Agent based P2P Social Neworks Modeling

Agent based P2P Social Networks Modeling

Eleonora Iotti, Agostino Poggi and Michele Tomaiuolo
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma
Parma, Italy
{eleonora.iotti,agostino.poggi,michele.tomaiuolo}@unipr.it

***Abstract* — Nowadays, social networks are the most important means for the interaction between people on the Web. The large part of such networks are deployed on a centralized architecture that allows a simple browser-based user experience and, moreover, many algorithms, e.g., friend suggestion, are far easier and more efficient to implement in this setting. Peer-to-peer social networks do not exploit a central server for storing users' data. Therefore, their development and maintenance is more difficult, but they enable users to have more control on their profile content, ensuring a higher level of privacy. The main challenge of such a kind of network comes from guaranteeing availability of the data of the user profiles when their owners are offline. Different solutions have been proposed, but each of them presents advantages and drawbacks (e.g., data availability vs. cost). In this paper we present our preliminary work on the design of a peer-to-peer social network architecture that took advantage of an actor based development system for the modelling and analysis of a set of possible algorithms that can support the availability of the profiles of the offline users in the social network.**

Keywords—P2P social network; network analysis; agent based modeling and simulation; software development system; actor model.

I. INTRODUCTION

Online social networks are used by hundreds of millions of people every day and play the main role in the spread of information in the Internet. Even if the social networking systems are greatly dissimilar in their user base and functionality, they are almost always centralized systems. The centralized nature allows a simple browser-based user experience and, moreover, many algorithms, e.g., friend suggestion, are far easier and more efficient to implement in this setting.

Peer-to-Peer (P2P) define an open and decentralized overlay network on top of the Internet that users can use for directly communicating to find and share resources, often music and movie files [1]. Such networks are one of the few largest distributed computing systems ever, and more surprisingly, they can run with great stability and resilient performance in face of possibly the most ferocious dynamics [2].

Thus, the use of P2P technologies for the development of social networks is not only viable, but also highly desirable [3]. First of all, P2P systems essentially achieve automatic resource scalability, in the sense that the availability of resources is proportional to the number of users. This property is especially desirable for media sharing social networking systems, considering the exceptionally high amount of resources needed. Secondly, the popularity over time of most content on such systems exhibits either a power-law or an exponential behavior and is consequently well suited for P2P distribution [4], possibly with fallback strategies for less popular content. Finally, enable users to have more control on their profile content, ensuring a higher level of privacy and support to anonymity and resilience to censorship.

In a P2P social network there is no single provider but a set of peers that take on and share the tasks needed to run the system. The development of the existing functionalities of social networks in a distributed context requires finding ways for providing robustness against churn, distributing storage of data, propagating updates, defining an overlay topology and a protocol enabling searching and addressing, etc. One of the main challenge comes from guaranteeing the availability of a user profile even when she/he is offline. Some of the solutions rely on external storage systems, for example exploiting a distributed file system [5], while some other more recent approaches [6][7] propose to store the profile of a user on the storage support provided by users' friends. In these proposals, a user serves his own profile when she/he is online, and elects a subset of his friends to make the profile available when he is offline.

This paper presents an actor based development system, ActoDeS, (Actor Development System) providing a set of suitable software components for the modeling and simulation of social networks and the analysis of their results and its use for the design of the architecture of a P2P social network. The next section introduces related work. Section 3 provides an overview of the software framework. Section 3 describes the features of such development system and shows how it makes easy the developing of agent based models and simulations (ABMS). Section 4 introduces our modelling and analysis work on a set of possible algorithms for supporting the persistent availability of the data to the offline users of a P2P social network. Finally, section 6 concludes the paper by discussing its main features and the directions for future work.

II. RELATED WORK

A lot of work has been done for the development of agent-based software platforms that can be also used for the modelling and simulation of complex networks. Moreover,

several researchers propose solutions for supporting the availability of the profiles of the offline P2P social networks. The rest of the section presents some of the most interesting works on the previous two topics.

