Journal of Computer Science 4 (3): 256-271, 2008
ISSN 1549-3636

Design, Validation, Simulation and Parametric Evaluation of a Novel Protocol for Locating Mobile Agents in Multiregion Environment

[1]Rama Sushil, [2]Kumkum Garg and [3]Rama Bhargava
[1](SGRRITS, Dehradun) and R/S IIT Roorkee India
[2] Department of Electronics and Computer Engineering, IIT Roorkee India
[3]Deptartment of Mathematics, IIT Roorkee India

Abstract: Mobile agents are processes that can be dispatched from source computer and be transported to remote servers for execution, have been widely argued to be an important enabling technology for future systems. Location management is a necessity for locating mobile agents in a network of mobile agent hosts for controlling, monitoring and communication during processing and it still represents an open research issue. The cost of location management strategies mainly depends on the cost of search and update. We concentrated on reducing the cost of update and improving the speed of processing of the agents. We proposed a location management technique applicable for multi-region environment in which mobile agent did not update its location at every migration. The technique named as Broadcasting with Search by Path Chase (BSPC). We used the tool time Petri net analyzer TINA to model, analyze and to simulate BSPC. We found that the BSPC behaves as expected and free from any deadlock. We measured the efficiency of BSPC and compared with some existing location management techniques by parametric evaluation. BSPC provided better scalability, location updating availability and interaction fault rate with theoretical considerations of network usage and network fault rate. It gave its best performance for applications of low CMR and having high migration rate of mobile agents within birth region.

Key words: Birth region, location finding protocol, location update and search cost, location management technique, mobile agents, mobile host, model analysis, petri net

INTRODUCTION

Mobile agents are software agents that can physically travel across a network to perform tasks on behalf of the user, under their own control as per their itinerary or deciding their movements dynamically according to execution results[18,34]. Mobile agents are one form of mobile code[16,37] that execute on machines that provide agent hosting. Mobile agent technology is making significant impact on almost all aspects of the computing discipline. It is being promoted as an emerging technology that makes it much easier to design, implement and maintain distributed systems and is an alternative to the client server model[8,18]. There are many applications of mobile agents, including network management, e-commerce, distributed information retrieval, software distribution and load balancing. Mobile agent research has evolved with the creation of many different agent platforms of similar characteristics and built by research groups spread all over the world, for optimization and better understanding of specific agent issues. A mobile agent platform is a software that can create, interpret, execute, clone, transfer and terminate mobile agents e.g. Grasshopper[19], Aglets[2], ARCA[10], PMADE[32] etc. A mobile agent host is a node in the network which has a mobile agent platform installed in it.

A mobile agent location management strategy is required by an agent owner to control the agent for many reasons like to stop processing being done by the mobile agent, to give some more task to the agent, to know the intermediary results of the processing being done by the agent, to change its itinerary and for agent-agent communication and cooperation[1,7,9,14,21,31]. Therefore, the ability to locate mobile agents while they are migrating from one node of a network to another is of great importance in the development of agent-based applications which is a part of the location management strategy. A major issue with location management is the high cost associated with location updates and search[6,13,25,28,33,38]. The goal of an efficient location management strategy should be to minimize the combined cost of the location search and update. This cost is characterized by the time taken for each

Corresponding Author: Rama Sushil, (SGRRITS) and R/S IIT Roorkee India

operation, number of messages sent, size of messages, or the distance the messages need to travel. Some problems, which exist in location management protocols, are unnecessary communication overhead, location data base server bottlenecks, high location update and search cost.

We concentrated on location update schemes used in existing location management strategies. We propose a location management technique named BSPC (broadcasting with search-by-path-chase) which uses the general concepts of distributed systems[15,17,24,29] and is based on search-by-path-chase (SPC)[12]. In BSPC, We propose a location update scheme which costs less for applications having a low frequency of queries for contacting mobile agents i.e., for low call to mobility ratio (CMR). We model the BSPC using TIme PetriNet Analyzer (TINA)[39], perform the reachability analysis, structured analysis and simulated for total seven queries, some are in processing and some waiting to process as an initial marking of the net.

Some previous studies have been made on the location management and message delivery protocols in a mobile agent computing environment: DL, PP, Blackboard, Shadow, SPC, HB, optimal location update scheme, MBLM, Scalable Hash-Based Mobile Agent Location Mechanism, Mailbox-based scheme for mobile agent communications. In Database logging protocol (DL)[1,12] location information of mobile agents is stored at a specific server called location database server, upon every migration. The location information is used to find the mobile agents. As the location information is stored in centralized way, location server can represent a bottleneck in several conditions when number of mobile agents grows, mobile agents migrate frequently. If mobile agent moves far away from a specific server, the cost of location update comes to be relatively high. Moreover, if the location information in specific server is not latest means at the contact time if a mobile agent already left before catching the agent a following problem arises.

In path proxy protocol (PP)[1,12] a mobile agent leaves a forwarding proxy at the source node at every migration. Mobile agent is located by following chain of proxies. No location update procedure is used. If path proxies are long and even one of the path proxies fails, mobile agent cannot be located. The shadow protocol[7,20] is a combination of DL protocol and PP protocol. It provides functionality for locating agents, termination and for orphan detection. A mobile agent updates current location to an associated shadow according to TTL (time to live), which is a particular fix time interval. After the TTL a mobile agent updates its current location to its shadow instead of updating at every hop and by this method path proxies get cut short. In the shadow concept, each application creates one or more dependency objects called shadows, a data structure on a place. An agent is an orphan when the shadow no longer exists.

In an optimal location update and searching algorithm for tracking mobile agent, mobile agent updates its location after it migrates through continuous d hosts since its last update. There is an optimal threshold d that makes the total cost of search and update minimized. If d assigned dynamically, it is called dynamic location update scheme[25,40]. Blackboard protocol[41] maintains the blackboard at each node which is a shared information space for message exchange. When any mobile agent wants to deliver a message, it puts the message in the blackboard no matter where receiving mobile agents are or when they read it. Afterword the receiving mobile agents move to the corresponding node where a message is stored and get the message. In this protocol, every node has storage where messages are deposited. In order to receive a message, mobile agents must move to the corresponding node, which makes unnecessary communication overhead occur. Because of not regular migration pattern message is not delivered immediately.

The SPC protocol[12] is a combination of DL protocol and PP protocol, applied to a multi-region mobile agent computing environment. Location information is stored in a distributed way at a Region Agent Register (RAR) or a Site Agent Register (SAR). Agent is achieved by following a part of the links that the agent has left on two registers. Number of location update operations are reduced in SPC protocol by applying an optimal location update and searching algorithm for tracking mobile agent[25] on SPC. The modified protocol is named as MBLM (Movement Based Location Management)[28,32]. This protocol is implemented on PMADE[32] and found to have lesser cost of location update than SPC protocol.

Scalable Hash-Based Mobile Agent Location Mechanism[27] have special agents called information agents (IAgents). IAgents are responsible for maintaining the current location of a set of agents. Set of mobile agents associated with each IAgents is determined through the hash function. This association changes over time as new IAgents are created or existing IAgents are merged depending on the system workload.

A Home Blackboard (HB) protocol[36], location management and message delivery protocol, concentrated more for confirmed message delivery. A HB protocol is a hybrid of a DL protocol and a Blackboard protocol[41] in order to complement each

other and overcome. Mailbox-based scheme[11] assigns each agent a mailbox to buffer messages, but decouples the agent and mailbox to let them reside at different hosts and migrate separately.

LOCATION MANAGEMENT WITH BSPC

The protocol function is based on an efficient algorithm, which follows a part of the links left by the agents on the registers of the visited sites or regions and this protocol uses broadcasting to search an agent if mobile agent is in its birth region.

