A Multi-Agent Implementation of Social Networks

Enrico Franchi
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma
Parma, Italy
efranchi@ce.unipr.it

Abstract—**In this paper, we present a multi-agent system implementing a fully distributed Social Network System supporting user profiles as FOAF profiles. This system is built around the idea that users should be the sole owners of the information they provide (either consciously or unconsciously) and addresses privacy issues by design, also minimizing the amount of information users have to disclose in order to make new friends. Users are represented by agents that both mediate access to private data and proactively negotiate with other agents in order to extend their user's social network. We also present the distributed connection discovery algorithm used by the agents and detail the representation of data in the users' profiles used to support the algorithm. The design is rather agnostic about the layer responsible for the communication technology; here we present a possible implementation on the top of the HDS software framework.**

Keywords; Middleware for distributed social systems, social networks, multi-agent systems

I. Introduction

In social sciences, a social network is a structure of individuals connected with some kind of relationship; the focus is placed almost entirely on relations and individuals themselves are often just represented as tiny dots in a graph [1]. However, from our point of view, these tiny dots are rich of important information; information at least as important as relations since it can be used to discover the relations themselves. Therefore, we define a Social Network (SN) as a connected graph of public and/or semi-public profiles. A profile represents a user in the network and connections between users are represented as labeled edges in the graph. A profile is public if all the information therein contained is accessible to all users in the system or to all users that have a connection with the profile owner. A profile is semi-public if there are restrictions on the pieces of information that are available to other users. Fully private profiles are of no interest: if no information were accessible at all, we would not even know that the user is registered.

A Social Network System (SNS) is a software system that supports the persistent storage of SNs and that provides means to update, to add and to query information. This definition is roughly equivalent to the one used in [2], which describes a SNS as a site allowing users to: i) construct a public or semi-public profile within a bounded system; ii) articulate a list of other users with whom they share a connection; iii) view and traverse their list of connections and those made by users within the system. We also expect a SNS to suggest proactively possible acquaintanceships among users, using the information in user profiles (or other user provided data) according to user specified policies. This is indeed the main service a SNS offers: it gathers information on users' social contacts, construct a large interconnected social network and reveal to users how they are connected to others in the network.

In traditional SNSs most profile entries are related to hobbies and interests, such as music and sports [3]. Other features commonly found in traditional SN profiles provide information about education and past jobs; some well-known SNSs (e.g., LinkedIn [4]) are entirely centered on the latter kind of data.

In order to discover the connections, traditional SNSs store every possible piece of information. The huge amount of recorded data can raise privacy concerns and, even though most SNSs have rather acceptable privacy policies (e.g., [5]), visibility rules on contents is more geared towards protecting content from other users than from the system itself (or its advertising partners). For example, in Orkut [6] it is possible to define which parts of the user profile are visible to: i) the user himself; ii) his friends; iii) friends of friends; iv) everyone. Another serious problem regards the ownership of inserted data. Some communities make it clear that the sole owner remains the original user [7], while in other systems the issue is foggy. On the other hand, in a completely distributed system, the user is the sole owner and, by design, has full control of his data.

In this paper, we present a multi-agent system implementing a fully distributed SNS. This system is built around the idea that users should be the sole owners of the information they provide (either consciously or unconsciously) and addresses privacy issues by design, also minimizing the amount of information users have to disclose in order to make new friends.

In Section II we briefly review the results in the field of social networks; in Section III the abstract design of our system is detailed, especially detailing: i) the connection discovery algorithm we devised in order to discover acquaintanceships between users, without resorting to any centralized omniscient entity; ii) the semantic structure of user profiles. In Section IV the HDS framework is introduced and in Section V we present how the system described in Section III can be built on top of HDS. Eventually, in Section VI we draw some conclusions and we propose some future research directions based on the current work.

II. Social Networks and Multi-Agent Systems

Social networks have been studied for at least 50 years; one of the earliest and more important results is Milgram's small world phenomenon [8] [9]; the small world phenomenon is a basic statement about the abundance of short paths in a graph whose nodes are people, with links joining pairs who know one another. The chains of acquaintance are also known as "six degrees of separation", since six is their average length. The results of Milgram's original experiment have been reproduced in recent years using emails [10].