Swarm [8] is the ancestor of many of the current ABMS platforms. The basic architecture of Swarm is the simulation of collections of concurrently interacting agents, and this paradigm is extended into the coding, including agent inspector actions as part of the set of agents. So in order to inspect one agent on the display, you must use another hidden, non-interacting agent. Swarm is a stable platform, and seems particularly suited to hierarchical models. Moreover, it supports good mechanisms for structure formation using multi-level feedback between agents, groups of agents, and the environment (all treated as agents).

Ascape [9] is a framework for developing and analyzing agent based models following some of the ideas of Swarm. However, it is somewhat easier to develop models with Ascape than with Swarm. Indeed, its goal is to allow people with only a little programming experience to develop quite complex simulations by providing a range of end user tools. Ascape is implemented in Java and users would require some ability to program in Java together with understanding of the object orientation philosophy.

NetLogo [10] is an ABMS platform based on the Logo programming language. Its initial goal was to provide a high-level platform allowing students, down to the elementary level, to build and learn from simple ABMS applications. Now it offers many sophisticated capabilities and tools that make it suitable for complex applications too. Moreover, a big advantage respect to the other platforms is the simplicity of its own language.

Repast [11] is a well-established ABMS platform with many advanced features. It started as a Java implementation of the Swarm toolkit, but rapidly expanded to provide a very full featured toolkit for ABMS. Although full use of the toolkit requires Java programming skills, the facilities of the last implementations allow the development of simple models with little programming experience [12].

MASON [13] is a Java ABMS tool designed to be flexible enough to be used for a wide range of simulations, but with a special emphasis on “swarm” simulations of a very many (up to millions of) agents. MASON is based on a fast, orthogonal, software library to which an experienced Java programmer can easily add features for developing and simulating models in specific domains.

PeerSoN [6] is a prototype of P2P social network designed to provide encryption, decentralization and direct data exchange in the field of social networks. A DHT is used to trace the user's network presence and for obtaining the index of the user's recent content. However, this DHT is logically a separate and central entity that is could become a bottleneck and single point of failure of the social network

Conti et al. [14] present a distributed storage support which guarantees the users' data persistence of a social network. In their system, users dynamically elect a minimal set of point of storage among their friends and their data are dynamically transferred between online users in order to maximize the availability of users' profiles in the social network.

My3 [15] is a privacy-friendly decentralized online social network that exploits some interesting features of the current online social networks (i.e., locality of access, predictable access times, friends geo-localization, unique access requirements of the social content, and implicit trust among friends. It particular, it proposes different replication strategies that support users’ profile availability, access delay, freshness and storage load.

III. ACTODES

ActoDeS is an actor based software framework that has the goal of both simplifying the development of concurrent and distributed complex systems and guarantying an efficient execution of applications.

ActoDeS is implemented by using the Java language and takes advantage of preexistent Java software libraries and solutions for supporting concurrency and distribution. ActoDeS has a layered architecture composed of an application and a runtime layer. The application layer provides the software components that an application developer needs to extend or directly use for implementing the specific actors of an application. The runtime layer provides the software components that implement the ActoDeS middleware infrastructures to support the development of standalone and distributed applications.

In ActoDeS an application is based on a set of interacting actors that perform tasks concurrently. An actor is an autonomous concurrent object, which interacts with other actors by exchanging asynchronous messages [16]. Moreover, it can create new actors, update its local state, change its behavior and kill itself.

Communication between actors is buffered: incoming messages are stored in a mailbox until the actor is ready to process them; moreover, an actor can set a timeout for waiting for a new message and then can execute some actions if the timeout fires. Each actor has a system-wide unique identifier called reference that allows it to be reached in a location transparent way. An actor can send messages only to the actors of which it knows the reference, that is, the actors it created and of which it received the references from other actors. After its creation, an actor can change several times its behavior until it kills itself. Each behavior has the main duty of processing a set of specific messages through a set of message handlers called cases. Therefore, if an unexpected message arrives, then the actor mailbox maintains it until a next behavior will be able to process it.