MATERIAL AND METHODS

Location finding requires the presence of a suitable repository of the current locations of all the agents of the entire distributed system. Let us refer to a Name and Location Base-NLB-collecting the tuples (m, α, λ) with m the name of the agent α which is currently placed at location λ. No hypothesis is made about the structure of m, α and λ, which merely represent identifiers for the relative objects regardless of their real structure, or on the nature of the NLB (e.g., whether it is centralized or distributed, or where it is really located). Any location protocol for mobile agents deals with three aspects: name binding, migration and location, each related to a particular phase in the agent's life. On NLB, we define four operations:

- Bind (m, α, λ): performed when a name m is assigned to mobile agent α, which is currently placed at location λ. This operation causes the insertion of the new tuple m, α, λ in the NLB. As the agent name has to be unique, this operation fails if a tuple with the name field equal to m already exists in the database
- Newloc (m, α, λ´): performed when agent α changes its location, by migrating to λ´. This operation changes the tuple (m, α, λ) already present in the NLB, into the new tuple (m, α, λ´)
- Find m→(m, α, λ´): performed when agent α has to be located in order to interact with it. Given agent name m, this operation returns the relevant tuple if present in the NLB, thus determining the bound agent α and its current location λ
- Unbind (m): performed when name m is no longer used (e.g., the agent is disposed of). This operation causes the deletion of the relative tuple from the NLB

Solving the locating problem for mobile agents implies finding a suitable way to implement the NLB and the four operations previously defined. This is not simple, as several aspects have to be taken into account which may affect the performance of the whole multiagent system. Any solution has to deal with the traditional issues in distributed systems computing, meeting the reliability, efficiency and scalability requirements. These aspects are tied to the real architecture of the NLB: As a centralized solution in a wide distributed environment is not reliable, efficient, or scalable, a distributed approach opens the question on how and where to distribute the information effectively. This affects the choice of the protocols used for executing the four primitives, which, if not suitably designed, can degrade the performance of the multiagent application. In this sense, the most critical operations are newloc and find, as they are used very frequently during the mobile agent's life. An inefficient or unreliable newloc may significantly affect the performances of the agent migration process and the same applies to the find primitive as concerns the interaction with a given agent. In addition, it is worth noting that the finding process does not end when the agent location is found, but when the given agent is really reached in that location. It may be the case that, once the current location of an agent is found, the latter has left the site, thus requiring repetition of the entire process (location finding + agent catching) until the agent is reached. Thus, when the agent is rapidly moving, several retries may be required, making interaction with the agent burdensome.

System design for BSPC: Computing environment is considered as the collection of regions. A region is a collection of mobile agent hosts (Fig. 1). Different hosts can spawn different mobile agents, so a particular mobile agent is spawned by a particular host which belongs to a particular region; this region is called the birth-region of this particular mobile agent.

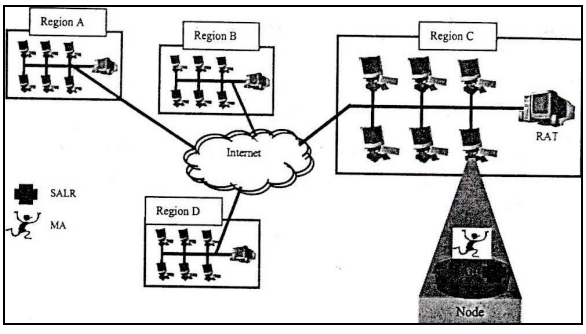

Fig. 1: Multi-region environment

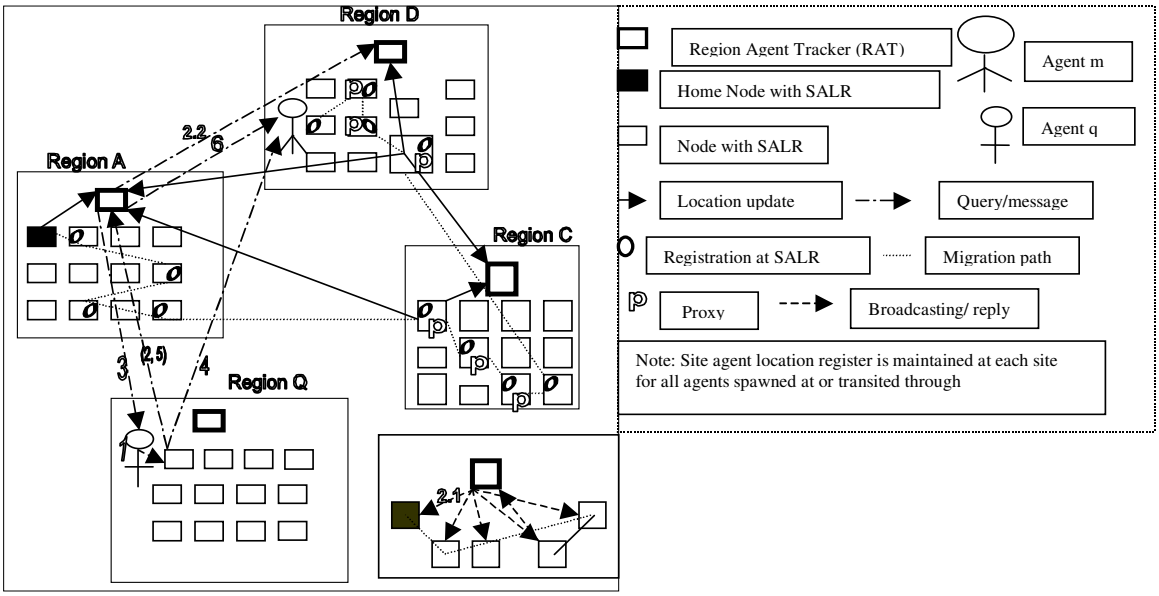

Fig. 2: BSPC function scenario

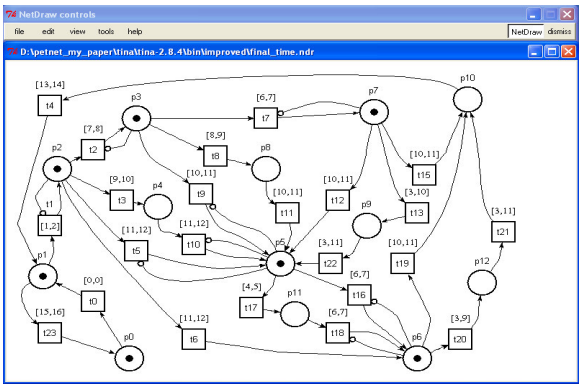

Fig. 3: Time petrinet model for interregional migration for BSPC in TINA

In each region there exists a site acting as a Region Agent Tracker (RAT), which manages a database called Region Agent Location Register (RALR). RALR stores the information about all the agents that have been created in the region or have transited through it. On each site there is a Site Agent Location Register (SALR), which contains information about all the agents that have transited through (in the past) or are at that location (Fig. 3).

Mobile agent migrations can be intraregional or interregional. In case of intraregional migration when region is not the birth region, the relevant entries are updated in both SALR and RALR of the source region. Before starting all the updating operations, an exclusive lock is placed on the entry of the SALR relevant to particular mobile agent and is released when update operation is finished (or an error occurs). It ensures the exclusive access to SALR and resolves several inconsistencies which may happen in case of concurrency between migration and interaction. SALR locks allow concurrent interaction and migration processes to be serialized, thus avoiding the agent's status inconsistencies.

In case of the interregional migration, the protocol updates SALR of the source region, RALR of the birth region and RALR of the source region if the source region is not the birth region and the RALR of the destination region and finally updating the SALR of the destination region. When the source region is different from the birth region the birth region RALR is updated in a background (concurrent) process. With the hypothesis, that exclusive lock on each entry of the register, we can assert that, for each location reached λ at time t_l by an agent named m, querying $SALR_\lambda$ i.e., site agent location register of location λ (at time t_q (with $t_q \geq t_l$) 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 t_l). exclusive lock is used not only to control the concurrent access to an entry of the SALR, but also to prevent inconsistencies which may happen during concurrent interaction and migration processes. A similar assertion can be made for RALR, 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 the agent, the updating operations performed during the migration phase are designed in order to shorten this path, thus increasing interaction efficiency and reducing the overhead. Locating an agent follows the following steps: First, the name of the agent's region of birth is extracted from the name, m and then the relative RAT 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 RAT of the new region is contacted and the search is repeated.

