the past, it is very likely to have the same preferences in the future. There are several methods that can be used to compute similarities between the users: neighborhood methods (Pearson coefficient) or latent factors decomposition.

Because this software focuses the design of agents none of the actual algorithms for the recommendation have been implemented. We only state some of the potential methods for doing this. Right now, we have placed some stubs methods which wait for a random number of milliseconds and return a random result. The usefulness of this software could be justified by this.

Figure 1 shows the architecture of the system. We analyze each component in the next step.

- **AssistantAgent** (AssistantAgent.rb). This agent communicates with the users. It exposes two interfaces for getting and setting recommendations (details on figure 2).
- **The environment** (environment.rb). Is a service which shares a MessageQueue object with different agents. It is the layer that makes the communication possible between the AssistantAgent and the InformationAgent. Each time he has to respond to a query (which he takes from the queue, in form of a message), he "sends" a RecommenderAgent to ask the InformationAgent some info based on the query. The RecommenderAgent does this using Remote Method Invocation. By chaining all these calling methods, the environment gets back the answer and puts it into the queue again but in a form of an answer (they will be then used by the AssistantAgent as an answer to their initial query).
- **The InformationAgent** (InformationAgent.rb). This agent runs like a standalone service. He waits for it's methods to be called using RMI by the RecommenderAgent. It's methods can be seen in figure 2.
- **The RecommenderAgent** (RecommenderAgent.rb). This agent is instantiated by the environment and "sent" to ask the InformationAgent to solve it's query. It has only one ask method but other things might be implemented here.
- **The message queue**. The message queue holds messages which can be of type "question" and "answer" and represent communication messages between the AssistantAgent and the environment. More on this can be read about in the next chapter. The mechanism for communication between agents is **asynchronous message passing**.

