EAI Endorsed Transactions
on Creative Technologies

Research Article ICST.ORG

Mobile Agent Communication in Highly Dynamic Networks: A Self-Adaptive Architecture inspired by the Honey Bee Colony

Phuong T. Nguyen[1,*], Volkmar Schau[2], Wilhelm R. Rossak[2]

[1]Research and Development Center, Duy Tan University, Da Nang, Vietnam
[2]Department of Computer Science, Friedrich-Schiller-Univerität Jena, Ernst-Abbe-Platz 2-4, D-07743 Jena, Germany

Abstract

Communication is considered as a building block for mobile agent systems. In highly dynamic networks, thanks to environmental stimuli such as changes in connection quality and network topology, performance of communication between mobile agents may be degraded considerably. With focus on attaining fault tolerance and reliability, we propose a context-aware architecture for agent communication model inspired by the honey bee colony. To validate the hypothesis, a software prototype has been designed and implemented according to the proposed mechanism. Encouraging experimental results on a test system show that our approach brings benefits to a colony of agent platforms.

Received on 29 October 2014; accepted on XXXX; published on 16 December 2014

Keywords: Context-Awareness, Mobile Agents, Agent Communication

doi:10.4108/casa.1.2.e5

1. Introduction

Communication between mobile agents has been identified as an important issue in mobile agent technology, alongside other technical aspects of mobile agent technology, such as migrations and control mechanisms [9],[10],[11]. Under certain circumstances mobile agents can benefit from sending communication messages instead of migrating to remote platforms [9],[13].

For systems working in highly dynamic networks, agent communication is expected to adapt to environmental stimuli which appear only at execution time. Communication should deal not only with interoperability and location-transparency, but also with adaptability. Several studies have been published so far which address the design of an appropriate, highly flexible model for mobile agent communication. However, to the best of our knowledge none of the suggested solutions has been able to achieve the necessary performance and quality attributes to count as a practical solution. In most cases, these existing approaches seem to neglect the inherent dynamics of modern networks. Thus, in our view adaptive mobile agent communication remains a challenging topic. We are interested in developing an adaptive communication model for mobile agents which is suitable for tasks in highly dynamic networks.

In this paper we present an adaptive communication for mobile agents in highly dynamic networks. The main objective is to develop a model for agent communication that supports fault tolerance and reliability as the key qualities. Being inspired by the exotic behaviours of the honey bee colony, we suggested building a self-organizing network in the agent platform colony. Based on the proposed theory, we design, implement a software prototype for supporting mobile agent communication in highly dynamic networks. To validate our hypothesis, we also perform some evaluations for the software prototype in a laboratory scale test system.

The paper is organized as follows. In Section 2 we present the motivations as well as the aims of our work. Background on swarm intelligence is going

*Corresponding author. Email: phuong.nguyen@duytan.edu.vn

- t_d is the time to deserialize a scout agent from an incoming stream of byte.
- t_s is the time to serialize a scout agent into a byte stream to be sent over the network.

By measuring the processing time, we examine how fast a scout agent performs its tasks at a specific platform.

The average exploration time is computed:

$$t_{exploration} = t_{end} - t_{begin} \tag{4}$$

where t_{begin} and t_{end} are the time when a scout starts and completes one round of exploration, respectively. This paramater indicates how often a scout agent supplies a RegionMaster with up-to-dated information for the self-organizing activities.

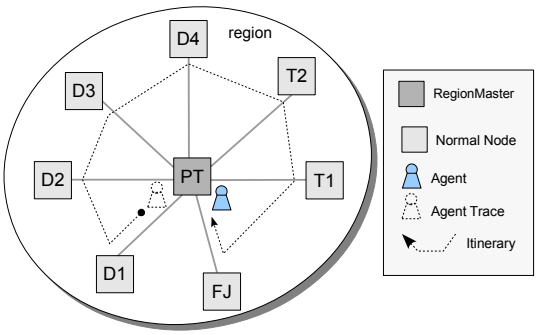

Figure 7. Network Setup

Figure 7 shows the logical connection for the first evaluation test. In this scenario, one scout agent is created at RegionMaster. The RegionMaster assigns the list of agent platforms $\{D1, D2, D3, D4, FJ, PT, T1, T2\}$ to the scout agent. One round of surveillance takes place when the agent is sent by RegionMaster, travels around the nodes of the region, performs its routine at every node and then migrates back to RegionMaster.