In the former case, the SALR of the location is queried: If the resulting tuple contains a value for α then the agent is found otherwise, the location information λ is used to restart the search. In particular, if λ refers to a GLI, the relevant SALR is contacted, while, if λ refers to a region, the relevant RALR 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 SALR is queried for an agent named m, an exclusive lock is set on the relevant tuple and is reset only if the agent does not reside in that place (i.e., the α field of the tuple is nil), otherwise the lock is maintained.

Maintaining SALR locked allows the execution of the overall location finding and interaction procedure as an atomic activity, thus preventing an agent that has just been located from migrating until the interaction has taken place (otherwise the agent might not be found and another location re-attempt will be necessary). Even if an interaction takes place immediately after a location phase, the proposed solution could increase the execution time of the destination agent; an alternative could be to forward the message to chase the migrating agent. However, to privilege agent collaboration with respect to autonomy, the first solution has to be preferred since a message can strongly influence the behavior of the destination agent (for example, it can influence the choice of the next site in the itinerary).

BSPC protocol: Broadcasting with Search-by-Path-Chase protocol (BSPC) functions are given below in algorithmic form.

- Agent q makes a request to the location management protocol (l. m. p.) to locate agent m. Location management protocol is available with each host as part of the mobile agent system or as a separate location management module. The l. m. p. extracts the birth region of the agent to be located from its name
- The birth region's RAT (Region Agent Tracker) is contacted. As per location information, the following steps take place
  - If this region is the birth region, RAT broadcasts a query to all its member hosts (MH). The host on which agent m is residing returns the 'agent found' message and locks m for migration, else
  - The related RAT is contacted and the birth region RAT uses this information to start locating m in that region
- RAT returns the location information to the requesting host's l. m. p. which then returns the location of m to agent q
- Agent q communicates with agent m and informs agent m's birth region RAT when completed
- RAT unlocks agent m, making it free to move

Complete scenario of the BSPC protocol is shown in Fig. 2, as the agent migrates out of its birth region or roams within it. Table 1 explains the meaning of the contents of the Region Agent Location Register and the Site Agent Location Register.

Location update operations: Performance and reliability of the BSPC protocol strongly depends on the register update operations made during migration and binding operation. To avoid the burden of agent migration, the protocol aims to minimize interregional messages with respect to intraregional ones. With the assumption that the connections between sites in the same region are faster and more reliable than connections between different regions BSPC can have les overhead and improved efficiency. Commonly in WANs like Internet: Sites belonging to the same subnetwork6 are often connected by LANs (10-100 Mb sec^{-1}), while connections between sub networks are point-to-point links working at a lower speed (64 Kb s^{-1}-2 Mb s^{-1}).

The binding phase occurs when agent α is spawned, there is registration of the agent's name m and the birth location λ_m of α in $RALR_{m:region}$ (m.region is the region of birth). This is handled by a two-step protocol performed by the platform executing at location λs. First, $RALR_{m:region}$ is contacted and, here,

Table 1: RALR/SALR tuple meaning

Ralr Tuple	Meaning	SALR 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 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, α, nil)	The agent is in the same location as the SAR

the tuple (m, λ_s) is registered. Then, the tuple (m, α, nil) is stored in $SALR_{\lambda s}$. Before starting all the updating operations, an exclusive lock is placed on the entry of the SALR relevant to m and is released when registration is finished (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 location, 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 $\lambda_{d.region} = \lambda_{d.region}$ and it is the birth region then the aim is to not to update the entries relevant to the migrating agent in both $SALR_{\lambda s}$ and $RALR_{d.region}$. But the RALR of the birth region is updated only when the agent is crossing the region.

If λ_s and λ_d belong to two different regions (interregional migration), the migration protocol has to update $SALR_{\lambda d}$, $RALR_{m:region.}$ (if $\lambda_{d.region} \neq$ m.region), by writing $\lambda_{d.region}$ as location information; it also updates by writing $RALR_{\lambda d..region}$ as location information and, finally, $SALR_{\lambda s}$ registering the presence of the agent.

This allows the location finding protocol, which starts from the region of birth of the agent, to reach the current region of the agent and, finally the current location. At location λ_s first the entry of the SALR is locked, then, after agent transfer, the tuple (m, nil, $\lambda_{d.region}$) is stored on the SALR, then the tuple(m, $\lambda_{d..region}$) is stored on $RALR_{\lambda d..region}$ 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 SALR and the $RALR_{\lambda d..region}$ is updated with λ_d as location information. When migration ends, first $SALR_{\lambda d}$ is updated by storing the tuple (m,α,nil) then the lock in the SALR is released and, finally the agent execution is resumed. At this time, if $\lambda_{d.region} \neq$ m.region, a background (concurrent) process is started in λ_d that aims to remotely update $RALR_{m.region}$ by writing a tuple with $\lambda_{d.region}$ as location information.

In the described protocol, SALR locks play a fundamental role: They not only ensure exclusive access to SALR, but above all they help to resolve several inconsistencies which may happen in the case of concurrency between migration and interactions.

In fact, if an interaction request arrives when the agent is migrating; the latter is neither at location λ_s nor at location λ_d but on the net, thus it cannot be contacted anyway. 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 author's opinion, a better solution is to wait for migration completion and then contact the agent at the destination location. In our protocol, this is automatically performed exploiting SALR locks.

In addition, it is worthwhile to remember that in many object-based mobile agent frameworks (such as Arca, Voyager, Mole, etc.,) the interaction between agents is performed through a kind of remote method invocation. Often this means that, from the point of view of the receiving agent, the invoked method is executed concurrently with the main activity of the agent, i.e., the main activity and the method invocation execute as different threads. As both of these threads can access the attributes of the same object-agent, the migration of status and code and method invocation operations on the same agent must be mutually exclusive otherwise, an interaction performed at source location, which updates an attribute already transferred to the destination location, will cause the loss of the updated information.

A similar situation may happen during interaction, which causes attribute modification, the main agent's threads start a migration process. Using SALR locks allows concurrent interaction and migration processes to be serialized thus avoid agent's status inconsistencies.

Example: To illustrate how the Broadcast-Search-by-Path-Chase protocol works, let us consider a distributed environment with three regions, named abc.it, klm.org and pqr.com. Also, let us suppose a mobile agent, spawned in the site www.abc.it and named agent:roamer@abc.it that has to accomplish the following itinerary: sun03→abc.it →www.klm.org →eye.pqr.com→www.pqr.com. Now, let us introduce the following notation:

- $RALR_{RegionName}$ = (AgentName, Location) is the entry in the RALR of the region RegionName relative to the agent AgentName. If the location represents a region, the name is prefixed with @

- $SALR_{SiteName}$= (AgentName, α, Location) is the tuple in the SALR of the site SiteName relative to the agent α whose name is AgentName.

During the mobile agent lifetime, the registers of the regions and sites involved are updated according to the agent's itinerary. Let us suppose we want to locate the agent. Starting from its name agent:roamer@abc.it, the birth region is extracted (abc.it) and the relative ANS is contacted. The latter will return @pqr.com as the location information; this means that the agent could be found in region wipro.com. The ANS of the latter region is then contacted, which returns www.pqr.com as location information. Finally, by contacting the host www.pqr.com, the agent can be sought.

MODELING BSPC USING TINA

A PetriNet is a mathematical formalism intended to be used for modeling, analysis and simulation of different kinds of distributed and parallel systems and processes[3,22,23,26,30]. In PetriNets, there are places and transitions, places are denoted by circles and are used to indicate system states like processing, accessing or waiting etc., transitions are denoted by directional edges and show the change of states after an event. Inhibition arcs are used for modeling error conditions in PetriNets. An inhibitor arc from a place to a transition disables the transition if the corresponding input place is not empty. In our petrinet model we used the inhibitor arc for locking function, when particular agent A_m is updating its location in the register that register should not be accessible for retrieving the location of A_m means register should be locked during that period of location update operation**.** Places are allowed to contain several tokens. A token in a TPN may be in one of the two following states: available or unavailable. Initially each place p contains tokens available. A transition t may fire when, 1. There is at least one token in each of its input places, 2. There is no token in any of its inhibiting places, 3. Its enabling function evaluates to true and 4. No other transition u with priority over t and satisfying 1, 2 and 3 exists. This removes the token from the input place and put the token in the output place. A token remains unavailable in input place during the transition occurs.

