Location update schemes for mobile agents

RAMA SUSHIL[1]
RAMA BHARGAVA[2]
KUMKUM GARG[3]

[1]Asst. Prof. SGRRITS D. Dun, R/S Dept. of Maths, I. I. T. Roorkee, India, ramasushil@yahoo.co.in
[2]Prof. of Mathematics, I. I. T. Roorkee, India, bhargava_iitr@rediffmail.com
[3]Prof. of Computing, I. I. T. Roorkee, India, Sr. Member IEEE, kgargfec@iiternet.in

Abstract. For mobile agents roaming in a network, location management strategies still represent a current research issue, for contacting and communicating with these agents. The cost of a location management strategy mainly depends on the cost of location search and update. In order to save this cost, mobile agents should not update their location every time they change the host. This paper discusses three strategies in which mobile agents make decisions about when and where to update their location: the time-based strategy, the movements-based strategy and the strategy based on counting the migrations host wise and birth-region wise. We propose the birth-region wise movement based location update strategy, named Broadcasting with Search by Path Chase (BSPC), which is an extension of SPC (Search by Path Chase). We have analyzed and compared the performance of both SPC and BSPC and found that BSPC costs less than SPC for low call to mobility ratio (CMR).

Keywords: Location management strategy, Mobile agent, mobile agent system, mobile agent host, cost of location update, birth-region.

(Received August 27, 2007 / Accepted December 05, 2007)

1 Introduction

Mobile agent technology is one of the most vibrant and active areas of research and development in information technology. It is making significant impact upon almost all aspects of the computing discipline. A mobile agent is a running program that can move from host to host in a network when and where it chooses. A mobile agent is an extension of mobile code which itself is an extension mobile object, in which an object (code and data) moves from one host to another. The mobile agent abstraction extends this notion further by moving code, data, and a thread from one host to another [9, 14, 23]. Mobile agents are used for network management, e-commerce, information retrieval etc, and a university might let mobile agents circulate to allow large scientific calculations to run during idle moments on university owned workstations.

A mobile agent system is a platform that can create, interpret, execute, transfer and terminate mobile agents [17, 1, 10, 12, 24, 2, 22, 27]. A node in a network with mobile agents is called a mobile agent host. A mobile agent migrates from one node to another while performing a task autonomously on behalf of a user. When a mobile agent migrates, it chooses the next destination according to an itinerary, which is a predefined travel plan, or dynamically according to execution results.

Because of the extremely dynamic nature of mobile agents, mobile agent systems not only feature mobility, but also a technique to discover their position at any time. This is called the mobile agent location management strategy. It is needed in order to have control of the agent by its owner and also for agent-agent communication. The ability to locate mobile agents while they are migrating from one node of the network to another one is of great importance for the development of agent-based applications, in geographically distributed environment.

The major issue with location management in mobile agent computing is the high cost associated with location update and search. The goal of an efficient location management strategy should be to provide low cost of location search and updates [16, 18]. The cost of a location update and search is characterized by the time taken for each operation, number of messages sent, size of messages, or the distance the messages need to travel. An efficient location management strategy should attempt to minimize the combined cost of the location search and update.

At one extreme, up-to-date information of the exact location of all mobile agents is maintained at each and every node in the network. This reduces the search time to locate the mobile agent. But each time the location of the mobile agent changes, a large number of associated location databases must be updated, which involves cost of location update. At the other extreme, no information is stored at any site of the network. To locate a mobile agent, a global search at all network sites must be initiated; however, when the mobile agent moves, no cost is associated with updating the location database. Between these two extremes, various approaches that balance the cost of search against the cost of updates are possible. In this paper, we concentrate on location update schemes used in existing location management strategies [4, 7, 19, 28]. We propose a location management technique which is based on search-by-path-chase technique [29](SPC) and named Broadcasting with Search-by-Path-Chase (BSPC). In BSPC, we propose a location update scheme, which costs less in applications having a low frequency of queries for contacting mobile agents i.e. for low CMR. This is discussed in detail in section 4.1.2.