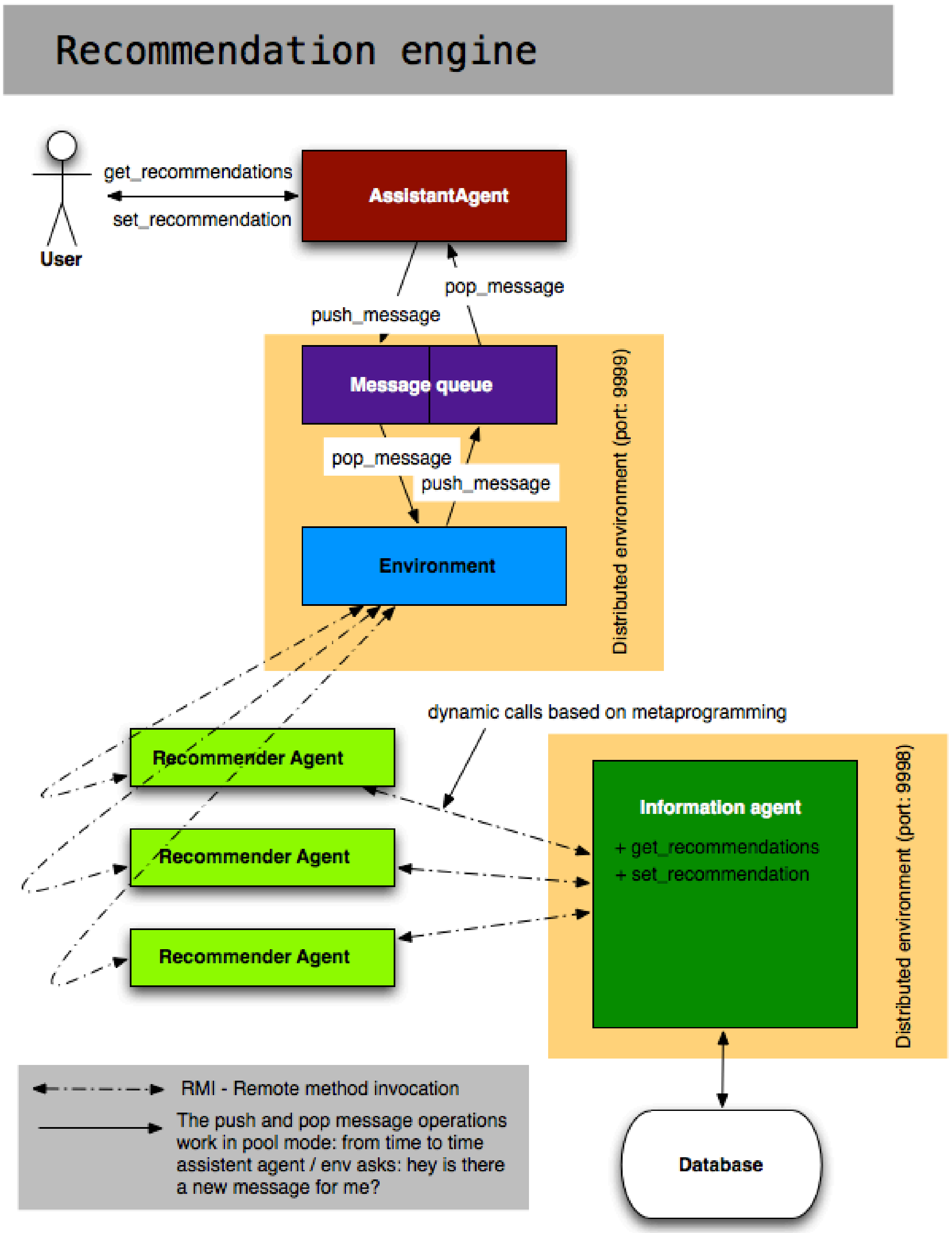

Recommendation engine
User
get_recommendations
set_recommendation
AssistantAgent
pop_message
push_message
Message queue
pop_message
push_message
Environment
Distributed environment (port: 9999)
dynamic calls based on metaprogramming
Recommender Agent
Recommender Agent
Recommender Agent
Information agent
+ get_recommendations
+ set_recommendation
Distributed environment (port: 9998)
Database
RMI - Remote method invocation
The push and pop message operations work in pool mode: from time to time assistent agent / env asks: hey is there a new message for me?

Figure 1: ReCo architecture

2 Implementation

This piece of software has been implemented using DRb module which can be found in the standard API of the Ruby distribution.

DRb allows Ruby programs to communicate with each other on the same machine or over a network. DRb uses remote method invocation (RMI) to pass commands and data between processes.

The ENVIRONMENT starts as a service at port 9999 and exposes a queue. This queue is shared with different ASSISTENTAGENTS that need to use the recommendation engine. From time to time, the environment looks through the queue (pool) to see if there are any new messages. If there are, it takes them and sends them to the RECOMMENDERAGENT. This Agent is instantiated in the ENVIRONMENT and calls the ASKINFORMATION method on the InformationAgent. The param to it's query is a MESSAGE. Based on that MESSAGE, the InformationAgent, which runs at port 9998 can figure which kind of info the RECOMMENDERAGENT needs (weather it is GET_RECOMMENDATION or SET_RECOMMENDATION).

In this time the ASSISTANTAGENT waits for itś response. It does this by checking from time to time the shared queue. He basically asks: "hey is there any new message for me".

A message contains the info shown in figure 2. The KIND field can be "Q" or "A" weather the message refers from a query of an agent or a response of the environment respectively. The TYPE field may be: GET_RECOMMENDATIONS or SET_RECOMMENDATION and this is used by the INFORMATIONAGENT to figure out what type of query should he respond. The params field contains a Hash with params like: USER_ID, MOVIE_ID, RATING or AGENT_ID.

The classes which form the system are shown in figure 2.

3 Demonstration

In order to be able to test the system Ruby 1.8 has to be installed on the system. The standard installation for Ruby contains the DRb system.

The simluation.rb file contains a simulation for this system. The file is quite intuitive and can easily be modified. It spawns 10 threads and each thread initializes one agents which make two recommendation requests.