An actor can be viewed as a logical thread that implements an event loop [17][18]. This event loop perpetually processes events that represent: the reception of messages, the behavior exchanges and the firing of timeouts. The life of an actor starts from the initialization of its behavior that then processes the received messages and the firing of message reception timeouts. During its life, an actor can move from a behavior to another one more times, and its life ends when it kills itself.

ActoDeS provides different actor implementations and the use of one or of another implementation represents one of the factors that mainly influence the performance of an application. In particular, actor implementations can be divided in two classes: active actors, i.e., actors that have their own thread of execution, and passive actors, i.e., actors that share a single thread of execution. In this last case, the scheduler has the duty of guaranteeing a fair execution of all the actors.

IV. ACTODES AND ABMS APPLICATIONS

The features of the actor model and the flexibility of its implementation make ActoDeS suitable for building ABMS applications and for analyzing the results of the related simulations [19]. In particular, actors have the suitable features for defining agent models that can be used in ABMS applications and to model the computational agents found in MAS) and DAI systems. In fact, actors and computational agents share certain characteristics: i) both react to external stimuli (i.e., they are reactive), ii) both are self-contained, self-regulating, and self-directed, (i.e., they are autonomous), and iii) both interact through asynchronous messages and such messages are the basis for their coordination and cooperation (i.e., they are social). Moreover, given that actors interact only through messages and there is not a shared state among them, it is not necessary to maintain an additional copy of the environment to guarantee that agents decide their actions with the same information (thing that is usually necessary in some application domain with other ABMS platforms). Finally, the use of messages for exchanging state information decouples the code of agents. In fact, agents do not need to access directly to the code of the other agents to get information about them, and so the modification of the code of a type of agent should cause lesser modifications in the code of the other types of agent. Finally, the use of actors simplifies the development of real computational agents in domain where, for example, they need to coordinate themselves or cooperate through direct interactions.

Moreover, the use of ActoDeS simplifies the development of flexible and scalable ABMS applications. In fact, the use of active and passive actors allows the development of applications involving large number of actors, and the availability of different schedulers and the possibility of their specialization allow a correct and efficient scheduling of the agents in application domains that require different scheduling algorithms [20]. Moreover, the efficient implementation of broadcasting and multicast removes the overhead given to the need that agents must often diffuse the information about their state to the other agents of the application (e.g., their location in a spatial domain).

A. Simulation

In large part of ABMS platforms usually a simulation is given by a sequence of steps where each agent needs only to get information about its surround (i.e., about a subset of the other agents and about the environment) and then to use such information for deciding its actions.

In ActoDeS the simulation is similar, but agents get information about agents and the environment through messages. Moreover, to simplify the interaction between agents and the environment, the relevant parts of an environment are represented by a set of actors whose goals are to inform the agents acting in the environment about their presence and their state, and to update their state when the agents act on them. Given that the behavior of such actors is similar to the one expressed by the agents acting in the environment, we call both agents, but we divided them in active and passive agents. Active agents are the typical agents of an ABMS, i.e., they represent the entities able to move and cooperate with other entities acting in the environment. Passive agents define the environment of an ABMS, i.e., they represent the relevant elements of the environment (e.g., in a spatial domain the obstacles and the reference points for the movement of the active agents).

Such agents are usually implemented taking advantage of the shared actor implementation provided by ActoDeS, but it is necessary to develop a specific scheduler. Such a scheduler executes repeatedly all the agents and after each execution step broadcasts them a “clock” message. This last message allows to the agents to understand that they have all the information for deciding their actions, therefore, they decide, perform some actions and, at the end, broadcast the information about their new state.