The most important features of PetriNets are their graphical representation of modules and their precise mathematical foundation, which are the main reasons for the large number of analysis techniques developed for PetriNets. The main objective of PetriNet modeling is to check the formal properties of a proposed protocol or solution, particularly its liveness, to avoid potential deadlocks and possible conflict activities. PetriNets allow a clear description of the concurrency, conflicts and synchronization of parallel processes. Thus they present a simple, yet elegant formalism for modeling parallel processes.

The Timed Petrinet (TPN) is an extension of the ordinary Petrinet in which a transition fires after a predefined interval, once it is enabled. It can be used for the modeling, functional analysis and correctness evaluation of time dependent protocols. In another version of the Timed PetriNet, which we have used, two time values a and b are given for a transition. The actual firing is instantaneous but this must not happen before time a or after time b from the instant of enabling. We consider only the time an agent takes to complete a task. The number of states in the PetriNet does not play any significant role in the theoretical analysis of the model.

TIme petriNet Analyser software tool TINA[5,39] proposes the construction of a number of representations for the behaviour of Time Petri nets, in addition to the graphic-editing facilities. Various techniques are used to extract views of the behavior of nets, preserving certain classes of properties of their state spaces. For Petri nets, these abstractions help prevent combinatorial explosion, relying on partial order techniques such as covering steps and/or persistent sets. For Time Petri nets, which have, in general, infinite state spaces, they provide a finite symbolic representation of their behavior in terms of state classes. For BSPC we have performed the reachability analysis for finding the deadlock and liveness in BSPC models, for two different sub-functions. Analysis is needed to check whether the resulting system is free of logical errors. Many process designs suffer from deadlocks and live locks that could have been detected and avoided using verification techniques. Validation is needed to check whether the system actually behaves as expected. Validation is context dependent while verification is not. A system that deadlocks is not correct in any situation. So verifying whether a system exhibits deadlock is context independent. Validation is context dependent and can be done only with knowledge of the intended process.

Eventuality of protocol operation and hence its liveness properties can be easily specified using Timed Petrinets because of the restrictions that enabled transitions must fire as soon as the enabled input places have available tokens. This is not possible to do using Finite State Machines (FSM) or ordinary Petrinets. Enumeration Analysis[4] consists of the construction of an accessibility graph from the initial marking M0. The

graph is obtained by firing one by one all the possible transitions starting from the initial marking until no new transition could be fired. Each node of the graph corresponds to a marking of the system, each arch to the transition which allowed generating the new marking. This is the most common method used for the verification of properties in Petri Nets.

For Colored and Predicate-Action Petri Nets, the principle used to construct the accessibility graph is the same, only the fire rules change. Some techniques of reduction and projection can be used during the enumeration analysis to reduce the size and the complexity of the graph. The reduction and projection techniques allow obtaining simplified views of the system. The reduction technique allows reducing the graph before the accessibility graph is built. The projection allows one to reduce the accessibility graph in order to obtain an equivalent abstract view. It is up to the person analyzing the system to specify the adequate equivalence relation as well as the transitions of the model that will remain visible (the others will become interns and non visible).

Structural Analysis consists of specifying invariants associated with places. The results obtained are independent of the initial marking. The invariants represent the fact that a predicate joining the marking of a certain number of places remains always valid. A transition tj is said to be live for an initial marking M0 if for all marking accessible from Mi belonging to M0* there exists a firing sequence containing tj from Mi. A Petri Net is said to be live for an initial marking M0 if all the transitions are live. In other words there are not transitions in the Petri Net that can not be fired.

For a clear description of the concurrency, conflicts and synchronization, we plan to consider only certain subsystems to model and analyze each using Time Petrinets. We have considered the following two subsystems:

- Interregional migration
- Intraregional migration when migration is in the region other than the birth region

In the following discussion A_f is the agent who queries to find another agent A_m. Fig. 3 shows all possible places with their states and state transitions for the BSPC protocol in a TINA model, when locating a mobile agent during interregional migrations. The following is a description of the places and transitions:

ANALYSIS AND SIMULATION

We have performed the reachability analysis[35] of the BSPC protocol by generating its PetriNet for

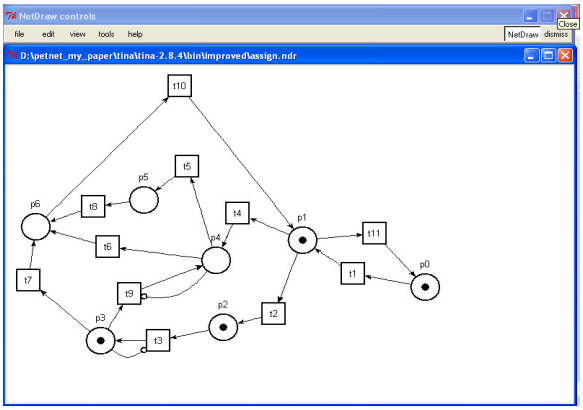

Fig. 4: Petri net model for intraregional migration of the mobile agent in the region other than the birth region

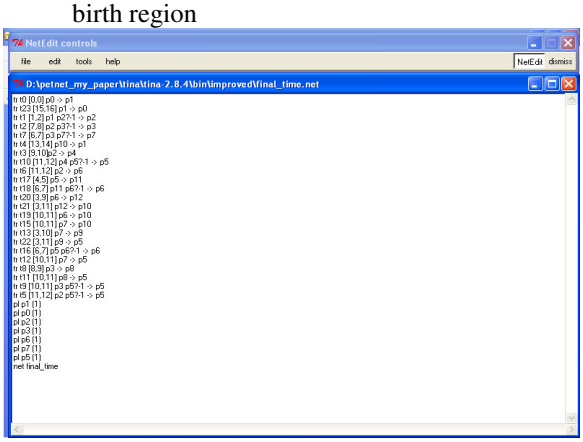

Fig. 5: Model specification in TINA for interregional migration of the mobile agent

interregional migration and intraregional migration. A time petrinet analyzer software called TINA is used to perform the automatic validation of properties in the Petri Nets, which uses enumeration approach[4].

Reachability: a marking Mi is said to be *reachable* from an initial marking M0 if there exists a sequence of firings that transform M0 to Mi. It has been proved that the reachability problem is decidable[22] although it takes exponential space (and time) to verify in the general case.

Boundness: a place Pi is said to be *bounded* for an initial marking M0 if for all marking accessible from M0 the number of tokens in Pi is finite. A Petri Net is said to bounded for an initial marking M0 if all the places are bounded. Formal specification of BSPC model in TINA is shown in Fig. 5.

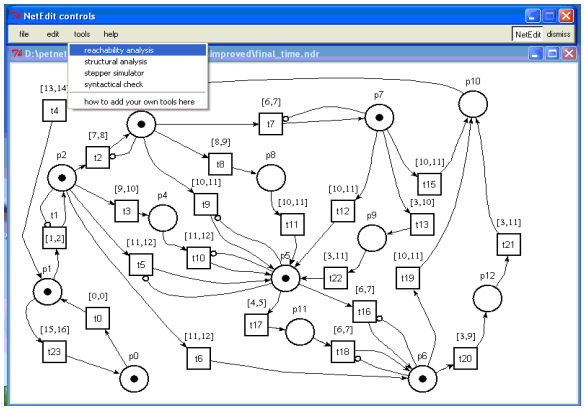

Fig. 6: Model analysis options, a snapshot in TINA

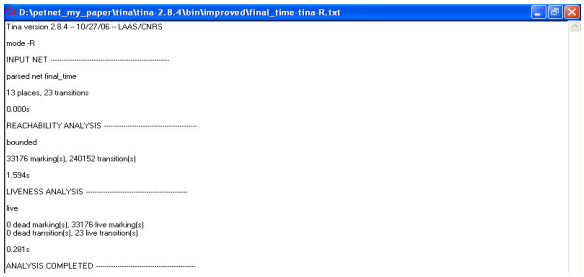

Fig. 7: Output in quiet format for marking graph upto level 1for petrinet in Fig. 3

TINA provides the tools for reachability, structural analysis and stepper simulator (as shown in Fig. 6); in this paper we performed the reachability analysis for marking graph at level 1, level 2 and at level 3, structural analysis and stepper simulator too for models shown in Fig. 3 and 4. Marking graph at level 1, output in format quiet reports in Fig. 7, that petrinet has 13 plces, 23 transitions, TINA takes 0.0 seconds for this task.