The rest of this paper is organized as follows: Section 2 introduces the time-based location update model. Section 3 includes a discussion of movements-based location update strategy (host wise). Broadcasting with search by path chase protocol, which we propose, is described in section 4. In section 5 there is a comparative discussion of all the three basic location update models. The paper concludes with a report on further work.

2 Time Based Location Update Model

The path concept provides functionality for locating agents and for their termination. The idea of paths is a well-known technique in the area of distributed systems [30] used for locating distributed objects. A path is a distributed structure pointing towards the object in question, and has to be followed to reach the object. The single elements of the path, i.e. the data structures on a node, are called proxies.

An agent moves through a mobile agent system in an unforeseeable manner, i.e. normally no predictions can be made about its location at a certain time. But if every agent leaves information about its new location on its old place when it migrates, i.e. leaves a proxy, then a path of proxies is created. This path of proxies can be followed if the place where the agent has been created, called the anchor place, is known. This path ultimately leads to the agent place, i.e. the place at which the agent resides currently. The main problem with this approach is the house keeping, i.e. how the proxies can be removed when the path is no longer valid, and the dependence on the availability of a high number of participants, the proxy nodes.

The path can be created by creating a proxy pointing to the destination place of an agent when it leaves. Please refer to [15] for details. A variant of the path concept that allows a decrease in the number of path proxies, the time based location update model, is discussed in [5, 6]. In this model, information about the location of the agent is sent to the anchor place. Setting the target of the proxy at the anchor place to the new place can shorten the path and this is done after a certain period of time. The MASIF proposal and the Aglets system propose a similar mechanism to find agents [17].

3 Movement Based Location Update Model (host-wise)

We now model a single agent visiting several hosts. At some time an agent A1 visits Host1 and resides on it for time T1 [15, 21]. Then it leaves Host1 and migrates to Host2 within t1. It stays on Host2 for T2 and goes on. We see that the mobile agent resides on a host it is visiting for a generally distributed time interval and then moves on to the next host. Notice that Ti is normally far bigger compared to ti (i.e. the transmission time can be ignored in a high speed LAN). Ti is the Host Residence Time (HRT) of the agent. For an agent performing some fixed tasks, the HRTs on different hosts may have similar values, but vary within some thresholds, due to network transmission and host execution delays. The probability density function of the HRT is denoted by $f_m(T)$which has Laplace-Stieltjes transform $f_m(s)$ and mean $\frac{1}{\lambda_m}$ [3, 8, 20].

In this model, a roaming agent updates its current location using the movement based location update scheme. In this scheme, an agent updates its location after d movements i.e. after crossing over d hosts, since its last update. If d is assigned dynamically, the scheme is called dynamic location update scheme. Each mobile agent is simulated to follow a Poisson process with rate λ_c since this is a stochastic process. Call to Mo-

bility Ratio is given by $\theta = \frac{\lambda_c}{\lambda_m}$ where λ_c is the rate of Poisson process describing the incoming call arrivals and $\frac{1}{\lambda_m}$ is the mean of probability density function of HRT. The average number of location update costs per request (C_u) is small for a large d. Derivation of C_u is shown in [12, 27].

4 Movement Based Location Update Model (birth-region wise)

The model we propose is based on some assumptions made on the distributed reference environment [26]. A complete computing environment is considered as the collection of regions. A region is a collection of mobile agent hosts. Different hosts can spawn different mobile agents. For a mobile agent spawned by a host, the region to which the host belongs is called the birth-region of that agent.

We assume that in each region a site acting as Agent Name Server (ANS) exists, which manages a database called Region Agent Register (RAR), which stores information about all the agents that have been created in the region or have transited through it. Each entry of a RAR is in the form of (m, λ) where the field λ represents the location information related to the agent whose name is m (the primary key of the database). λ is a GLI or a region name (GLI.region), which means that the agent can be found on that location-or region-or it has transited from that location or region. The ANS has to export suitable services to allow remote access to the RAR (tuple insertion, modification, query, or deletion), but each service has to be executed atomically.