In ActoDeS, all the agents are usually represented by one or more actor behaviors that process the input messages through two cases. The first case processes the messages informing an agent about the state of the other agents. The second case processes the “clock” messages. However, while active agents exchange messages and perform other types of action (e.g., in a spatial domain to change their location), often, passive agents have the only duty of sending messages for informing the active agents about their presence (e.g., immutable obstacles or path points in a spatial domain). Therefore, such passive agents are represented by an actor behavior providing a case that get the “clock” messages for deciding when sending the information about their presence and state.

Of course, different types of agent have different implementations of the cases of their behaviors. In particular, ActoDeS provides some abstract behavior implementations for developing applications in different domains. Such implementations define the state information that an agent need to maintain in its specific application domain and provides a set of abstract methods for processing incoming information and for performing the actions in response to the “clock” messages.

Often the modelling of some systems (e.g., social networks) requires a massive number of agents. However, in such kind of systems, usually only a part of them is simultaneously active and the actions of the different agents do not need a synchronization. Therefore, it is necessary a scheduler that can manage a massive number of agents, but that can try to optimize the execution by scheduling only the active agents. The solution we implemented derives from the virtual memory techniques used by operating systems: agents increment an inactivity counter in the scheduling cycles in which they do not process messages and reset it in the cycles in which they process a message. The scheduler can get the value of such counters and can move an actor in a persistent store when its

inactivity counter becomes greater than a fixed (or dynamic) threshold. The scheduler reloads an actor from the persistent store when it receives a new message from another agent.

Of course, the number of active agents can vary over the simulation, but the quality of the simulation can be guaranteed if the number of the agents, maintained by the scheduler, remains in a range that depends on the available computational resources. The adopted solution, to limit to the number of active actors and to guarantee good performances, is to provide a scheduler able to move an inactive agent in the persistent storage on the basis of a variable number of inactive cycles. In particular, this number is low when there is a large number of scheduled agents and high when there are few scheduled agents (i.e., the scheduler spends time for storing agents in the persistence storage and reloading them only when there may be memory problems for maintaining all the actors).

B. Data Analysis

After a simulation is important to summarize and analyze the results, in a way that will yield maximum insight and help with decision-making. However, before the analysis is necessary to define the data that a simulation must provide as its result and must write the code necessary for generating such data during the simulation.

ActoDeS provides a logging service that allows the recording (as serialized objects) of the information about the relevant actions of actors of the simulation (i.e., initialization, reception, sending and processing of messages, creation of actors, change of behavior, and its shutdown). In particular, such an information allows to get the state of each actor for any simulation cycle. Therefore, it is possible to analyze all the information about both the interaction among the actors and the dynamics of their state.

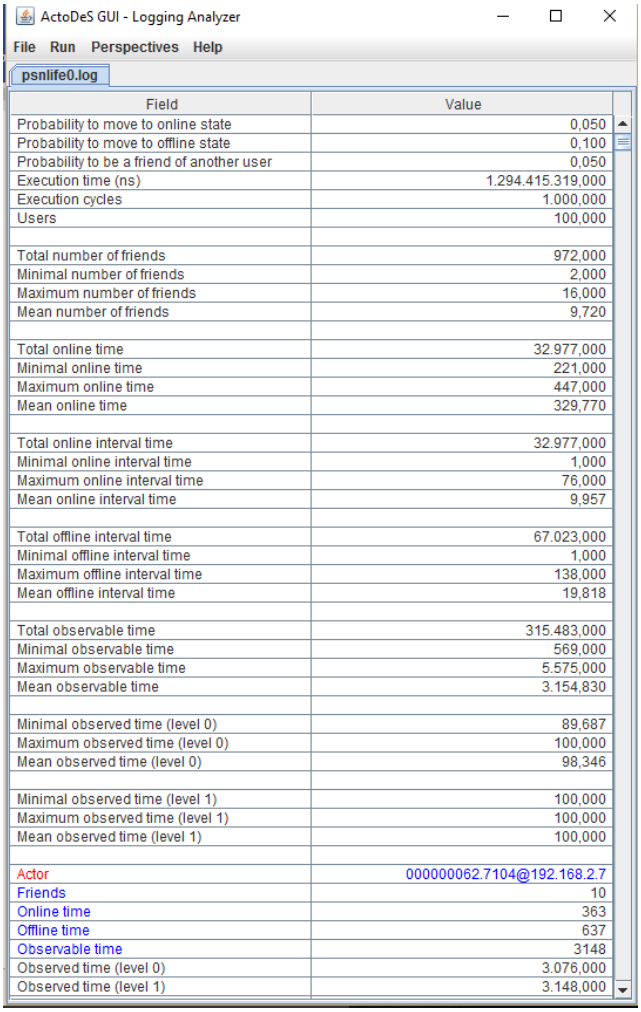
Moreover, ActoDeS provides some tools for filtering the logging information and extracting statistical data from such information and offers a simple API to summarize the results in tables and charts.