In reachability analysis, Petrinet (Fig. 3) is found bounded, has 33176 markigs, 240152 transitions and it takes 1.594s for this task. Liveness analysis in TINA reports that net is live, has nil dead markings and transitions, has 33176 live markings with 23 live transitions and takes 0.281s for this task (Fig. 7). Similarly reachability analyses for level 2 and level 3 of the petrinet (Fig. 3) is found bounded and live.

In reachability analysis, Petrinet (Fig. 4) is found bounded, has 182 markigs, 763 transitions and it takes 0.000s for this task. Liveness analysis in TINA reports that net is live, has nil dead markings and transitions, has 182 live markings with 11 live transitions and takes 0.000s for this task (Fig. 8). Similarly reachability analyses for level 2 and level 3 of the petrinet (Fig. 4) is found bounded and live.

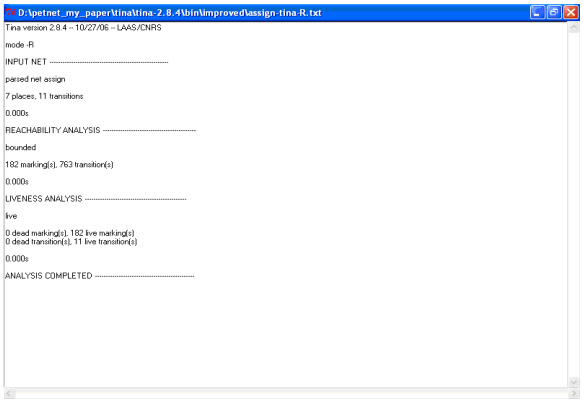

Fig. 8: Output snapshot of reachability analysis of the model for marking graph at level-1, for petrinet in Fig. 4

For simulating the BSPC, Stepper simulator is run for model with initial marking shown in Fig. 3. There is smooth running of all the queries in the form of tokens. With the shown initial markings and for some other initial markings we could see the processing of the seven queries successfully. Similarly model shown in Fig. 4 is also simulated using the same marking shown in the figure and for some other markings too.

PARAMETRIC EVALUATION AND COMPARISON RESULTS

To evaluate a location management technique for mobile agents, following are some considerable parameters:

- **Availability:** It is defined as the percentage of the time in which the system works. According to the[1], the availability can be calculated as following by denoting by A, A= MTTF/(MTTF+MTTR) where MTTF means mean time to fail and MTTR means mean time to repair
- **Scalability:** It measures the overall system response when the number of agents grows
- **Migration overhead:** It is defined as the percentage of the time spent by the execution of the location update operation during migration process. It can be determined using the following formula

$$O_m = T_u / (T_u + T_m)$$

where O_m is migration overhead, T_u is the time spent in location update operation and T_m is the duration of the agent migration from one host to other.

Interaction overhead: It is defined as the percentage of the time spent in operation of finding the current location of the agent to be located from the location database and the duration of the catching during an interaction process. The following formula using notation O_i can determine it; $O_i = T_c / (T_c + T_i)$

Where, T_c is the duration of the location finding and the agent catching phase, while T_i is the duration of the interaction (i.e. message sending). Any well-designed agent location finding technique should aim to achieve the best values for these parameters. It is hard to guarantee for having simultaneous optimization of all these parameters. Therefore, aim remains to obtain a good trade - off between all above parameters

Message complexity: It is defined as the total number of messages needed to travel around the network to locate an agent.

We calculate the migration availability and interaction availability by calculating the location updating fault rate and location finding protocol availability respectively. Both we calculate in terms of fault rate of generic network link fault rate and site fault rate. Assumptions: Ignoring any node failure i.e. we expect there is no failure of any node.

Availability strongly depends on Mean Time To Fail (MTTF) and Mean Time To Repair (MTTR) of the system parts (computers, network etc.) involved in migration and interaction. MTTF can be expressed as $MTTF=1/R_f$ where R_f is the average fault rate. Thus to evaluate the availability, we will compute the overall fault rates of the migration and interaction phases.

Location updating availability: As far as migration is concerned, the fault rate can be expressed as the sum of the agent transfer fault rate and the location updating fault rate. The contribution of the first term does not depend on the location protocol used and will not be considered since it does not affect our comparison. For evaluating migration fault rate concentrating on location updating fault rate only, In the Database Logging (DL) technique, each migration implies the updating of a remote database and involves the network and a specialized site (the location database site):

$$R_f^{(DL)} = R_{fn} + R_{f\check{s}} \quad (1)$$

In the Path proxy (PP) technique, each migration involves the creation of a proxy in the local site:

$$R_f^{(PP)} = R_{fs} \quad (2)$$

In SPC protocol, during intraregional migration (birth region or other region), if the RAR of the region cannot be updated and only Site Agent Registers (SAR) are updated, the agent can still be located;

$$R_f^{(SPC)} = R_{fs} \quad (3)$$

While, for interregional migration the updating operations of the RAR of the source region and the agent's region of the birth can fail without affecting the correctness of the location phase. This implies that, in evaluating migration availability, we can consider the fault tolerance of the mandatory operations alone, unless the overall migration has had to fail because the agent could not be reached.

These (operations without which locating mobile agent is not possible) are the updating of $SAR_{\lambda s}$, for intraregional migration and updating of $SAR_{\lambda s}$ and $RAR_{\lambda d.region}$, for interregional migrations. By indicating the percentage of intraregional migration in the whole distributed system with £, we can express the total migration fault rate of the SPC protocol as a mean between intraregional fault rate ($R_{f\,s}$) and interregional fault rate ($R_{fs} + R_{f\,\check{s}} + R_{f\tilde{n}}$):

$$\begin{aligned} R_f^{(SPC)} &= £\, R_{fs} + (1-£)(R_{fs} + R_{f\,\check{s}} + R_{f\,\check{n}}) \\ &= R_{fs} + (1-£)(R_{f\,\check{s}} + R_{f\,\check{n}}) \end{aligned} \quad (4)$$

In case of BSPC protocol, during intraregional migration within birth region of the mobile agent, no update operation is required; agent can still be located (as within birth region broadcasting is used, on locating request only), means no RAR or SAR update is done. So intraregional fault rate:

$$R_f^{(BSPC)} = \text{nil} \quad (5)$$

During intraregional migration other than the birth region if RAR cannot be updated, still agent can be located (as SAR are updated). So intraregional fault rate:

$$R_f^{(BSPC)} = R_{fs} \quad (6)$$

While for interregional migration

$$R_f^{(BSPC)} = R_{fs} + R_{f\,\check{s}} + R_{f\,\check{n}} \quad (7)$$

By indicating the percentage of intraregional (birth region) migration as $\dot{I}_b$, intraregional migration (other than birth region) as $\dot{I}$, Now we can express the total migration fault rate of BSPC protocol as a mean

between intraregional fault rate R_{fs} and interregional fault rate ($R_{fs} + R_{f\check{s}} + R_{f\check{n}}$):

$$R_f^{(BSPC)} = \dot{I}_b \text{ (nil)} + \dot{I}\, R_{fs} + (1-(\dot{I}_b + \dot{I}))(R_{fs} + R_{f\check{s}} + R_{f\check{n}})$$

or

$$R_f^{(BSPC)} = (1-\dot{I}_b)(R_{fs} + R_{f\check{s}} + R_{f\check{n}}) - \dot{I}(R_{f\check{s}} + R_{f\check{n}}) \quad (8)$$

From Eq. 8, we observe that more is the value of $\dot{I}_b$ less is the total fault rate and $0 < \dot{I}_b < 1$ and $0 < \dot{I} < 1$.
From Eq. 1, 2, 4 and 8, we can assert that PP protocol offers the best location updating availability degree; DL has the worst behavior while BSPC offer better than SPC for more intraregional migrations within birth-regional.