To get a more reliable result, by every platform a measurement is done for 80 times. The box plot diagram in Figure 8 depicts the processing time $t_{processing}$ of the platforms. This parameter represents the time that the agent needs to perform its tasks at that platform. It is dependent on the processing power of agent platforms as well as the latencies to the other platforms, which are in turn dependent on the network connection speed. It can be seen that, except platform $T2$ that has a higher processing time because of its limited processing power (as specified in Table 2), the processing time for the other platforms is considerably low. It guarantees that the processing activities place comparatively little burden on the system performance.

To measure $t_{exploration}$, the agent is sent around the network for different numbers of rounds r. In

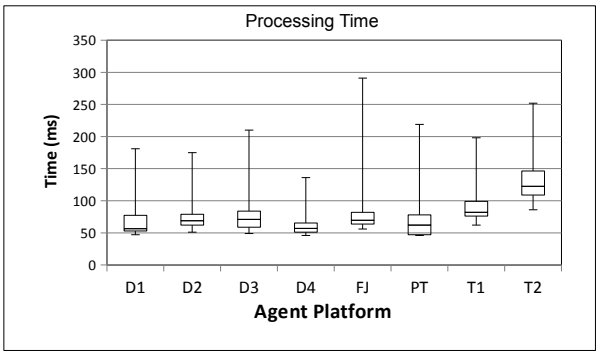

Figure 8. Processing time

the second experiment r is set to different values, i.e. $r = \{100; 200; 300; 500; 1000; 2000; 3000; 5000\}$. The average exploration time is given by:

$$t_{exploration} = \frac{t_{end} - t_{begin}}{r} \tag{5}$$

This parameter demonstrates how fast the scout agent can supply RegionMaster with up-to-date information about the network. If the agent needs a long time to perform its tasks at the platforms or to hop from platform to platform, the information submitted to RegionMaster might be out-of-date. As a consequence, the reactions produced by RegionMaster would not be adequate.

However, the measurement results show that this is not the case. In the diagram in Figure 9, the curve represents the accumulated exploration time. The straight line which depicts the average time for finishing one round of surveillance (second/round) provides evidence that the parameter is stable, no matter how many rounds the agent has migrated. This indicates that the scout agent produces no overhead when it works in the long run.

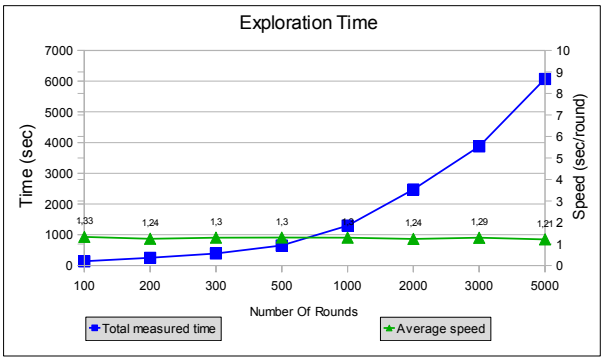

Figure 9. Average time for a scout agent to explore the region

8.3. Self-Organizing

In a dynamic network it is a common occurrence that a group of platforms leaves other platforms of the same region and swarms onto a new location, causing the connectivity to change correspondingly. Under the circumstances, maintaining a fixed logical connection between the abandoning platforms and the stationary platforms is not an optimal solution. It is more sensible to re-organize the colony to adapt to the new structure. The abandoning platforms should form a new region while the stationary platforms group into a second region. A reasonable re-organizing manoeuvre helps facilitate a harmonious correlation between the platforms, thereby balancing network load and saving processing time.