V. MODELLING P2P SOCIAL NETWORKS

In the last year, our work has been mainly oriented to the analysis and simulation of social networks [21][22][23]. In particular, currently we are working on the modeling of P2P social networks [24] and we are using ActoDeS for designing a peer-to-peer social network that supports the availability of the profiles of the offline users in the social network. In fact, the problem of P2P social networks is that they do not have a centralized service that maintain the information shared among the users and so, for example, is difficult for a user that wakes up after a period of inactivity to get the last information about the users that moves to the offline state during her/his last inactivity time.

Our work was divided in two phases. The first phase had the goal of measuring the level of users' profile availability provided by the algorithms presented in [14] and in [15] and by two simple algorithms that replicated the users' profile on all the online friends. In particular, the third algorithm allows that a user gets a friend's profile from another friend, and the forth one allows that a user gets a friend's profile from any user of the network that is friend of the offline user, (i.e., she/he might be not a friend of the user that require the profile). The second phase (that is not yet terminated) has the goal of measuring their costs and performances of the algorithms and finding solutions for the cases in which the availability of the profiles of the offline users cannot be guaranteed from the simple interaction among the online users.

The first work of the first phase was the generation of the data of a set of social networks where a thousand of users have different values of the probability for becoming friends and different values of probability for moving from the online to the offline state and vice versa. Such data was obtained through a set of simulations of the social networks where their users are modelled as simple agents that are created with a set of friends (on the basis of the friendship probability value) and that can decide to move between the online and offline state (on the basis of the probability value assigned in the simulation).



Field	Value
Probability to move to online state	0,050
Probability to move to offline state	0,100
Probability to be a friend of another user	0,050
Execution time (ns)	1.294.415.319,000
Execution cycles	1.000,000
Users	100,000
Total number of friends	972,000
Minimal number of friends	2,000
Maximum number of friends	16,000
Mean number of friends	9,720
Total online time	32.977,000
Minimal online time	221,000
Maximum online time	447,000
Mean online time	329,770
Total online interval time	32.977,000
Minimal online interval time	1,000
Maximum online interval time	76,000
Mean online interval time	9,957
Total offline interval time	67.023,000
Minimal offline interval time	1,000
Maximum offline interval time	138,000
Mean offline interval time	19,818
Total observable time	315.483,000
Minimal observable time	569,000
Maximum observable time	5.575,000
Mean observable time	3.154,830
Minimal observed time (level 0)	89,687
Maximum observed time (level 0)	100,000
Mean observed time (level 0)	98,346
Minimal observed time (level 1)	100,000
Maximum observed time (level 1)	100,000
Mean observed time (level 1)	100,000
Actor	000000062.7104@192.168.2.7
Friends	10
Online time	363
Offline time	637
Observable time	3148
Observed time (level 0)	3.076,000
Observed time (level 1)	3.148,000

Fig. 1. Analysis results of the two simple algorithms.

