Actors-A Instructions

Learning objectives

Learn how to structure and build decentralized systems
Deepen knowledge of actors
Gain familiarity with rule-based programming

Agent-Based Referral Networks

We study referral networks as an important exemplar of a decentralized system.

The idea in this assignment is to mimic interactions between people as they locate services in the real world. Imagine that each person has some capabilities (e.g., paint a house) that could be of value to others. Each person has needs that others could fulfill. However, each person has only a limited knowledge of the others' capabilities and of the knowledge they have of others.

Thus each person tries to locate a desired service provider by querying others: such querying being initially limited to those directly known. Upon receiving a query, a person may ignore the query, provide the capability (e.g., offer to paint the house), or provide a referral to another person.

What distinguishes a referral network from a flooding approach is that the queries are not forwarded but a referral is provided to the querying person. The querying person controls whether and whom to query, and there could be a potentially unbounded series of queries and referrals: we term this series a referral chain. For simplicity, let us assume that the querying person terminates the process either when it realizes that no matching service is being located or as soon as the first provider of the desired service is located.

Upon receiving the service, each person evaluates it. The evaluation may lead the person to update a private opinion (there being no central reputation repository) of the service provider and of all those who produced a referral that led to that service provider. If the service experience is great, everyone in the referral chain gains in the private opinion; and conversely if the service experience is atrocious. Each person determines which of the others it knows are generally helpful and valuable and which are not: this process can lead to each person

The goals of this assignment are

- Build a pretend referrals network. You specify an actor class, which is instantiated for each user. Your actors send and receive messages and maintain the knowledge as explained above.
- Build an Android app that communicates with one designated instance of the above-mentioned actor class.

Computational Representation

The above approach, though it is good and has lasted for millennia, places severe demands on people to produce referrals. We are interested in offering computational support for referrals. Our ulterior motive is to develop fully automated, decentralized service location approaches but we won't go into that for now.

Each person is assisted by a user interface agent and a referrals agent. The user interface agent is implemented as a simple Android app that communicates with the same user's referrals agent. (The app communicates with only one referrals agent.) The referrals agent is implemented as an actor via rule-based reasoning. The service in question is treated as an information service: that is, not "paint the house" but "how much paint should I buy for my kitchen." Thus the service itself can be provided via a message. The bulk of the action is between the actors (referrals agents of the various users). Each actor can handle incoming queries and can produce two messages: referrals and answers.

Each actor maintains knowledge of an arbitrary number of acquaintances, of whom a fixed number are considered neighbors. In graph terms, there is an out edge for each neighbor, because neighborhood is not symmetric in this setting. An actor's initial queries go to a subset of its neighbors. Its set of neighbors can possibly change after each time it seeks and receive or doesn't receive an answer.

The knowledge that each actor has about itself is limited to

- Its needs based on which its queries it will generate. For simplicity, we model the needs as a fixed width vector of reals between 0 and 1, e.g., [0.7, 0.113, 0.9, 0.01]. Each dimension corresponds to some need type. For example, we can imagine that there are four information needs: cooking, house repairs, education, and entertainment. The vector [0.7, 0.113, 0.9, 0.01] corresponds to a student nearing graduation, whose needs are highest for education and food, and pretty low for house repairs and entertainment.
- Its capabilities for providing answers to queries from others. For simplicity, we model this knowledge as a fixed width vector of reals between 0 and 1, e.g., [0.6, 0.93, 0.15, 0.45]. For example, we can imagine that there are four information capabilities: cooking, house repairs, education, and entertainment. The vector [0.6, 0.93, 0.15, 0.45] corresponds to a landlord who provides cooked food of middlish quality, takes care of all house repairs, supports education by providing good WiFi and nothing else, and offers not stellar entertainment.

The knowledge that each actor has about another actor is limited to

- Its expertise, i.e., ability to provide answers on the different need types, easily represented as a vector such as the above.
- Its sociability, i.e., ability to provide referrals on the different need types, easily represented as a vector such as the above.

For simplicity, an actor evaluates the quality of an answer based on the inner product (multiply corresponding elements, then sum) of the answer vector and the query vector.

The messages each actor produces are determined based on the following approach. (We fix the approach to simplify evaluation of the assignments.)

- The specific queries to be sent to others are generated based on a random process by adding a small noise to each element of the need vector. The TA will provide the query generation function.
- The neighbors to be selected to receive a query depend upon a weighted sum of (1) the inner product of the query and the neighbor's sociability and (2) the inner product of the query and the neighbor's expertise. If the values for different neighbors are sufficiently different (more than a preset threshold), contact only those that have high values, else contact all neighbors.
- If an incoming query matches one's own expertise, then generate an answer from the expertise and send it to the querying actor, else if the query matches the sociability or expertise of any neighbors (same weighted sum as above), provide a referral to those neighbors.

After each service episode, each actor updates its models of the other actors. The TA will provide the update function. As a result of the update, select the top so many (parameter to be supplied) as neighbors for use in the next episode.

Deliverables and Evaluation

You will use the Drools Rule Engine to capture the decision making of each actor. To make the problem tractable, we have simplified it a lot. As a result, it would not require many of the facilities afforded by Drools. However, it is important to get your feet wet with Drools [excuse the mixed metaphor :-)], for which this assignment will provide you an opportunity.

For this assignment, we will provide the following:

- An input graph, whose vertices are the actors (each a referrals agent of a pretend user) and whose directed edges correspond to each agent's neighbors. Each actor would have a print name such as Wolf Ridge Apartments.
- The initial values for the various vectors.
- The necessary query and update functions.
- The number of neighbors allowed.
- How the messages are to be logged.
- How an English query is mapped to a vector.

The first part of the assignment is for you to execute a specified number of queries. For example, if there are 20 actors, you would generate 25 rounds in which each actor produces and resolves a query. (In this example, there would be 500 queries executed through the system.) The message traces and resulting network should match what the TA.

The second part of the assignment is to generate a query from the Android app and display an answer. For example, the user may ask for "good food, no repair hassles, focus on education" and the Android app may send in a vector [0.7, 0.113, 0.9, 0.01] to the actor (user's referral agent), who will run with it and produce an answer such as Wolf Ridge Apartments says "OK food, no repair hassles, near library, movies in the Oval." The TA will figure out details of this part of the assignment.

Getting Started

- You already have familiarity with the Play Framework. Remind yourself of it.
- You already have familiarity with Akka. Remind yourself of how to program actors in Akka.
- Download Drools from http://drools.org/
- Review the documentation link on http://drools.org/
    - Please note that the official documentation is quite good (and better than the alternatives I found). However, it is organized in a strange way. Read Chapter 5 for useful background. However, concentrate on Chapter 6 to learn about Drools rule syntax and other aspects.