Social network structure has been thoroughly studied [11] and mathematical models have been devised in order to explain the small world phenomenon and the "searchability" of social networks [12] [13] [14], i.e., the fact that people with mostly local information are able to route messages to people they don't know. Essentially, the social networks feature some individuals that have a number of contacts above average and these individuals work as "hubs" connecting different groups of people. Moreover, there are some "non local" links, which allow connections between otherwise distant groups.

While early web communities were centered around shared hobbies and interests, the recent advent of modern social network sites changed entirely the focus: people preferred to be linked with people they know in real life [2].

Among the early works on software agents supporting social networks the papers on Yenta [15] and on ReferralWeb [16] are particularly relevant. These early systems mined various public and private (such as email archives) resources to build social networks, but were intended primarily as an "expert-finding tool" and secondarily to form interest-based communities. Yenta also had the idea of "sending a message to a group", which can be regarded as a key feature of modern SNSs. However, the focus was still on common interests rather than real life acquaintance.

More recently a social network navigation and analysis tool called Flink [17] has been developed. Flink uses FOAF profiles and other public data to present and analyze the social network of the semantic web community. Polyphonet [18] is another social network mining system tailored to facilitate communication and mutual understanding for an academic community. Differently from Yenta [15] and ReferralWeb [17], these are centralized systems.

None of these systems is a SNS, even though each solved issues similar to ours, such as: i) the extraction of relations from profiles and ii) the idea of building social networks from sparse data.

III. System Design

This section briefly explains the challenges of designing a completely distributed social network system (DSNS) along with the adopted solutions. The design is rather agnostic about the layer responsible for the communication technology. We essentially assume that agents have a unique identifier and can receive and send synchronous and asynchronous messages. A proactive software agent represents every user and his tasks are: i) it mediates access to user's data; ii) it actively negotiates with agents in its social network to enrich with new connections the social network itself. The negotiation is performed using data stored in the user's profile, according to user specified rules.

In the following subsections, we detail the connection discovery algorithm used to negotiate new friendships and the representation of profiles containing user information.

A. The connection discovery algorithm

The main problem is that as soon as we break the system unity into multiple autonomous agents, things get more complicated: each agent accesses only its users data and there is no "omniscient" third party which draws the connections among users. A distributed algorithm to discover connection in a way that privacy is not violated is a mandatory requirement of such a system: even if the same amount of data is present in the system, it is up to the agents to discover the acquaintance information implied by the data in user profiles. We say that a link is *latent* if a pair of users would like to be connected provided they were aware of the other user's existence in the system. A connection is *active* if the user's respective agents are actually connected.

Here we present a distributed algorithm that aims at activating most latent connections. In order to simplify the presentation, we say that two connected agents are friends, even though the system supports fine-grained semantic connections, which express the true kind of relationship between users. In fact, every pair of users may be linked through multiple connections (and probably should).

We assume that some agents are already connected, for example because their user manually connected when they have been invited to the system. The problem is that the active connections are a minimal part of the latent connections, while our goal is that all latent connections become active. In order to reach our goal, the connection discovery algorithm let each agent provide some personal information to the agents it is already linked to and allows them to broker new links with their respective friends.

In Fig. 1, the connection discovery algorithm is presented, assuming that agent A is linked to agent B with a connection of type L and wants to make friends with agents connected to B through a connection of type L. A sends a FindConnection(L, ex) to B, where ex is a list of agents A does not want to connect with (for example, because it is already connected with) and where A is sure that each agent in ex is known to B. B chooses a subset of its friends connected with a link of type L disjoint with ex and sends each of them a RequestConnection(A, L) message, meaning that A may want their friendship. At this point, C decides whether a connection with A is desirable or not. If so, A answers B with an AcceptConnection(A, L) message and consequently B sends A an AcceptedConnection(C, L) message to A. Eventually, A finalizes the link with C with an AcceptedConnection(A, L) message. If C prefers not to connect with A, it issues a RefuseConnection(A, L) to B. In the latter case, A will not be aware of C existence. Moreover, B could record C answer and