In network applications, a service is identified by the combination of an IP address and a network port. A traffic shaper controls network bandwidth by delaying traffic to meet pre-defined requirements. A traffic can be precisely allotted to a service by referring to the service's identification. By traffic shaping, two important factors for a service are determined. A minimum usable bandwidth ($MiUB$) which is the guaranteed bandwidth that the service can consume and a maximum usable bandwidth ($MxUB$) which is the bandwidth that the service can reach as long as there is free bandwidth available.

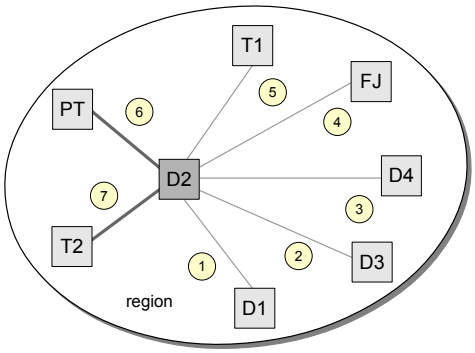

Figure 10. The network layout and bandwidth shaping imposed on ping services

The logical network layout for this test scenario is depicted in Figure 10. In the beginning, platform $D2$ is configured as RegionMaster. A scout agent is deployed to survey the region. We use traffic shaping to simulate the occurrence when the platforms, together, move far away from RegionMaster and swarm onto a new location, resulting in a decrease in the corresponding bandwidths.

In this test scenario, the traffic shaper which is comprised of software packages *tc* (traffic control), *iptables* and *iproute* in Linux distributions is used to regulate the traffic of the ping messages exchanging service between RegionMaster and the platforms $\{D1, D3, D4, FJ, T1\}$ on TCP port 3242. Table 3 lists the corresponding values $MiUB$ and $MxUB$ for every connection representing by the circled numbers depicted in Figure 10.

Con.	(1)	(2)	(3)	(4)	(5)	(6)	(7)
$MiUB$ (Mbps)	0,512	1	2	20	54	1000	1000
$MxUB$ (Mbps)	1	2	3	30	60	1000	1000

Table 3. Bandwidth allotted to each link

The bandwidth degradation will have a certain effect on the communication between agent platforms. While bandwidth shaping is being activated, there are changes in network connectivity within the region. The platform colony re-organizes to react to the changes in network bandwidth. Every platform sets the value *split* based on the ratio $\tau = \tau_{avg}/\tau_{RM}$. In this experiment the splitting threshold is set to $\tau < 50\%$. Since more than a half of the agent plaforms set $split = true$, the region is bisected. Two regions emerge from the original region. Figure 11 depicts the splitting process, those platforms with a ratio $\tau \geq 50\%$ namely $\{D2, PT, T2\}$ group into Region 1; Region 2 is made of the platforms with $\tau < 50\%$ i.e. $\{D1, D3, D4, FJ, T1\}$.

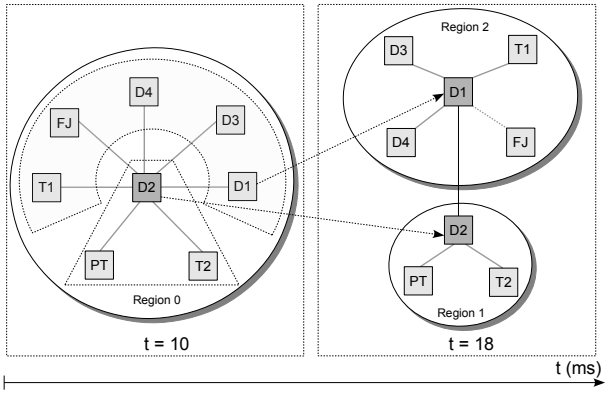

Figure 11. The self-organizing process of the platform colony

Table 4 displays the metrics measured for Region 1 and Region 2 before and after splitting. The measurement unit for the ratio τ is %. According to the metrics shown in the tables, both Region 1 and Region 2 benefit from the splitting, the ratio τ grows markedly and exceeds the threshold 50%. An increase in the ratio τ implies that the connection from each platform to RegionMaster has been improved. All the platforms reset *split* to $false$.