Interaction fault rate: For interaction fault tolerance, we will evaluate the availability of the location finding protocol required at time t by generic agent wants to interact with the ith agent. Once again, this will be determined by calculating the total fault rate.

In the DL technique, location finding involves a query to the location database site:

$$R_f^{(DL)'}(i, t) = R_{f\,n} + R_{f\,\check{s}} \quad (9)$$

Where s(i, t) represents the number of migrations performed by the ith agent at time t.

If PP is used to find an agent, the complete agent path starting from the home site has to be followed:

$$R_f^{(PP)'}(i, t) = s(i, t)\,(R_{f\,n} + R_{f\,s}) \quad (10)$$

Applying the query propagation technique, the interaction fault rate of the SPC protocol strongly depends on the updating operations performed during migration. The fault rate for SPC comprised between a minimum value $R_{f(best)}^{(SPC)'}(i, t)$ in best case when during migration all the updating operations are performed and a maximum value $R_{f(worst)}^{(SPC)'}(i, t)$ in worst case when only the mandatory operations are performed. Assuming that the searched agent is not in its birth region and in its itinerary it does not visit the same site and region more than once, then interaction fault rate for SPC in best case and worst case values are:

$$R_{f(best)}^{(SPC)'}(i, t) = 2(R_{f\,n} + R_{f\,\check{s}}) \quad (11)$$

$$R_{f(worst)}^{(SPC)'}(i, t) = r(i, t)(R_{f\,n} + R_{f\,\check{s}}) + s(i, t)\,(R_{f\,\check{n}} + R_{f\,s}) \quad (12)$$

The general case can be expressed as:

$$R_f^{(SPC)'}(i, t) = k_r(i, t)\,(R_{f\,n} + R_{f\,\check{s}}) + k_s(i, t)(R_{f\,\check{n}} + R_{f\,s}) \quad (13)$$

$k_r(i, t)$ represents the number of regions in the search path from the $RAR_{m.region}$ to the current location of the agent. it depends on the number of last consecutive interregional migrations featuring only the mandatory updating operations. $k_s(i, t)$ represents the number of locations (hosts) in the search path from the $RAR_{m.region}$ to the current location of the mobile agent, they depends on the number of last consecutive intraregional migrations featuring only the mandatory updating operations. Each time $RAR\lambda_{s.region}$ cannot updated, $\mathbf{k_s}(i, t)$ increases as λ_s is now on the search path. Each time $RAR_{m.region}$ cannot be updated $k_r(i, t)$ increases as $RAR\lambda_{s.region}$ now belongs to the search path. While when migration completes with a success of all the remote register updating, $k_r(i, t)$ and $k_s(i, t)$ immediately reach their minimum values (respectively 2 and 0). So looking best case possibility, there is low probability that $k_r(i, t)$ and $k_s(i, t)$ can reach high values, unless there is very high fault rate. Obtaining an analytical expression for these two parameters requires a complex analysis, which, from our view does not give additional important information in determination of availability.

In BSPC protocol also, interaction fault rate strongly depends on the updating operations performed during migrations in any application domain and same as in SPC, if we look upon the best application domain area for BSPC (in which there is a low frequency of locating request), like in a company which has several franchise spread all over the world, a single franchise is having its own LAN and each franchise's LAN is interconnected with each other. Most of the tasks are locally managed like pay slip generation of the employees. Each franchise has several departments. If the employees records are kept in distributed manner then particular mobile agent for collecting the record of the employees will at most roam in its birth region collect the record and prepare the pay slip. So most of the migrations will be intraregional and within the birth region.

For normal application domain, with high mobility of mobile agents, with many intraregional migrations out of the birth region and there is high frequency of locating request.

$$R_{f(best)}^{(BSPC)'}(i, t) = 2(R_{f\,n} + R_{f\,\check{s}}) \quad (14)$$

For specific application domain area of low frequency of locating query and high degree of migrations within birth re:

$$R_{f(best)}^{(BSPC)'}(i, t) = R_{f\,\check{n}} + R_{f\,\check{s}} \quad (15)$$

$$R_{f(worst)}^{(BSPC)'}(i, t) = r(i, t)(R_{f\,n}+R_{f\,\check{s}})+ (1-\dot{I}_b)s(i, t) (R_{f\,\check{n}}+R_{f\,s})+\dot{I}_b\, s(i, t)(R_{f\,\check{n}}+R_{f\,\check{s}}) \quad (16)$$

The general case can be expressed as:

$$R_f^{(SPC)'}(i, t) = k_r (i, t)(R_{f\,n} + R_{f\,\check{s}})+ (1- \dot{I}_b)k_s (i, t) (R_{f\,\check{n}}+ R_{f\,s}) + \dot{I}_b\, k_s (i, t) (R_{f\,\check{n}} + R_{f\,\check{s}}) \quad (17)$$

Now, we analyze the interaction availability with respect to the number of migrations made by the agent (hop count). The results are reported in the form of graph. Where SPC and BSPC are evaluated for some reference values of k_r (i, t), k_s (i, t) and s(i, t). The graph in Fig. 2 shows that BSPC-best case in specific application domain offer the highest interaction availability and also the BSPC-general case presents a very high value. The worst performance is registered by the PP technique, which, for a high number of hop-count is also worse than the BSPC.

For making these comparisons for the BSPC protocol, we have considered the case of searching of the agent always begin from Agent Name Server of the agent's birth region.

Scalability: Scalability can be evaluated by considering the overall distributed system response when the number of agents n(t) and the number of migrations of each agent s(i, t) increase. These parameters affect the network usage (U_n). Network usage increases as the number of agents to locate grows. Site usage (U_s) indicates the number of entries used in all the location database of the distributed environment (including proxy elements). We express the global system usage (U) as the sum of U_n and U_s suitably weighted.

The determination of the usage parameters may be hard, so a simplification of the system model is required. In determining an expression for U_n, the topology of the global network should be considered which leads to evaluating the usage of the links between the various sites (Site Agent Registers) and the specialized sites (like Region Agent Registers). However if we assume a uniform distribution of the sites over the various links, the overall network usage of each link can be expressed as proportional to a factor Tm/n, where Tm is the total number of messages and n is the total number of sites queried. In this case, network usage could play a substantial role if we consider the presence of a small number of location database sites n. If C is the average capacity each link then for the condition Tm/n>C. If we assume that n is adequate for the considered environment such that Tm/n<C, we can concentrate our analysis on the role of U_s.

We calculate system usage by looking two factors first one is the number of entries related to the particular mobile agent at the sites (as the mobile agent register itself at the site) and second is the number of agents.

For DL we assume the number of database sites such that network congestion is not caused. Sites involved in DL are the location database site and the current site of the agent where agent registers its presence.

So

$$U^{(DL)}(t) = 2n(t) \quad (18)$$

For PP there is proxy on each site for reference to the next location of the agent, thus

$$U(PP)\ (t) = \sum_{i=1}^{n(t)} (1 + s(i,t)) \quad (19)$$

For SPC, site usage can be evaluated by considering the registers growth in the worst case - when all the registers are updated during migration. In best case when only mandatory updating is performed;

$$Uw(SPC)\ (t) = \sum_{i=1}^{n(t)} (s(i,t) + r(i,t)) \quad (20)$$

$$Ub(SPC)(t) = \sum_{i=1}^{n(t)} (1 + r(i,t)) \quad (21)$$

In general global system usage can be expressed as;

$$U(SPC)\ (t) = \sum_{i=1}^{n(t)} (1 + ks(i,t) + r(i,t)) \quad (22)$$

For BSPC[23], consideration of intraregional migrations is the best case, as per its best application domain we assume that each region has adequate number of sites and mobile agents locating queries such that no network congestion occur when within the birth region broadcasting is used to locate the agent;

$$Ub(BSPC)\ (t) = \sum_{i=1}^{n(t)} s(i,t) \quad (23)$$

For BSPC, worst case is when there are more inter-regional migrations and intra-regional migrations within the birth region are nil then it performs like SPC, so equation no. 20 is suitable for worst case of BSPC;

$$Uw\ (BSPC)\ (t) = \sum_{i=1}^{n(t)} (s(i,t) + r(i,t)) \quad (24)$$

$$U(BSPC)(t) = \sum_{i=1}^{n(t)} (ks(i,t) + r(i,t)) \quad (25)$$

Now, if we consider that the database sites can be designed to handle a large number of entries, their contribution in evaluating scalability can be ignored. This leads to the following equations:

$$U^{(DL)'}(t) = n(t) \quad (26)$$

$$U(PP)'(t) = \sum_{i=1}^{n(t)} (1 + s(i,t)) \quad (27)$$

$$Ub^{(SPC)'}(t) = n(t) \quad (28)$$

$$Uw(SPC)'(t) = \sum_{i=1}^{n(t)} s(i,t) \quad (29)$$

$$U(SPC)'(t) = \sum_{i=1}^{n(t)} (1 + ks(i,t)) \quad (30)$$

$$Ub(BSPC)'(t)\ a = \sum_{i=1}^{n(t)} s(i,t) \quad (31)$$

$$Uw(BSPC)'(t) = \sum_{i=1}^{n(t)} s(i,t) \quad (32)$$

$$U(BSPC)'(t) = \sum_{i=1}^{n(t)} (ks(i,t)) \quad (33)$$

Above relations shows that DL and SPC-best case present the best scalability, while PP is the worst technique. In general BSPC scalability is better than SPC.