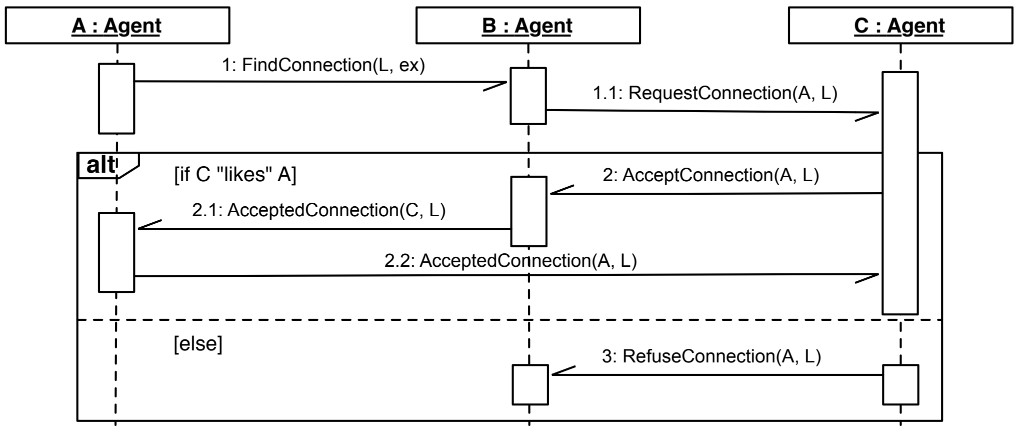
A : Agent
B : Agent
C : Agent
1: FindConnection(L, ex)
1.1: RequestConnection(A, L)
alt
[if C "likes" A]
2: AcceptConnection(A, L)
2.1: AcceptedConnection(C, L)
2.2: AcceptedConnection(A, L)
[else]
3: RefuseConnection(A, L)

Figure 1. The sequence diagram presenting the connection discovery algorithm.

consequently exclude C from successive connection requests from A.

The algorithm is designed to propagate the least possible amount of information needed to establish new connections. The FindConnection(L, ex) message in fact propagates no new information at all: A simply allows to share some information (on A) that B already has. A can create the ex list with the identifiers of agents suggested by B through a previous AcceptedConnection message (both in case A did accept the connection or it did not).

The RequestConnection(A, L) message *does* propagate information: before that message, C could not even know about A existence (not that A had something to do with L or it knew B). However, this cannot be avoided. Some information must be shared in order to create a connection. With our algorithm the ones who want to make new friends are the only ones that start sharing information and are perfectly in control of what kind and amount of information they are willing to share. Of course, the more information they share, the more likely they connect with new agents.

AcceptConnection/RefuseConnection messages do not share new information: once again, they simply allow B to propagate information it already has and that is strictly needed to establish the connection. The information propagation is due to the AcceptedConnection messages, but they are strictly necessary.

In order to prove that the information provided is minimal, suppose we would like to provide *less* information. The only messages that increase the amount of information shared with someone are the RequestConnection and the AcceptedConnection. Firstly, without RequestConnection, C could not approve a friendship with someone he does not even know (and is guaranteed that if he refuses, A will not know anything); secondly, without AcceptedConnection, B could not inform A that a new friend is found.

B. The representation of profiles

In social network systems users enter data in a variety of ways. The most structured part is their user profile: this is essentially filled through some web form with fields for relevant entries; the exact nature of the data users put in their profiles is related with the kind of social network, e.g., whether oriented towards career [4] or hobbies [6] [3]. Users provide data in many other ways: for example users may add pictures, posts, comments, video, and audio-clips. Even more data can be gathered saving the search queries the user more frequently uses, his browsing habits (in the social network system) and similar statistical data. In principle, this huge amount of data is gathered in order to offer better suggestion to the user, meaning both “more” contacts and “better” advertising.