The metrics show that after network connectivity degrades, the splitting helps the platforms improve connection quality to RegionMaster significantly. Since

	Before splitting		After splitting	
Platform	τ(%)	**Split**	τ(%)	**Split**
Region 1				
D2	—	*false*	—	*false*
PT	75,0	*false*	75,0	*false*
T2	120	*false*	83,3	*false*
Region 2				
D1	14,8	*true*	—	*false*
D3	16,8	*true*	75,0	*false*
D4	15,7	*true*	100	*false*
FJ	38,9	*true*	87,5	*false*
T1	29,4	*true*	83,3	*false*

Table 4. Metrics measured for Region 1

RegionMaster is responsible for performing the self-organizing tasks of the whole region, an improvement in connectivity will lead to an increase in performance of the system. This test supports the hypothesis that the software framework can assist the system to react appropriately to changes happening in the surrounding environment. In our judgement, the features demonstrate that the software framework complies with one of the requirements: the ability to self-organize in a changing environment.

8.4. Context-Awareness

In the next experiment, platform *FJ* is configured as RegionMaster, a scout agent is deployed to survey the region. In this scenario, the connectivity between RegionMaster and the remaining platforms is going to be degraded using software. This aims to investigate the countermeasures of the system given that the quality of the connection to RegionMaster has declined. Since RegionMaster is responsible for the management of the whole region, the deterioration adversely affects system performance, processing speed may considerably slow down. Given the circumstances, it is expected that the software prototype helps the platform colony to re-organize and recover from the degradation.

To conduct this experiment, a simulation of connectivity degradation is required. A certain network traffic between RegionMaster and the other platforms will be created, resulting in a smaller bandwidth left to the remaining platforms. For the purpose of creating network traffic, we utilize the open source software Iperf [28]. This tool is used for measuring throughtput and performance of a network. It can also be used to produce both TCP and UDP data streams over networks; data sent by the client will be received and eventually discarded by the server.

Since Iperf consumes a certain bandwidth on the connection between RegionMaster and the rest of the region, there is smaller bandwidth left for other applications. As a result, each agent platform will experience effects from the traffic generator. The latencies between the platforms and RegionMaster grow sharply. These changes are sensed by the scout agent every time it visits the platforms. Every platform sets the value *split* based on the ratio $\tau = \tau_{avg}/\tau_{RM}$; where τ_{avg} is the average latency and τ_{RM} is the latency to RegionMaster, respectively. In this experiment the splitting threshold is set to $\tau < 50\%$.

Table 5 shows the ratio and the corresponding value *split* for every platform. At the end of a round of exploration, the scout agent goes back to RegionMaster and submits information it collected. Based on this information, RegionMaster makes a decision.

Platform	τ_{avg}/τ_{RM} (%)	Split
D1	47,8	*true*
D2	47,6	*true*
D3	37,5	*true*
D4	27,5	*true*
FJ	—	*false*
PT	37,5	*true*
T1	44,4	*true*
T2	29,7	*true*

Table 5. Metrics measured at the time of self-organizing

Since most of the platforms set the value *split* to *true* the region is splitted. The new region structure is depicted in the central part of Figure 12. There is only one platform staying at the old region, this is RegionMaster. The remaining nodes join the new region with *D2* promoted to be the new RegionMaster. In this case, an adaptation has been made, the old RegionMaster relinquishes its leadership in the only-one platform region and becomes an inferior node of the new region. Eventually, a new region emerges from the original region.