After that, we defined the four agent models that implement the four algorithms for providing users' profile availability introduced above, then we performed the simulations, and,

finally, we analyzed the results. The results of the analysis show that the two fourth algorithm provides better users' profile availability that the others. Of course, the results show also that the level of availability of the four algorithms depends on the number of friends and on time that each user spends online. Fig. 1 show the analysis results of both the third (level 0) and the fourth (level 1) algorithms for a specific set of probability values. However, a complete comparison of the four algorithms should take into account the costs (e.g., number of replicas, transfer of the profile data, algorithms for selecting the node where replicated the profile data, …) and the level of reliability provided by the four algorithms. We are in the middle of this activity and we hope to have the possibility to present the results in the next future.

VI. CONCLUSIONS

This paper presented a software development system, ActoDeS, (Actor Development System) that offers a set of suitable software components for the modeling and simulation of social networks and the analysis of their results.

ActoDeS is implemented by using the Java language and is an evolution of CoDE [25] that simplifies the definition of actor behaviors and provides more scalable and performant implementations. Moreover, it takes advantages of some implementation solutions used in JADE [26] for the definition of some internal components.

Current and future research activities are and will be dedicated to complete the analysis of the algorithms for supporting the availability of offline users' profiles and to start the analysis of the possible techniques to provide a good level of security in P2P social networks [27][28] and use the same techniques for the design of computer-supported cooperative work systems [28].

REFERENCES

[1] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in Proc. 1st Int. Conf. on Peer-to-Peer Computing Linköping, Sweden, 2001, pp. 101-102.

[2] D. Qiu, and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," ACM SIGCOMM Computer Communication Review, Vol. 34, No. 4, pp. 367-378, 2004.

[3] F. Wang, Y. Moreno and Y. Sun, "Structure of peer-to-peer social networks," Physical Review E, Vol. 73, No. 3, pp. 1-8, 2006.

[4] M. Zink, K. Suh, Y. Gu and J. Kurose, "Characteristics of YouTube network traffic at a campus network - Measurements, models, and implications," Computer Networks, Vol. 53, No. 4, pp. 501-514, 2009.

[5] I. Clarke, O. Sandberg, B. Wiley and T. Hong, T. "Freenet: A distributed anonymous information storage and retrieval system," in Designing Privacy Enhancing Technologies, LNCS, Vol. 2009, Berlin, Germany: Springet, 2001, pp. 46–66.

[6] S. Buchegger, D. Schiöberg, L. Vu, A. Datta, "PeerSoN: P2P social networking: early experiences and insights," in Proc 2nd ACM EuroSys Workshop on Social Network Systems, Nuremberg, Germany: ACM, 2009, pp. 46–52.

[7] R. Baden, A. Bender, N. Spring, B. Bhattacharjee and D. Starin, "Persona: an online social network with user-defined privacy," in Proc. Conf. on Data Communication Barcellona, Spain, 2009, pp. 135-146.

[8] N. Minar, R. Burckhart, C. Langton, and V. Askenasi, 1996. "The Swarm simulation system: a toolkit for building multi-agent systems," Santa Fe Institute, Santa Fe, NM, USA. [Online] Available http://www.swarm.org.

[9] M. T. Parker, "What is Ascape and why should you care," Journal of Artificial Societies and Social Simulation, vol. 4, no. 1, 2001.

[10] S. Tisue, and U. Wilensky, "Netlogo: A simple environment for modeling complexity," in Proc. Int. Conf. on Complex Systems (ICCS 2004), 16-21, Boston, MA, USA, 2004, pp. 16-21.

[11] M. J. North, N. Collier, and J. Vos, "Experiences in creating three implementations of the repast agent modeling toolkit," ACM Trans. on Modeling and Computer Simulation, vol. 16, no. 1, pp. 1-25, 2006.

[12] M. J. North, T. R. Howe, N. T. Collier, and J. R. Vos, "The Repast Simphony runtime system," in Proc. Conf. on Generative Social Processes, Models, and Mechanisms, Chicago, IL, USA, 2005

[13] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A multiagent simulation environment," Simulation, vol. 81, no. 7, pp. 517-527, 2005.