To find the ANS of a given region, we assume the presence of a suitable network protocol, such as the Internet DNS protocol which allows determination of the name of the host acting as mail exchanger from a domain name [13, 25]. In the following, we will refer to the RAR of a region named r with the notation RARr, the RAR of the region of ownership of location λ will be indicated with $RAR_{\lambda.region}$.

We also assume that, on each location, there is a Site Agent Register (SAR), which contains information about all the agents that have transited through (in the past) or that are at that location. Each entry of SAR is a tuple (m, α, λ) where λ represents the location information related to the agent α whose name is m. In a possible implementation, the symbol α plays the role of agent identifier, and can have the form of, for example, a numeric id, a reference, or a pointer to the object, which encapsulates the agent, etc. Also, for the SAR, the primary key is the agent name m. Here, λ is a GLI, a region name (GLI.region), or the nil value.

The former case means that the agent can be found at that location (or region) or it has transited through that location (or region). The latter case means that the agent is at the same location as the SAR. If the agent does not reside on the location of the SAR (λ is not nil), the α attribute is equal to nil. The right side of Table 1 summarizes the cases. No assumption is made about concurrency in accessing a SAR, but we assume the use of an exclusive lock on each entry of the register. This capability will be used not only to control concurrent access to the SAR, but also to prevent some inconsistencies, which may happen during concurrent interaction and migration processes. In the following, the SAR of a location λ, will be indicated by SAR_λ.

For fault handling, we assume that both the registers are stored on a stable storage and that, after a site crash, suitable management routines e.g., (the same used in any DBMS) are able to restore the registers content. The stability of SAR_s is required only if the mobile agent framework supports the storage and check pointing of agents, allowing restart of computations after a site crash. Otherwise, the stability of SAR_s is meaningless. In addition, if a site crash does not happen, we assume that accessing a SAR or RAR may fail only if a network error occurs. Finally, we assume that when a process holding a lock on the SAR crashes, the lock is automatically released by the operating system (as showed in Table 1).

Table 1: RAR/SAR tuples

RAR Tuple	Meaning	SAR Tuple	Meaning
(m, GLI)	The agent is at location GLI or has traveled through it.	(m, nil, GLI)	The agent is at location GLI or has travelled through it.
(m,GLI. region)	The agent is at region GLI or has traveled through it.	(m,nil, GLI. region	The agent is at region GLI.region
		(m,, nil)	The agent is in the same location as the SAR

4.1 Broadcast with Search by Path Chase Protocol (BSPC)

The protocol we propose is called Broadcast-Search-by-Path-Chase (BSPC); its functioning is based on an efficient algorithm which follows a part of the links the agent has left on the registers of the visited sites or regions and broadcasting to search if an agent is in its

birth region. Given the hypotheses made in the previous section, we can assert that, for each location reached at time tl by an agent named m, querying SAR at time tq (with tq tl) will return a tuple containing either the agent (if it has not yet changed location) or the name of the location (or of the region) reached by the agent at its next migration (after tl). A similar assertion can be made for RAR, with respect to each region traversed by the agent. This means that, at time tq, starting from any site or region visited by the agent before tq, following the links left in the registers implies reaching a subset of the locations and regions of the itinerary followed by the agent before tq, until the agent itself is caught.

Given this, since locating an agent could require following a long path before reaching it, the update operations performed during the migration phase are designed in order to shorten this path, thus increasing interaction efficiency and reducing overhead.Locating an agent entails the following steps: First, the name of the agent's region of birth is extracted from its name m, then the relative ANS is contacted; the latter's register will contain an indication of the location the agent could be on or the name of its current region- if it is different from that of birth. In the latter case, the ANS of the new region is contacted and the search is repeated. In the former case, the SAR of the location is queried: If the resulting tuple contains a non-zero value for , then the agent is found, otherwise is used to restart the search. In particular, if refers to a GLI, the relevant SAR is contacted, while, if refers to a region, the relevant RAR is contacted. It is up to the binding and migration phase to maintain consistency of location information in the registers in such a way as to always allow agent finding (unless a system or network crash occurs).