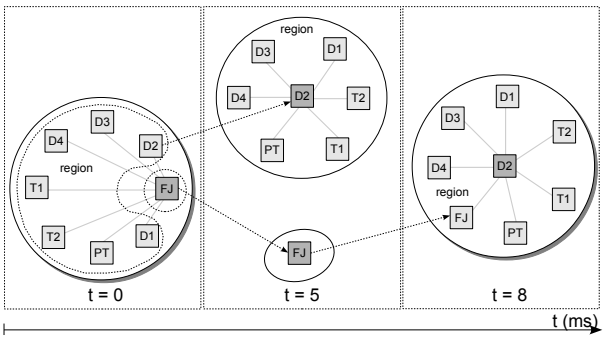

Figure 12. Exchange in role of the command node

Figure 13 displays the latency to RegionMaster τ_{RM} of every platform in three phases. For a platform, the left column is the latency before `Iperf` is activated;

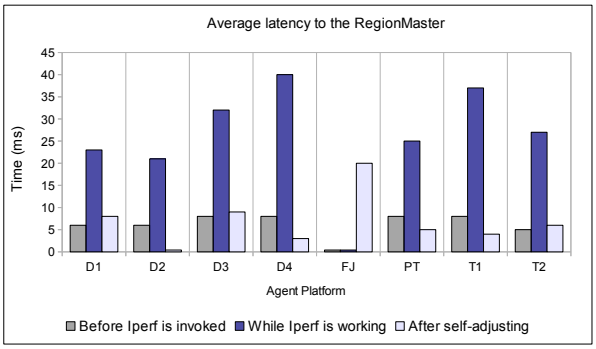

Figure 13. Average latencies to RegionMaster

the middle column represents the time while Iperf is operating and the right column is the latency of the platform after the self-adapting process has occurred. In the beginning and while Iperf was working FJ was RegionMaster so $\tau_{RM}(FJ) = 0$. Similarly, after $D2$ has taken its job as RegionMaster $\tau_{RM}(D2) = 0$. As usual expected, while Iperf is being executed, the latencies to RegionMaster measured for every platform increase significantly. However, once $D2$ takes over as RegionMaster, the latencies decrease proportionally.

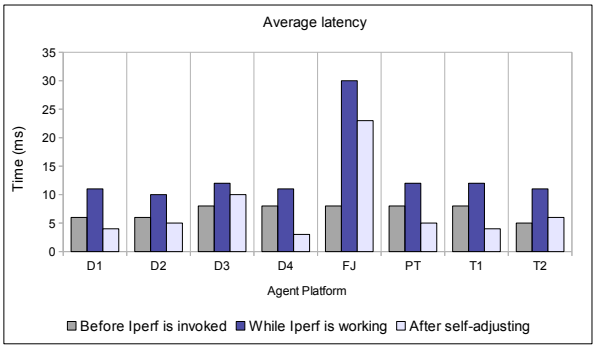

Figure 14. Avarage latencies of all platforms

Figure 14 shows the average latencies τ_{avg} of every platform in three phases: Before Iperf is executed, while Iperf is being executed and after the adaptation has been made. These latencies also shift in the same pattern as by τ_{RM}. Before additional bandwidth was produced, the average latencies had been at a normal level. While Iperf was operating the latencies rose markedly. After the region has been restructured, these values resume to a normal level. It can be seen that the network monitoring activities supply the framework with up-to-date information about network situation, thereby facilitating the decision making process. The measurement results suggest that the swap in role of RegionMaster from FJ to $D2$ brings a more stable connectivity to every platform compared to that of the old arrangement, right after Iperf started producing bandwidth.

8.5. Fault Tolerance

We decided to export the test program to run on simulation environment, since a more precise evaluation result can be obtained if performance tests are performed with presence of several network nodes. The software Network Simulator NS2 is a discrete event simulation framework. It was written in the programming languages C++ and Python and has been used widely for simulating applications running in wired and wireless networks. NS2 supports various routing protocols, including TCP, UDP, multicast. NS2 is normally utilized for testing distributed network applications on a laboratory scale. Tcl (Tool Command Language) is used as the language for scripting simulation scenarios in NS2 [30].