Message complexity of SPC & BSPC: Network overhead occurs because of travel time of messages to find mobile agent location in response to a location finding query. We can calculate network overhead in terms of the messages complexity. We define the message complexity as the total number of messages needed to travel on the network, for serving single location finding query. So we derive the following expression for network overhead for single query:

Network overhead $\propto$ message complexity For BSPC protocol in case of intraregional migration (birth region), for any query 1 message will come first to the birth region of the mobile agent which is to be contacted for any reference then if there are suppose maximum m hosts in a region and minimum 1 hosts then m number of messages will be broadcasted to all hosts and the only one massage will be sent back from the host where the agent will be residing, so total n_h+2 messages are required for serving a single query, where n_h=m, irrespective of number of hops of the mobile agent and n_h denotes the number of hosts (queried) in a region.

The message complexity for SPC protocol is n_h+2 where $1 \le n_h \le m$, it depends on the number of mobile agent hops. As per the best of our knowledge we calculated it first time. Figure 5 shows the simulation results- message complexity in BSPC is not more than the maximum possible limit in SPC but equal to maximum possible limit.

In interregional migration, let the agent can migrates through (from its birth region) at the most n regions. Assume that there are m sites in each region. For total n region, one will be birth region. So $1 \le n_r \le n$. If mobile agent crossed n_r regions and in final destination region is only at one site, one message is required to reach from RALR to SALR.

Now if it is on the same SAR agent is found by using n_r+2. If there are at most m migrations within that region then maximum n_r+1+n_h ($1 \le n_r \le n, 1 \le n_h \le m$) ($n_r$ is the number of regions migrated or crossed over by the mobile agent and n_h is the number of hosts migrated within a region) total messages will be required.

Advantage of BSPC over SPC:

- In SPC there is an update operation at each migration whether the agent is in its birth region or in any other region, while in BSPC these update operations get canceled in case of the agent is in its birth region. Therefore cost of update operation for BSPC is reduced. Figure 11 shows the same for 10 hosts in a region, it shows that cost of update operations is saved completely consequently, the speed of processing of the agent also increases, as at the time of migration, time is not wasted in doing the operation of leaving proxy and no need of contacting to the $RALR_{m:region}$ (birth region's Region Agent Location Register) also
- It will be very effective for very large network, with large number of regions and where each region has not large number of hosts, as broadcasting is expensive for large area (in our protocol it is limited by broadcasting in birth region only)
- Best possible results are expected at low frequency of queries for an agent and the low frequency of interregional migrations and high frequency of intraregional migrations (in birth region)

- Locating agents is also comparatively faster in BSPC as following the long path is not needed
- BSPC uses the memory more efficiently by reduction in saving of proxies at the host machines in case mobile agent is migrating within birth region at high frequency

RESULTS AND DISCUSSION

We developed a novel location management protocol BSPC applicable for multi-region environment. BSPC is designed for the applications having low frequency of locating queries (i.e., low call to mobility ratio) and with high frequency of migrations of mobile agents in its birth region compared to any other region. To locate the mobile agent when it is roaming in its birth region broadcasting is used. To locate the agent not present in its birth region path of proxies are followed. It allows the agents to roam more freely in their birth region without contacting to the home node or source node as no update operation is required and no proxy is to be left at migration within birth region. This improves the speed of performing the task by the mobile agent. Location database is distributed in region's registers RALR and SALR.

Petri net modeling has been chosen because of the complexity of the system and the configuration of the network. Inhibitor arcs were used for modeling lock function for the registers. This is explained by the fact that from every regular place, a transition is possible to the next regular place but if some error occurs (here error is basically the state when register is locked as being in use by the agent for location updating), transition cannot occur. Eventuality of protocol operation and hence its liveness properties can be easily specified using Time Petrinets, because of the restrictions that enabled transitions must fire as soon as their input places have available tokens.

Analysis results at different marking levels and in different output formats report that model is having 13 places and 23 transitions, reachability graph is bounded and live. Analysis clearly reports that there are nil dead markings and nil dead transitions. It is found that the Petri Net model for interregional, intraregional migration is found bounded and live**.**

The liveness property encapsulates the concept of a system which will be able to run continuously means a system which does not deadlock. In the presented Petrinet model, the request goes in the form of the token, which starts from the birth region of the mobile agent (to be searched) and passing through all other region agent registers and site agent registers, depending on the mobile agent migration path, finally token reaches to the final state (named agent found) and the protocol informs the found location information to the user/agent who requested to locate the mobile agent.

A Petri Net modeling, analysis and simulation of the BSPC using TINA, which is using the enumerative approach for analyzing the time Petri net and found the model bounded and live. Verification, validation and simulation based on TINA are found to be an efficient way to improve the design process.

We measured the efficiency of Database logging, path proxy and search by path chase and broadcasting with search by path chase mobile agent location management techniques by evaluating the parameters location updating availability, interaction fault rate and scalability.

We calculated migration availability and interaction availability BSPC[19] in terms of fault rate of generic network link and generic site (0.02fault h^{-1}) with MTTF = 50 h and fault rate of generic network link belonging to a sub net (0.01fault h^{-1}) with MTTF = 100 h, results shown in Fig. 9 tells that interaction fault rate for BSPC is lower than SPC. PP protocol offers the best location updating availability degree; DL has the worst behavior while BSPC offers better than SPC for more intraregional migrations within birth-region. DL and SPC-best case present the best scalability, while PP is the worst technique. In general case BSPC scalability is better than SPC.

Message complexity of BSPC also does not go beyond the maximum value which goes in SPC[10]. Figure 10 and 11 shows that BSPC costs less with respect to message complexity and number of update operations for low Call to Mobility Ratio (CMR) and for high frequency of migrations within birth region. In this way BSPC perform better, as a reactive approach for locating mobile agents, than the path proxies, database logging and SPC. Even than best applicable application domain of BSPC is, where there is low frequency of request for locating mobile agents and there are more (in birth region) intraregional migrations of the mobile agents like pay slips generations of the employees of an organization having worldwide branches, in which each branch have their own network like LAN, a particular mobile agent is performing the task of collecting the employee's database locally from different sections of the branch and generating the pay slips i.e. most of the migrations are within birth region.