In fact, since all this information is available to the software agents, we expect they can use it as effectively as a centralized system, although with less privacy concerns. Semantic structure can be obtained from textual comments and articles like in [19] through systems such as Lucene [20]. Semantic processing of images is an active research subject (e.g., [21]); however, satisfactory results can be obtained by means of manual “tagging”, which is a standard practice in existing social networks.

However, from an abstract point of view, we prefer to use the non restrictive assumption that all the relevant information is in a profile written using the Resource Description Framework (RDF) [22] [23]. The assumption is non restrictive because we could simply add every datum gathered with other means to such RDF profile.

Friend Of A Friend (FOAF) [24] is a widespread machine-readable ontology describing persons, their activities and their relations to other people and objects; moreover, FOAF is extensible. We use the popular extensions Description Of A Career (DOAC) [25] and Description Of A Project (DOAP) [26]. In essence, FOAF is a descriptive vocabulary expressed using RDF and the Web Ontology Language (OWL) [27].

The idea behind FOAF is that there should not be any centralized database. However, a minor problem in the model FOAF proposes is that every profile is meant to be entirely public, while we prefer to let the agent decide which portions are accessible to whom. This issue has already been addressed in some systems [28] [29], although their main focus appears to be distributed authentication.

In a FOAF profile, the owner is put in relation with other entities, such as the Schools and Universities where he studied, the companies he worked for, sport societies or clubs he frequents. The idea behind the connection discovery algorithm is to use this data to find similarities and possible acquaintances among users. For example, if a user attended a given university, he is likely to know other people who attended the same school of university. The connection essentially uses a third entity, which differentiates different connections between the same users; with a small abuse of language we say that the connection is "typed". For example, two persons may be connected through an "attended University of Parma" link. In Fig. 2 users P1 and P2 both hold a degree in Computer Engineering at the University of Parma and this is the type of their connection. This way it is possible to derive relationships among users from elements in their profile.

This is the principal way to form new connections in our distributed social network system. However, the standard way people can be related to other people directly in FOAF is through the "knows" term [30]; the FOAF specification requires that the term should be used only if some kind of reciprocity exists (especially mentioning stalking as a limit case of *not* knowing). However, since each user owns and controls his FOAF profile, our system cannot enforce the requirement. In other words, we cannot automatically remove knows-entries from user profiles if the user claiming the acquaintance is not connected to the other user. We believe that users are the sole owners and responsible party of their profile; consequently, we do not edit them against their will, even if they are doing wrong.

If a white-pages service mapping foaf:Person's to Agent IDs exists, it is possible to use foaf:knows terms to establish new connection in a more direct way than using the connection discovery algorithm. This step can also be used in order to create some more connections to bootstrap the connection discovery algorithm itself. In this case the white-pages service can be used as a broker.

Suppose that A has a knows-entry in its profile for Person B. Then A can ask the white-pages service for the id of the agent corresponding to Person B. The service sends a message RequestConnection(A, knows) to Agent B and then the negotiation can proceed as in the usual case.

More precise kinds of relationships have been part of the FOAF ontology, but they were removed because "they were somewhat awkward to actually use, bringing an inappropriate air of precision to an intrinsically vague concept" [30]. However, extensions [31] have been proposed.

IV. HDS

HDS (Heterogeneous Distributed System) [32] [33] is a software framework merging the client-server and the peer-to-peer paradigms, whose goal is to simplify the realization of distributed applications. HDS implements all the interactions among the system processes through the exchange of typed messages. In particular, HDS provides both active and passive processes (respectively called actors and servers) and an application can be distributed on a (heterogeneous) network of computational nodes (from now on called runtime nodes).

The software architecture of a HDS application can be described through the three different models:

- the concurrency model, which describes how the processes of a runtime node can interact and share resources.
- the runtime model, which describes the services available for managing the processes of an application.
- the distribution model, which describes how the processes of different runtime nodes can communicate.

A. Concurrency Model

A process can interact with the other processes through the exchange of messages based on one of the following three types of communication: i) synchronous communication, the process sends a message to another process and waits for its answer; ii) asynchronous communication, the process sends a message to another process, performs some actions and then waits for its answer; iii) one-way communication, the process sends a message to another process, but it does not wait for an answer.