During location finding, each time a SAR is queried for an agent m, an exclusive lock is set on the relevant tuple and reset only if the agent does not reside in that place (i.e. the value of is nil), otherwise the lock is maintained. As we will see in the next section, keeping the SAR locked prevents the agent from migration, allowing performing interaction, after finding it.

4.1.1 Register Update in the BSPC Protocol

Performance and reliability of the location-finding phase of the BSPC protocol, strongly depends on the register update operations made during binding and migration. To avoid the burden of agent migration, the protocol aims to minimize interregional messages as compared to intraregional ones. This can reduce overhead and improve efficiency if we assume that connections between sites in the same region are faster and more reliable than connections between different regions. This hypothesis is not restrictive for a distributed environment and reflects a common practice in WANs like Internet: Sites belonging to the same subnetwork are often connected by LAN (ten to hundred MB per sec), while connections between sub networks are point-to-point links working at a lower speed sixty four Kb/s to two Mb per sec.

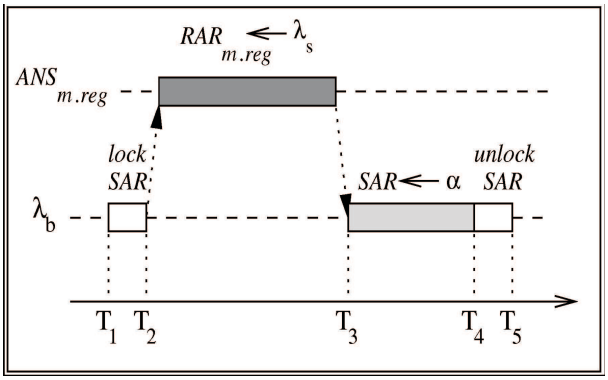

Figure 1: Operations performed during agent name binding

The binding phase (figure 1), occurring when agent α is spawned, entails the registration of the agent's name m and the birth location λ_m of α in $RAR_{m:region}$ (m.region is the region of birth). This is handled by a two-step protocol performed by the platform executing at location λ_s. First, $RAR_{m:region}$ is contacted and, here the tuple (m, λ_s) is registered. Then, the tuple (m,α, nil) is stored in SAR_s. Before starting the update operations, an exclusive lock is placed on the entry of the SAR_{λ_s} relevant to m; it is released when registration is over or an error occurs.

The migration phase involves updating the location information of the migrating agent. Given λ_s and λ_d, the source and destination locations, the sequence of operations can be split into two steps, performed in λ_s and λ_d respectively, before and after agent transfer. These steps vary according to whether λ_s and λ_d belong to the same region or not. In the case of intraregional migration $\lambda_d.region$ =$\lambda_s.region$ and it is the birth region then the aim is to not to update the entries relevant to the migrating agent in both SAR_{λ_s} and $RAR_{\lambda_{d.region}}$. But the RAR of the birth region is updated only when the agent crosses the region. If λ_s and λ_d belong to two different regions (interregional migration), the migration protocol has to update SAR?d, RARm:region. if $\lambda_d.region \neq m.region$, by writing $\lambda_d.region$ as the location; it also updates by writing $RAR_{\lambda_{d.region}}$ as the location and, finally SAR_{λ_s}, to register the presence of the agent.

This allows the location finding protocol, which starts from the agent's region of birth, to reach the its current region and, finally its current location. At location λ_d first the entry of the SAR is locked, then, after agent transfer, the tuple (m, nil, $\lambda_{d.region}$) is stored on the SAR, then the tuple(m, $\lambda_{d.region}$) is stored on $RAR_{\lambda_{d.region}}$ and finally, the lock is released. At the destination location λ_d when the agent transfer begins, a lock is placed on the entry of the SAR and the $RAR_{\lambda_{d.region}}$ is updated with λ_d as location. When migration ends, first SAR_{λ_d} is updated by storing the tuple (m,α, nil) then the lock in the SAR is released, and finally, agent execution is resumed. At this time, if $\lambda_d.region \neq m.region$, a background (concurrent) process is started in λ_d to remotely update $RAR_{m.region}$ by writing a tuple with $\lambda_{d.region}$ as location.