AgentJ is a software tool for embedding real applications written in Java in the NS2 simulation environment. AgentJ allows Java source code to execute on NS-2 with minor modification. Essentially, AgentJ works as a bridge between Java source code and NS2 [32]. The use of AgentJ is practical given that there is a need for simulating program written in Java with solicitation of a large number of nodes. The combination NS2 and AgentJ provides us with a convenient way to simulate evaluation tests without needing to setup a real network with multiple computers.

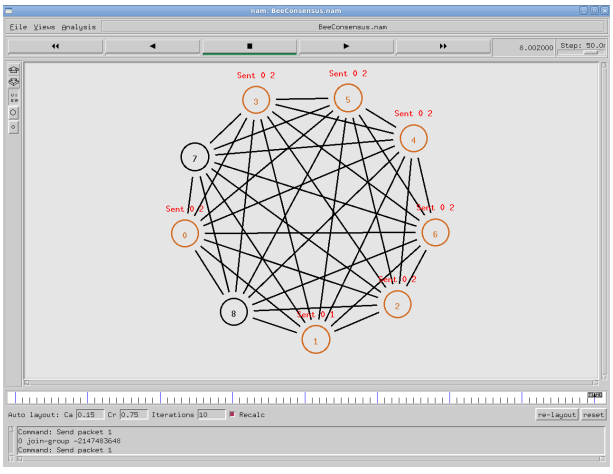

Figure 15. NS2 Simulation with a handful of network nodes

Network Animator (NAM) is a tool embedded in NS2 to visualize simulation. Figure 15 depicts an example of multicast network nodes in NAM, a circle represents a network node and a line between one pair of nodes corresponds to a link between them. In a multicast scenario, every node is connected to all other nodes. For a clear representation, in the figure we depict

only a network with a handful of nodes, however in simulation, we can increase the number of network nodes to meet our requirement.

We setup NS2 and AgentJ in the Linux Distribution Fedora 12. Network crash is simulated by shutting down some nodes randomly during execution given that at most $f < n/2$ nodes may fail. A node fails with a probability of a randomized value ranging from 0 to 1.

In the evaluation, the number of nodes is set to the following values $n = 5, 10, 15, 20, 25, 30$ and the number of failed nodes $f \in \{0, \lfloor n/2 \rfloor\}$. We ran the tests with different sets of input values and with repetition. This aims to cover a wide range of possibilities of execution.

For the implementation, Function $Evaluate(0, 1)$ in Line 15 is determined as $Evaluate(0, 1) = abs(C_p(k, 0) - C_p(k, 1))$. Once a node starts to follow a value v, it assigns the difference to the strength value $strength = Evaluate(0, 1)$. Every time it is called, Procedure $Decrease(strength)$ subtracts 1 from $strength$ until it is smaller than 0.

The outcomes of the execution are shown in Table 6 and Table 7. For each table, the first row represents the number of nodes taking part in building consensus n. The second row is the number of nodes that fail during execution f. The third and fourth rows r_1 and r_2 are the corresponding numbers of rounds that Ben-Or's algorithm and the honey bee inspired version reach a consensus, respectively.

n	5			10			15		
f	0	1	2	0	3	4	4	6	7
r_1	1	1	12	1	10	20	4	23	60
r_2	1	1	2	1	2	2	2	2	2

Table 6. Performance Comparison for $n = 5, 10, 15$

n	20			25			30		
f	6	8	9	8	10	12	10	12	14
r_1	17	32	189	32	291	1319	112	1531	7554
r_2	2	2	2	2	3	2	4	2	2

Table 7. Performance Comparison for $n = 20, 25, 30$

At first sight, it can be seen that, the larger the number of nodes is, the more difficult a consensus can be reached. If no node fails, that means $f = 0$ then both approaches can gain a swift consensus. When failure is present, there are differences in performance. By the original Ben-Or's algorithm, if the number of failed nodes increases, the number of rounds that it gains a consensus increases correspondingly. Especially, when f is nearly approaching $n/2$ a large number of rounds can be seen.