Actors can start communication of all the three types with

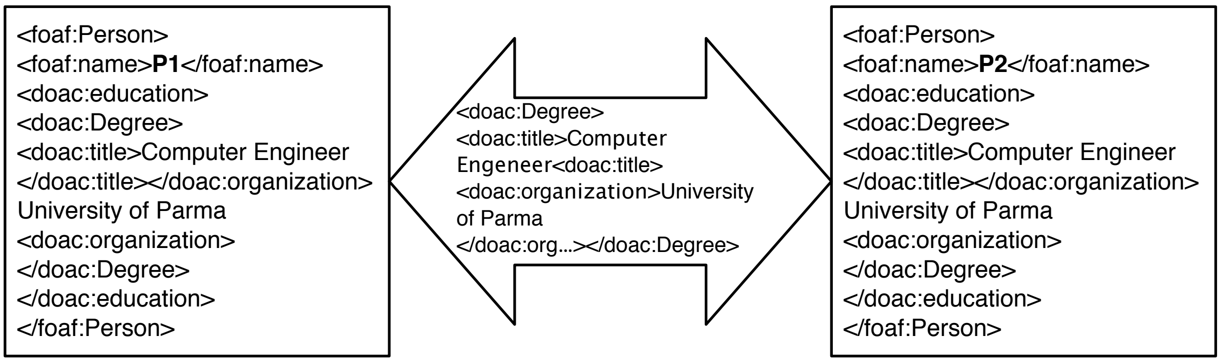

Figure 2. A link between users based on information in their FOAF profile.

every other process, while servers can only answer to requests and compose services provided by other servers through synchronous communication.

Processes delegate the task of exchanging messages with the other processes to a mailer. Mirroring the distinction between actors and servers, there are actor mailers and server mailers. Mailers both send the messages and keep a queue of received messages. These messages are rich objects, with header fields and a special content object holding the actual payload of the message. The content also determines the "type" of the message (much in an object oriented sense).

HDS features message filters that can: i) modify the normal delivery of messages; ii) manipulate the messages themselves (e.g., encrypt and decrypt); iii) provide additional capabilities, such as replication or logging services. Message filters are essentially composition filters [34]. Each agent has two lists of message filters: the ones of the first list, called input message filters, are applied to the input messages and the others, called output message filters, are applied to the output messages. When a new message arrives or is sent, the message filters of the appropriate list are applied to it in sequence until a message filter fails; therefore, such a message is stored in the input queue or is sent only if all the message filters have success.

B. *Runtime Model*

The runtime model defines the basic services provided by the middleware to the processes of an application. This model is based on four main elements: registry, processer, filterer and porter.

The registry is the runtime service responsible for the discovery of the processes of the application: i) it binds and unbinds the processes with their identifiers; ii) it provides a list of process identifiers; iii) it returns a reference on the basis of the process identifier. References are essentially proxies of the process; they make transparent the communication with respect to the process location. In order to send a message to another process, it is mandatory to obtain that process reference.

The processer is the runtime service responsible for the creation of new processes (and the related mailer) in the local runtime node. The creation is performed on the basis of the qualified name of the class implementing the process and a list of initialization parameters.

Since processes cannot directly modify the lists of message filters, the services of a filterer are needed. A filterer allows the creation and modification of the lists of message filters associated with the processes of the local runtime node.

Finally, a porter is a runtime service responsible for the creation of ports, which are special objects allowing an external application to use the services implemented by a server of the local runtime node. In essence, a port wraps a server and can: i) limit access to the process functionalities; ii) hide the use of some its services; iii) add some constraints on the use of some of its services.

C. *Distribution Model*

The distribution model defines the software infrastructure that allows the communication of a runtime node with the other nodes of an application, possibly through different types of communication supports, thus guaranteeing a transparent communication among their processes. This model is based on three kinds of element: distributor, connector and connection.