In the proposed protocol, SAR locks play a fundamental role: They not only ensure exclusive access to SAR, but also help to resolve several inconsistencies which may happen between migration and interaction. In fact, an interaction request may arrive when the agent is migrating; i.e. it is neither at location λ_s nor at location λ_d but on the net, and cannot be contacted at all. To take care of such a situation, a simple solution adopted by some existing frameworks entails the generation of an error condition, forcing the interacting agent to retry in the future. In the authors opinion, a better solution is to wait for migration to complete and then contact the agent at the destination location. In our protocol, this is automatically performed exploiting SAR locks.

4.1.2 Message complexity calculation

In the BSPC protocol, for intraregional migration queries, (when mobile agent is in its birth region only), one message comes first to the birth region of the queried agent, and suppose there are a maximum of m hosts in a region and a minimum 1, then m messages will be broadcast to all hosts and only one massage will be sent back from the host where the agent is residing. Let us suppose nh is the number of hosts in a region. Then Message complexity is:
$n_h + 2$ Where $n_h = m$ irrespective of the number of hops of the mobile agent. The message complexity for the SPC protocol is: $n_h + 2$ where $1 \leq n_h \leq m$, and depends on the number of mobile agent hops. Figure 2 shows the simulation results: message complexity in BSPC is equal to the maximum possible limit in SPC but not more. Even with more message complexity, BSPC has several advantages over SPC discussed in detail in section 4.1.3

For interregional migration, message complexity for SPC and BSPC is the same and is calculated as follows: Let the agent migrate from its birth region through at the most n regions. Assume that there are m sites in each region. Of the n regions, one is the birth region. So

$$1 \leq n_r \leq n$$

If the mobile agent crossed $n_r$regions and in the final destination region, it goes to only one site, one message is required to reach from RAR to SAR. Now if it is on the same SAR, the agent is found by using $n_r + 2$. If there are at most m migrations within that region, then message complexity is: $n_r + 1 + n_h$ where

$$1 \leq n_h \leq m$$

and n_h is the number of migrations within a destination region.

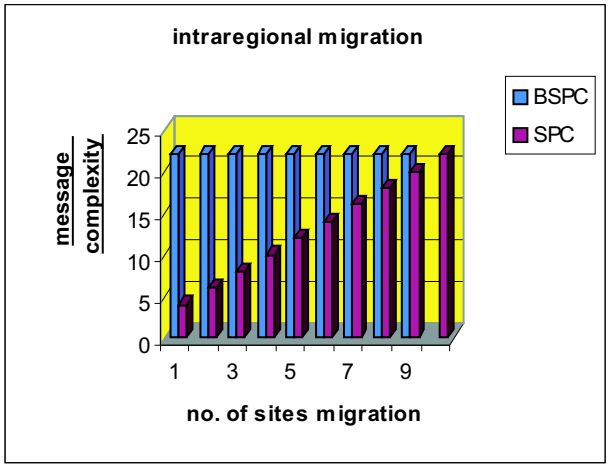

Figure 2: Number of sites migration in birth region vs. message complexity for 10 hosts in a region

4.1.3 Advantage of BSPC over SPC

1. In SPC there is an update operation at each migration whether the agent is in its birth region or any other, while in BSPC, these update operations get canceled for an agent which is in its birth region, so the cost of update is reduced. Figure 3 shows the same for 10 hosts in a region. Assuming that the cost of location update is proportional to the number of update operations, Figure 3 shows that cost of update is zero as no update operation is performed for migrations within the birth region. This improves the speed of processing of the agent, as no time is wasted in leaving proxies or contacting the home location.