Both tables show that the honey bee inspired algorithm has a considerably better computational performance, especially when network nodes fail en masse. Compared to Ben-Or's algorithm, it can gain a consensus after a small number of rounds of exchanging messages. In addition, its performance is stable towards the number of failed nodes. We witness an improvement in performance when applying the honey inspired mechanism.

9. Conclusion

In this paper, we have introduced our proposed architecture for mobile agent communication in highly dynamic networks. Our research was motivated by a mobile agent system working in networks for rescue forces in a mass casualty incident. In this type of networks, connection instability militates against successful transmission of agent messages. To facilitate message transmission, it is necessary to organize the logical connection among agent platforms in an operationally feasible manner. Our work aims to develop a communication model for mobile agent systems that is able to deal with the inherent dynamics of modern networks.

We found inspiration from the honey bee colony where honey bees have a special mechanism to reach self-organization. The organization of the whole colony is done through the interaction among thousands of bees. The behaviours of honey bees inspire us to employ the self-organizing model on the colony of agent platforms. We proposed a solution to a network management mechanism where network organizing tasks are performed in an autonomous manner.

Mobile agents are sent over the network to gather network and platform information. Information collected by mobile agents gives a base for managing the network. Each agent platform plays a contributory role in organizing the region. A final decision to reorganize the network is made based on a consensus of the platforms. In a platform colony, though the master node plays an important role, it is replaceable. Thanks to a failure detector, whenever it fails a second platform can substitute for it and system's operation is not interrupted. Based on the existing mobile agent system Ellipsis, the software prototype for an adaptive model for mobile agent communication has been designed and implemented.

The software prototype was deployed to run in a system of a laboratory scale where conditions of a real dynamic network had been simulated using various software tools. The experiments demonstrated that the software prototype provides the system with the ability of being self-adaptive. It helps the system to react adequately to environmental stimuli as well as to take suitable measures to counteract unexpected

network degradation. In addition, given that network failure happened, the system can achieve fault tolerance and maintain a reasonable processing cost for message coding and transferring. From our perspective, the software framework fulfils the basic requirements.

It is our firm belief that the honey bee inspired model cannot only be utilized for mobile agent systems. It can also be applied in other types of P2P network, where it is necessary to network member nodes in a flexible manner. The model can be used to solve other types of problems in network management, such as for discovering resources in P2P networks [29].

References

[1] Seeley, T.D. and Kirk Visscher, P. (2003) *Choosing a home: How the scouts in a honey bee swarm perceive the completion of their group decision making* Journal of Behavioral Ecology and Sociobiology, vol. 54, pp. 511–520

[2] Seeley, T.D. and Kirk Visscher, P. (2004) *Group decision making in nest-site selection by honey bees* Journal of Apidologie, vol. 35, pp. 101–116

[3] Nakrani, S. and Craig, T. (2004) *On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers* Journal Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems , pp. 223–240

[4] Karaboga, D. and Bahriye, A. (2009) *A survey: algorithms simulating bee swarm intelligence* Journal Artif. Intell. Rev., pp. 61–85

[5] Pham, T. and Afify, A. and Koc, E. (2007) *Manufacturing Cell Information using the Bees Algorithm* In Proceedings Innovative Production Machines and Systems Virtual Conference

[6] Nguyen P.T. and Schau V. and Rossak W.R. (2011) *Towards an Adaptive Communication Model for Mobile Agents in Highly Dynamic Networks based on Swarming Behaviour*. The 9th European Workshop on Multi-agent Systems (EUMAS)

[7] Nguyen P.T. (2013) *Building Consensus in Context-Aware Systems Using Ben-Or's Algorithm: Some Proposals for Improving the Convergence Speed*. In Proceedings of the Second International Conference on Context-Aware Systems and Applications, ICCASA 2013, pp. 87–96

[8] Nguyen P.T. and Schau V. and Rossak W.R. (2013) *A Context-Aware Model for the Management of Agent Platforms in Dynamic Networks*. In Proceedings of the Second International Conference on Context-Aware Systems and Applications, ICCASA 2013, pp. 77–86

[9] Finin, T. and Labrou, Y. and Peng, Y. (1998) *Mobile Agents can Benefit from Standards Efforts in Inter-agent Communication* IEEE Communications Magazine, vol. 36, pp. 50-56

[10] Braun, P. (2003) *The Migration Process of Mobile Agents* PhD Dissertation, Friedrich Schiller University Jena.