A distributor manages the connections with the other runtime nodes of the application. Each distributor manages connections that can be realized with different kinds of communication technology through the use of different connectors. Moreover, a pair of runtime nodes can be connected through different connections. A connector is a connection handler that manages the connections of a runtime node using a specific communication technology and allows the exchange of messages between the processes of the accessible runtime nodes that support such a communication technology.

A connection is a mono-directional communication channel that provides the communication between the processes of two runtime nodes through the use of remote references. In particular, a connection provides a remote lookup service offering the listing of the remote processes and the access to their remote references.

V. System Implementation

Given the extremely modest platform requirements of the system described in Section III, nearly any middleware framework could be used. We chose HDS because of its simplicity.

Each agent in our system has two main tasks: a) it uses information in the profile in order to discover new friendships and acquaintances on his owner's behalf and b) it mediates access to the profile information, allowing or refusing queries from other agents. While task b) does not need a full-fledged software agent, since a simple rule-based strategy suffices, task a) exhibits a typical proactive behavior, as agents actively pursue their owner's goal, without direct human intervention.

Task a) is accomplished using the connection discovery algorithm and has been implemented with three HDS processes: process i) searches new connections and friendships according to the data available; process ii) brokers connections between possibly mutual friends; process iii) accepts/refuses connections proposed by the first two processes. Processes i) and ii) are proactive, since they have to actively contact other agents, thus i) and ii) are HDS actors. Process iii), as well as the process implementing task b), are server processes.

In Section III, communication among agents is already described using typed messages. Those abstract messages can be mapped upon HDS typed messages, defining an appropriate Java class for each message.

In HDS every agent can query the registry to obtain a resource in order to send messages to it, but we require that only connected agents could communicate. The solution is to provide each agent with a pair of public/private keys. Every valid message is encrypted with the receiver's public key and signed with the sender's private key. Public keys are sent along with AcceptedConnection messages; consequently, only connected agents are able to communicate.

VI. Conclusion

In this paper we have presented a fully distributed social networking site, supporting user profiles stored as FOAF profiles. We presented an algorithm that suggests connections to the users, essentially constructing a social network through the information stored in their FOAF profile. Privacy is respected since: i) users can easily specify which data shall be used to construct their social network; ii) no central unit needs to access data in the user profile; iii) the amount of information that is propagated to users not directly connected is minimal; iv) the users receiving new information are friends of a friend and not total strangers. Moreover, we proposed the design of an implementation based on the HDS framework [33].

An experimental study will be carried out: i) to verify the effective construction of a user's social network, possibly using also foaf:knows terms and ii) to gather data for a more formal mathematical study. If the results would show that the information in the FOAF profile were not sufficient to activate a reasonable quantity of latent links, we propose to adapt the algorithms described in [35] to multi-agent systems, considering how the user can control both the quantity and the quality of information he actually shares (and with whom).

Eventually, we want to study algorithms to send messages to distant users using the social network constructed using the above techniques and examine the feasibility of a P2P file sharing application internal to the social network system, using the social network itself to route messages and files.

Acknowledgments

Stimulating discussions with Profs. A. Poggi and F. Bergenti are gratefully acknowledged.

References

[1] B. Wellman and S. D. Berkowitz, Social structures: a network approach, Jan 1997, JAI Press, p. 508.

[2] D. M. Boyd and N. B. Ellison, "Social Network Sites: Definition, History, and Scholarship," Journal of Computer-Mediated Communication, 13(1), article 11.

[3] Facebook. http://www.facebook.com/.

[4] LinkedIn. http://www.linkedin.com/.

[5] Facebook Policy. http://www.facebook.com/policy.php.

[6] Orkut. http://www.orkut.com.

[7] Orkut Content Ownership Policy. http://www.google.com/support/orkut/bin/answer.py?answer=57439.

[8] S. Milgram, "The Small-World Problem," Psychology Today, vol 1., May 1967, pp 61-67.

[9] J. Travers and S. Milgram," An experimental study of the small world problem," Sociometry, vol 3, Dec 1969, pp 425-443.

[10] P. Dodds, R. Muhamad, and D. Watts, "An Experimental Study of Search in Global Social Networks," Science, vol. 301, no. 5634, Aug 2003, p. 827.