2. BSPC is very effective for very large networks, with large number of regions and where each region does not have a large number of hosts, as broadcasting is expensive for a large area (in our protocol it is limited by broadcasting in birth region only).

3. BSPC gives best possible results when there is less frequency of queries for an agent, which has less number of interregional migrations and high frequency of intraregional migrations (in birth region).

4. Locating agents is comparatively faster in BSPC as following a long path is not needed.

5. BSPC uses memory more efficiently by reduction in saving of proxies at host machines.

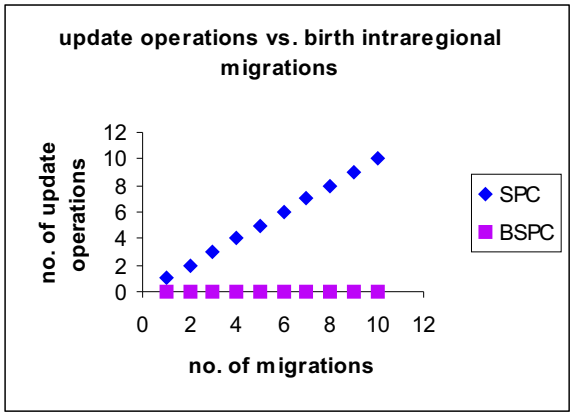

Figure 3: Number of update operations Vs. no. of intraregional migrations (birth region)

5 Comparison and Evaluation

The path concept allows agents to roam the network without contacting their home node, as location is not updated at the home node (the node at which the agent is spawned) but proxies are left at the source nodes. The disadvantage results from the path not being limited in length. Here message complexity is proportional to the length of the path. If the path is extended with time to live (ttl) [5], the location update is done at the anchor node after time ttl.

Movement based location update - host wise [21] allows agents to roam the network without contacting the home location and comparatively reducing the location update operation which takes place not at every hop but after some optimal threshold migrations. Here message complexity is proportional to the number of hops.

SPC allows the agents to roam the network without contacting its home node in a region. Contact with the home node of the birth region and the Agent Name Server of the current region are needed when the mobile agent crosses regions. Here message complexity is proportional to the number of migrations.

BSPC allows agents to roam more freely in their birth region without contacting the home node and without leaving the proxy at source node in the birth region. Here message complexity is equal to the number of nodes in the birth region if agent roams in its birth region. If not, message complexity is proportional to the number of migrations.

6 Conclusion and further work

To come back to the issue of when a mobile agent should update its location, we find that although the aim remains is to minimize the location update cost, a balance is to be made between the update cost and search cost. So the mobile agent can update its location at any one or more of the following times:

1. At every hop.

2. After d (optimal number) hops.

3. After d hops, where d is decided dynamically.

4. After a certain period of time.

5. After crossing its birth region.

Instead of broadcasting, multicasting [11] can be used by the ANS (Agent Name Server) for mobile agents generated by it, wherever they may be either in their birth or in any other region. We intend to try this approach next.

References

[1] Grasshopper home page `http://www.grasshopper.de`, 2002.

[2] Object Space Voyager, `http://www.objectspace.com`, 2002.

[3] Akyildix, I. F., Ho, J., and Lin, Y. Movement based location update and selective paging for pcs networks. *IEEE/ACM Trans. Networking*, 4(4):629–638, August 1996.

[4] Awerbuch, B. and Peleg, D. Concurrent online tracking of mobile users. *ACM*, 21(4):221–233, October 1991.

[5] Baumann, J. A comparision of mechanisms for locating mobile agents. TR 1999/11, Univ. of Stuttgart, Faculty of Computer Science, 1999.

[6] Baumann, J. *Control alorithms for mobile agents*. PhD thesis, IPVR, Stuttgart, 1999.

[7] Bhattacharya, A. and Das, S. K. Lezi update: An information theoritic approach to track mobile users in pcs networks. In *proc. of MOBICOM'99*, 17-19August 1999.

[8] Feller, W. *An Introduction to Probability Theory and its Applications*. New York: Wiley, 1966 edition.