[14] M. Conti, A. De Salve, B. Guidi, F. Pitto, and L. Ricci, "Trusted dynamic storage for dunbar-based P2P online social networks," In Prpc. OTM Int. Conf. on the Move to Meaningful Internet Systems, Berlin, Germany: Springer, 2014, pp. 400-417.

[15] R. Narendula, T. G. Papaioannou and K. Aberer, "A decentralized online social network with efficient user-driven replicatio," in Proc. Int. Conf. on Social Computing, Amsterdam, The Netherlands: IEEE, 2012, pp. 166-175.

[16] G.A. Agha, "Actors: A Model of Concurrent Computation in Distributed Systems," Cambridge, MA, USA: MIT Press, 1986.

[17] J. Dedecker, T. Van Cutsem, S. Mostinckx, T. D'Hondt and W. De Meuter, "Ambient-oriented programming in ambienttalk," in ECOOP 2006 – Object-Oriented Programming, Berlin, Germany: Springer, 2006, pp. 230-254.

[18] M. S. Miller, E. D. Tribble, and J. Shapiro, "Concurrency among strangers," in Trustworthy Global Computing, Berlin, Germany: Springer, 2005, pp. 195-229.

[19] A. Poggi, "Agent based modeling and simulation with ActoMoS," in Proc. 16th Workshop on From Object to Agents (WOA 2015), Naples; Italy, 2015, pp. 91-96.

[20] P. Mathieu, and Y Secq, "Environment Updating and Agent Scheduling Policies in Agent-based Simulators," in Proc. 4th Int. Conf. on Agents and Artificial Intelligence, Algarve, Portugal, 2012, pp. 170-175.

[21] F. Bergenti, E. Franchi, and A. Poggi,. "Selected models for agent-based simulation of social networks," in Proc. 3rd Symp. on Social Networks and Multiagent Systems (SNAMAS 2011), York, UK. 2011, pp. 27-32.

[22] F. Bergenti, E. Franchi and A. Poggi, "Agent-based interpretations of classic network models," Computational and Mathematical Organization Theory, Vol. 19, No. 2, 2013, pp. 105-127, 2013.

[23] E. Franchi, A. Poggi, and M. Tomaiuolo. "Open social networking for online collaboration," International Journal of e-Collaboration, vol. 9, no. 3, pp. 50-68, 2013.

[24] E. Franchi, A. Poggi and M. Tomaiuolo, "Blogracy: A Peer-to-Peer Social Network," International Journal of Distributed Systems and Technologies, Vol. 7, No. 2, pp.37-56, 2016.

[25] F. Bergenti, A. Poggi, and M. Tomaiuolo, "An Actor Based Software Framework for Scalable Applications," in Internet and Distributed Computing Systems, Berlin, Germany: Springer, 2014, pp. 26-35.

[26] A. Poggi, M. Tomaiuolo, and P. Turci, "Extending JADE for agent grid applications," in Enabling Technologies: Infrastructure for Collaborative Enterprises, Modena, Italy: IEEE, 2004, pp. 352-357.

[27] A. Poggi, M. Tomaiuolo and G. Vitaglione, "A Security Infrastructure for Trust Management in Multi-agent Systems," in Trusting Agents for Trusting Electronic Societies, Theory and Applications in HCI and E-Commerce, Berlin, Germany: Springer, 2005, pp. 162-179.

[28] E. Franchi, A. Poggi, and M. Tomaiuolo, "Information and password attacks on social networks: An argument for cryptography," Journal of Information Technology Research (JITR), Vol. 8, No. 1, 2015, pp.25-42.

[29] F. Bergenti and A. Poggi. "An agent-based approach to manage negotiation protocols in flexible CSCW systems," in Proc. 4th Int. Conf. on Autonomous agents, ACM, 2000, pp. 267-268.