From the command line, go to the root directory of the software. First make sure all the necessary files have execution rights (not necessary on

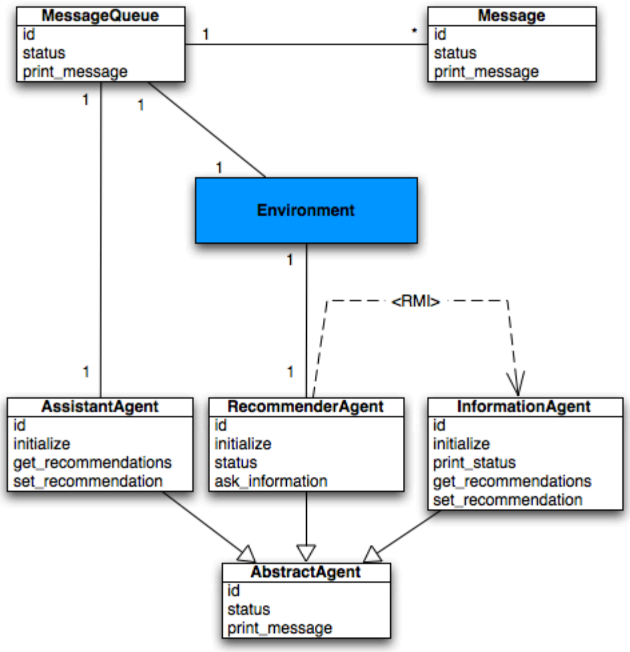


Figure 2: ReCo classes

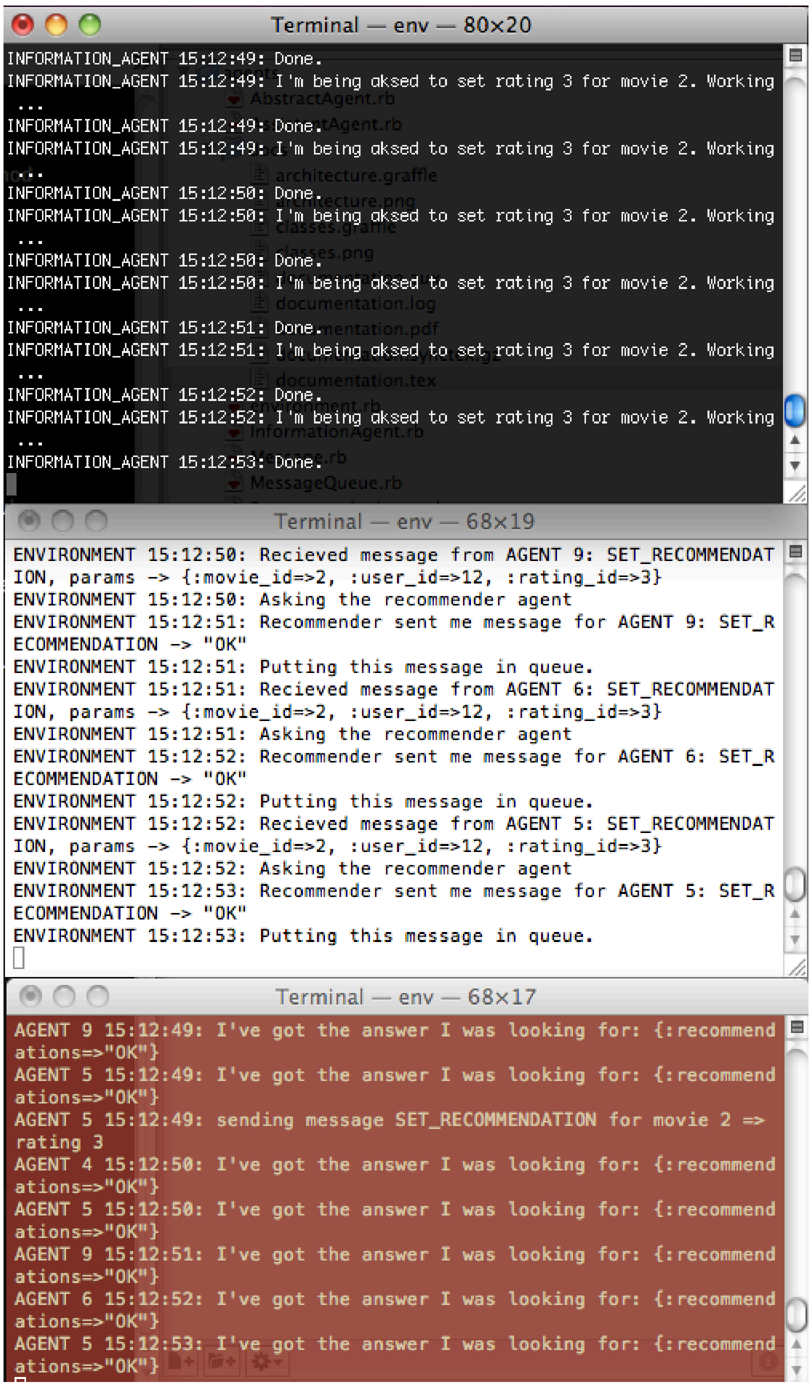
windows). This can be done using the following commands (Linux, MacOS):

```
chmod +x simulation.rb
chmod +x InformationAgent.rb
chmod +x environment.rb
```