[9] Fuggetta, A., Picco, G. P., and Vigna. Understanding code mobility. *IEEE Trans. Software Eng.*, 24(5):342–361, May 1998.

[10] Gray, R. S. Agent tcl: A transportable agent system. In *Proceedings CIKM Workshop Intelligent Information Agents*, December 1995.

[11] Hartroth, J. and Hofmann, M. Using ip multicast to improve communication in large scale mobile agent systems. In *Proceedings of the Thirty First Hawaii International Conference*, volume 7, pages 64 –73, 1998.

[12] Karnik, N. M. and Tripathi, A. R. Design issues in mobile-agent programming systems. *IEEE Concurrency*, 6(3):52–61, July-Sept 1998.

[13] Kotzanikolaou, P. and et al. Secure transactions with mobile agents in hostile environments. *LNCS, ACISP*, 4(1841):289–297, june 2000.

[14] Lange, D. and Oshima, M. *Programming Mobile Agents in Java with the Java Aglet API*. Addison Wesley, 1998 edition.

[15] Li, T. and Lam, K.-Y. An optimal location update and searching algorithm for tracking mobile agent. In *proc. of AAMAS02*, 15-19July 2002.

[16] Lin, Y. B. Reducing location update cost in a pcs network. *IEEE/ACM Trans. Networking*, 5(1):25–33, Feb. 1997.

[17] Milojacic, D., Breugst, M., Bussee, I., Campbell, J., Covaci, S., Friedman, B., Kosaca, K., Lange, D., Ono, K., Oshima, M., Tham, C., Sankar, V., and White, J. Mobile agent system interoperability facilities specification. TC Document orbos/97-10-05, OMG, 1999.

[18] Noy, A. B., Kessler, I., and Sidi, M. Mobile users: To update or not to update? *ACM- baltzer J. Wireless Networks*, 1(2):175–186, July 1995.

[19] Pitoura, E. and Samaras, G. Locating objects in mobile computing. *IEEE Trans. Knowledge and Data Eng.*, 13(4), July/Aug 2001.

[20] Ross, S. M. *Stochastic Processes*. New York: Wiley, 1993 edition.

[21] Roth, V. and Peters, J. A scalable and secure global tracking service for mobile agents. In *LNCS, MA*. Springer Berlin/Heidelberg, April 2001.

[22] Santoro. Arca: A framework for mobile agent programming white paper. Technical Report 4, Univ. of Catania, 1998.

[23] Silva, A., romao, A., Deugo, D., and da Silva, M. Towards a reference model for surveying mpbile agent systems. *Autonomous agents and multiagent systems*, 4(3):187–231, September 2001.

[24] S.Milojicic, D., LaForge, W., and Chauhan, D. Mobile objects and agents (moa). *Distributed System Eng.*, 5(4):214–227, Dec. 1998.

[25] Steen, F. J. V., Homburg, H. P., and Tanenbaum, A. S. Locating objects in wide-area systems. *IEEE Communication Magazine*, 5(4):104–109, Jan. 1998.

[26] Steen, M. V., Homburg, P., and Tanenbaum, A. S. Globe: A wide-area distributed system. *IEEE Concurrency*, 44(44):70–78, Jan.-Mar. 1999.

[27] Stefano, A. D. and Santoro, C. The coordination infrastructure of the arca framework. In *Proc. Fourth Intl. ACM Conf. Autonomous Agents Agents 2000*, pages 4–5, 4-8 June 2000.

[28] Stefano, D., Bello, L. L., and Santoro, C. Naming and locating mobile agents in an internet environment. In *Proc. Third Intl. Conf. on Enterprise Distributed Objects EDOC 99*, pages 4–5, Sept. 1999.

[29] Stefano, D. and Santoro, C. Locating mobile agents in wide distributed environment. *IEEE transactions on parallel and distributed systems*, 8(13):4–5, August 2002.

[30] Tanenbaum, A. S. *Modern Operating Systems*. McGaw Hill, 1991 edition.