[11] S. Wasserman and K. Faust, Social network analysis: Methods and applications, Cambridge University Press, 1994.

[12] D. Watts, P. Dodds, and M. Newman, "Identity and search in social networks," Science, vol. 296, May 2002, pp 1302-1305.

[13] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," Nature, vol. 393, June 1998, pp. 440-442.

[14] J. Kleinberg, "The small-world phenomenon: an algorithm perspective," Proceedings of the 32nd annual ACM on Theory of computing, 2000, pp 163-170.

[15] L. N. Foner, "Yenta: a multi-agent, referral-based matchmaking system," Proceedings of the First international Conference on Autonomous Agents, 1997, pp 301-307.

[16] H. Kautz, B. Selman, and M. Shah. "Referral Web: combining social networks and collaborative filtering," Communication of the ACM, vol. 40, Mar 1997, pp. 63-65

[17] P. Mika, "Flink: Semantic Web technology for the extraction and analysis of social networks," Web Semantics, vol. 3, 2005, pp. 211-223.

[18] Y. Matsuo, J. Mori, M. Hamasaki, T. Nishimura, H. Takeda, K. Hasida, and M. Ishizuka, "POLYPHONET: An advanced social network extraction system from the Web." Web Semantics. vol 5 (4), Dec. 2007, pp. 262-278.

[19] D. Shoujian and X. Youming, "Design and Implementation of Semantic Retrieval Model Based on Ontology and Lucene," Modern Electronics Technique, 2009.

[20] M. McCandless, E. Hatcher, and O. Gospodnetic, Lucene in action, 2nd ed., Manning Publications, 2008.

[21] M. Lux and S. Chatzichristofis, "Lire: lucene image retrieval: an extensible java CBIR library," Proceeding of the 16th ACM on Multimedia, 2008, pp.1085-1088.

[22] D. Beckett and B. McBride, "RDF/XML syntax specification (revised)," W3C recommendation, 2004.

[23] F. Manola, E. Miller, and B. McBride, "RDF primer," W3C recommendation, 2004.

[24] D. Brickley and L. Miller, Friend Of a Friend (FOAF), http://xmlns.com/foaf/spec/.

[25] R. Antonio, Description of a Career, http://ramonantonio.net/doac.

[26] E. Dumbhill, Description of a Project, http://trac.usefulinc.com/doap.

[27] G. Antoniou and F. Harmelen, "Web ontology language: Owl", Handbook on ontologies, Springer, 2009, pp 91-110.

[28] S. Kruk, A. Gzella, and S. Grzonkowski, "D-FOAF-Distributed Identity Management based on Social Networks," Proceedings of the 3rd European Semantic Web Conference (ESWC2006), 2006.

[29] H. Story, B. Harbulot, I. Jacobi, and M. Jones, "FOAF+SSL: RESTful Authentication for the Social Web," in European Semantic Web Conference, Workshop: SPOT2009, Heraklion, Greece, 2009.

[30] D. Brickley and L. Miller, "FOAF 'knows' specification," http://xmlns.com/foaf/spec/#term_knows.

[31] E. Vitiello, "A module for defining relationships in FOAF, http://www.perceive.net/schemas/20021119/relationship/.

[32] F. Bergenti, E. Franchi and A. Poggi, "Using HDS for realizing Multiagent Applications," Proceeding of the Third International Workshop on Languages, Methodologies and Development Tools for Multi-Agent Systems (LADS'010), Lyon, France, In press.

[33] A. Poggi and M. Tomaiuolo, "Use and Reuse of Multi-Agent Models and Techniques in a Distributed Systems Development Framework," in Seventh International Symposium "From Agent Theory to Agent Implementation," pp. 477-482, Vienna, Austria, 2010.

[34] L. Bergmans and M. Aksit, "Composing crosscutting concerns using composition filters," Communications of the ACM, vol. 44, Oct 2001, pp 51-57.

[35] M. Makrehchi and M. Kamel, "Learning social networks using multiple resampling method," International Conference on Systems, Man and Cybernetics, Oct 2007, pp. 406-411.