Then run:

```
./InformationAgent.rb
./environment.rb
./simulation.rb
```

On a MacOs Machine this should show something like the screens in figure 3.



```
INFORMATION_AGENT 15:12:49: Done.
INFORMATION_AGENT 15:12:49: I'm being aksed to set rating 3 for movie 2. Working
 ...
INFORMATION_AGENT 15:12:49: Done.
INFORMATION_AGENT 15:12:49: I'm being aksed to set rating 3 for movie 2. Working
 ...
INFORMATION_AGENT 15:12:50: Done.
INFORMATION_AGENT 15:12:50: I'm being aksed to set rating 3 for movie 2. Working
 ...
INFORMATION_AGENT 15:12:50: Done.
INFORMATION_AGENT 15:12:50: I'm being aksed to set rating 3 for movie 2. Working
 ...
INFORMATION_AGENT 15:12:51: Done.
INFORMATION_AGENT 15:12:51: I'm being aksed to set rating 3 for movie 2. Working
 ...
INFORMATION_AGENT 15:12:52: Done.
INFORMATION_AGENT 15:12:52: I'm being aksed to set rating 3 for movie 2. Working
 ...
INFORMATION_AGENT 15:12:53: Done.
```

```
ENVIRONMENT 15:12:50: Recieved message from AGENT 9: SET_RECOMMENDAT
ION, params -> {:movie_id=>2, :user_id=>12, :rating_id=>3}
ENVIRONMENT 15:12:50: Asking the recommender agent
ENVIRONMENT 15:12:51: Recommender sent me message for AGENT 9: SET_R
ECOMMENDATION -> "OK"
ENVIRONMENT 15:12:51: Putting this message in queue.
ENVIRONMENT 15:12:51: Recieved message from AGENT 6: SET_RECOMMENDAT
ION, params -> {:movie_id=>2, :user_id=>12, :rating_id=>3}
ENVIRONMENT 15:12:51: Asking the recommender agent
ENVIRONMENT 15:12:52: Recommender sent me message for AGENT 6: SET_R
ECOMMENDATION -> "OK"
ENVIRONMENT 15:12:52: Putting this message in queue.
ENVIRONMENT 15:12:52: Recieved message from AGENT 5: SET_RECOMMENDAT
ION, params -> {:movie_id=>2, :user_id=>12, :rating_id=>3}
ENVIRONMENT 15:12:52: Asking the recommender agent
ENVIRONMENT 15:12:53: Recommender sent me message for AGENT 5: SET_R
ECOMMENDATION -> "OK"
ENVIRONMENT 15:12:53: Putting this message in queue.
```

```
AGENT 9 15:12:49: I've got the answer I was looking for: {:recommend
ations=>"OK"}
AGENT 5 15:12:49: I've got the answer I was looking for: {:recommend
ations=>"OK"}
AGENT 5 15:12:49: sending message SET_RECOMMENDATION for movie 2 =>
rating 3
AGENT 4 15:12:50: I've got the answer I was looking for: {:recommend
ations=>"OK"}
AGENT 5 15:12:50: I've got the answer I was looking for: {:recommend
ations=>"OK"}
AGENT 9 15:12:51: I've got the answer I was looking for: {:recommend
ations=>"OK"}
AGENT 6 15:12:52: I've got the answer I was looking for: {:recommend
ations=>"OK"}
AGENT 5 15:12:53: I've got the answer I was looking for: {:recommend
ations=>"OK"}
```

Figure 3: ReCo demo