[11] Baumann, J. (2000) *Control algorithms for mobile agents* PhD Dissertation, the University of Stuttgart.

[12] FSU Jena (2011) *The SpeedUp Project (2011)* http://www.speedup-projekt.de

[13] Baumann, J. and Hohl, F. and Radouniklis, N. and Rothermel, K. and Strasser, M. (1997) *Communication Concepts for Mobile Agent Systems* In Proceedings of the First International Workshop on Mobile Agents

[14] Fischer, M.J. and Lynch, N.A. and Paterson, M.S. (1983) *Impossibility of Distributed Consensus with One Faulty Process* Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems, pp. 1–7

[15] Heylighen, F. (1999) *The Science Of Self-Organization And Adaptivity* The Encyclopedia of Life Support Systems, vol. 62, pp. 253–280

[16] Seeley, T.D. and Buhrman, S.C (2001) *Nest-site selection in honey bees: how well do swarms implement the best-of-N decision rule?* Journal of Behavioral Ecology and Sociobiology, vol. 49, pp. 416–427

[17] Yeom, K. (2010) *Bio-inspired self-organization for supporting dynamic reconfiguration of modular agents* Journal of Mathematical and Computer Modelling, vol. 52, pp. 2097–2117

[18] Heylighen, F. and Gershenson, C. (2003) *The meaning of self-organization in computing* IEEE Intelligent Systems.

[19] FIPA (2002) *ACL Message Representation in String Specification*

[20] FIPA (2002) *ACL Message Representation in Bit-Efficient Specification*

[21] Ben-Or, M. (1983) *Another Advantage of Free Choice (Extended Abstract): Completely asynchronous agreement protocols* Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing

[22] Vavala, B. and Neves, N and Moniz, H. and Veríssimo, P. (2010) *Randomized Consensus in Wireless Environments: A Case Where More is Better* Proceedings of the 2010 Third International Conference on Dependability, pp. 7–12

[23] Aspnes, J. (2003) *Randomized Protocols for Asynchronous Consensus* Journal of Distributed Computing, vol. 16, pp. 165–175

[24] Aspnes J. (2002) *Fast deterministic consensus in a noisy environment* Journal of Algorithms, vol. 45, no. 1, pp. 16–39

[25] Thom, C., Gilley, D.C., Hooper, D., Esch H.E.: The Scent of the Waggle Dance. PLoS Biology (2007)

[26] Hayes, J.: Pleasure chemical controls bee dance. Cosmos Online (2007)

[27] Gadagkar,R.: The Honeybee Dance-Language Controversy. Resonance: Journal of Science Education, vol. 1, pp. 63–70 (1996)

[28] NLANR/DAST: Network Performance Measurement. http://sourceforge.net/projects/iperf/, (2012)

[29] Mittelbach F. and Goossens M (2004) *The LaTeX Companion* (Addison-Wesley), 2nd ed.

[30] The Network Simulator NS2, http://www.cs.virginia.edu/ cs757/slidespdf/cs757-ns2-tutorial1.pdf

[31] Aguilera, M.K., Toueg,S.: The correctness proof of Ben-OrŠs randomised consensus algorithm (2011)

[32] Taylor, I., Downard, I., Adamson, B., and Macker, J. AgentJ: Enabling java NS-2 simulations for large scale distributed multimedia applications. In Second International Conference on Distributed Frameworks for Multimedia DFMA 2006, Penang, Malaysia, pp.1-7, (2006)