BSPC protocol can be extended to use multicasting by RALR (called Region Agent Tracker also) for its

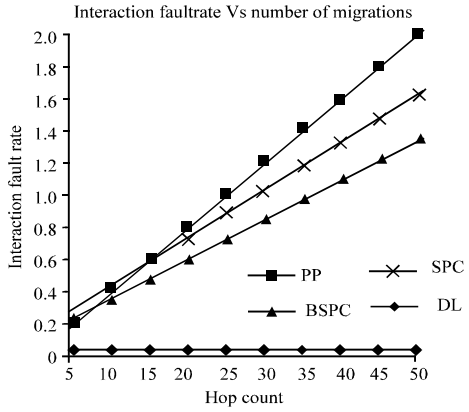

Fig. 9: Interaction fault rate vs mobile agent hop count

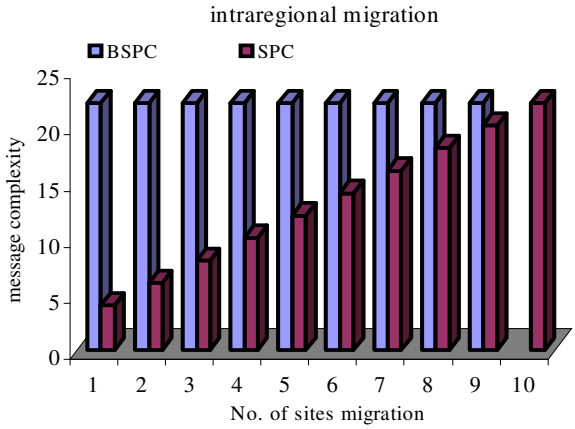

Fig. 10: Number of sites migration vs. message complexity

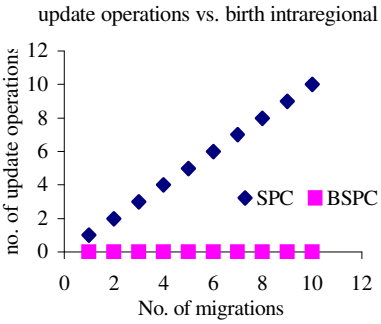

Fig. 11: Cost of update operations Vs. no. of intraregional migrations

own generated mobile agents only (whenever are they in the network) instead of broadcasting within birth-region. We intend to try for this extension in future in addition with BSPC implementation on Aglet network. Further we intend to calculate some more parameters like interaction overhead, migration overhead for DL, PP, SPC, BSPC and for some other existing mobile agent location management techniques.

REFERENCES

1. Stefano, A.D., L.L. Bello and C. Santoro, 1999. Naming and locating mobile agents in an internet environment. Proceeding 3rd International Conference Enterprise Distributed Objects (EDOC '99), Sep. 27-30, pp: 844-864.
2. Aglets Software Development Kit from IBM, Aglets 2.0.2, http://aglets.sourceforge.net/usermanual.html, 2006.
3. Berthomieu, B. and Michel Diaz, 1991. Modeling and verification of time dependent systems using time petri nets. IEEE Trans. Softw. Eng., 17: 259-273.
4. Berthomieu, V and M. Menasche, 1983. A Enumerative approach for analyzing time petri nets. Proceedings of the IFIP 9th World Computer Congress, Paris, France, Sep. 19-23, 1983, North-Holland/IFIP, ISBN 0-444-86729-5, http://www.laas.fr/tina/papers.php
5. Berthomieu, B., P.O. Ribet and F. Vernadat, 2004. The tool TINA-construction of abstract state spaces for petri nets and time petri nets. Int. J. Prod. Res., 42: 2741-2756 http//www.laas.fr/~poribet/PUBLICATIONS/ribet_2004_ijpr.ps
6. Noy, B., I. Kessler and M. Sidi, 1995. Mobile users: To update or not to update? ACM-Baltzer J. Wireless Networks (WINET), 1: 175-186.
7. Baumann, J., 1999. A comparision of mechanisms for locating mobile agents. TR 1999/11, Unv. of Stuttgart, Faculty of Computer Science, pp. 1-25, **http://elib.uni-stuttgart.de/opus/volltexte/1999/515/**
8. Berson, A., 1996. Client/Server Architecture. 2nd Edn., McGrawHill Inc, New York, NY, USA, ISBN: 0-07-005664-1, 1996.
9. Bhattacharya and S.K. Das, 1999. LeZi-Update: An information theoretic approach to track mobile users in PCS networks. Proc. 5th ACM/IEEE Ann. Conf. Mobile Computing and Networking (MOBICOM '99), pp: 1-12, Aug. 1999.
10. Santoro, C., 1998. ARCA: A framework for mobile agent programming white paper and programmer's tutorial. TR, Univ. of Catania. http://www.diit.unict.it/users/csanto/publications.html#educational.
11. Cao, X. Feng, J. Lu and S.K. Das, 2002. Mailbox-based scheme for mobile agent communications. IEEE Comput., vol 35, pp: 54-60.
12. Stefano, A.D. and C. Santoro, 2002. Locating mobile agents in wide distributed environment. IEEE Trans. Parallel Distrib. Syst., 13(8), 844-864.

13. Stefano, D. and C. Santoro, 2000. The coordination infrastructure of the ARCA framework. Proc. Fourth Int'l ACM Conf. Autonomous Agents (Agents 2000), June 2000.
14. Pitoura, E. and G. Samaras, 2001. Locating objects in mobile computing. IEEE Trans. Knowl. Data Eng., 13: 571-592.
15. Pitoura, E. and I. Fudos, 2001. Distributed location databases for tracking highly mobile objects. Comput. J., 44: 75-91.
16. Fuggetta, G.P. Picco and G. Vigna, 1998. Understanding code mobility. IEEE Trans. Softw. Eng., 24: 342-361.
17. Ceri, S. and G. Pelagatti, 1984. Distributed Database: Principles and Systems. McGraw Hill, New Yark.
18. Harrison, D.M, 1995. Chess and A. Kershenbaum Mobile agents: Are they a good idea? Technical Report, IBM research division, T.J. Watson reserch center, Yorktown heights, NY-10598.
19. Baumer, C., M. Breugst, S. Choy and T. Magedanz, 2000. Grasshopper: A universal agent platform based on OMG MASIF and FIPA standards. Technical report, IKV^{++} GmbH. http://citeseer.ist.psu.edu/baumer00grasshopper.html
20. Baumann, J.,1999. Control Algorithms for Mobile Agents. Ph.D. thesis IPVR Stuttgart.
21. Desbiens, J., M. Lavoie and F. Renaud, 1998. Communication and tracking infrastructure of a mobile agent system. In: Proceeding 31st Hawaii International Conference on System Sciences, Agent Mobility and Communication, pp: 54-63.
22. Esparza, J. and M. Nielsen, 1994. Decidability Issues for Petri Nets- a survey, Inform. Proc. Cybernet., 30: 143-160.
23. Peterson, J.L., 1981. Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs, N.J.
24. Van Steen, J., H.P. Homburg and A.S. Tanenbaum, 1998. Locating objects in wide-area systems. IEEE Commun. Magz., 36: 104-109.
25. Jie Li, H. Kameda and Keqin Li. Optimal Dynamic Mobility Management for PCS Networks. IEEE/ACM Trans. Networking, 8(3), pp: 319-327, June 2000.
26. Garg, K., 1984. Design and performance validation techniques for distributed system using timed Petrinets. Ph D. Thesis, Imperial College of science and Technology.
27. Kastidou, E. Pitoura and G. Samaras. A scalable hash-based mobile agent location mechanism. Proc. Of the 23rd Intl. conference on distributed computing systems workshops (ICDCSW'03), 19-22 May, IEEE.
28. Bavandla, M., 2004. Improving the performance of location management protocols in a multiregion environment. Master Thesis I.I.T. Roorkee.
29. Van Steen, M., P. Homburg and A.S. Tanenbaum, 1999. Globe: A Wide-Area Distributed System. IEEE Concurrency, pp: 70-78, Jan-Mar.
30. Opsenica *et al*. 2000. Petri net based modeling and simulation of email alert system. Proc. of 10th MELECON 2000 Conf., 1: 49-52.
31. Wojciechowski, P.T., 2001. Algorithms for Location-Independent communication between mobile agents. Technical Report 2001/13, Commun. Syst. Dept., EPFL, March 2001.
32. Patel, R.B. and K. Garg, 2001. Pmade- a platform for mobile agent distribution and execution. in the proceedings of the 7th International conference on information system analysis and synthesis (ISAS2001), Orlando, Florida, USA, July 22-25, pp: 287-293.
33. Sushil, R., K. Garg, R. Bhargava, 2007. Mobile agents: When to update their location? Proc, Int. Conf, Information and Communication Technology, IICT07, July 2007, Dehradun, India, pp: 852-857.
34. Tripathi, R., T. Ahamad and N.M. Karnik, 2001. Experiences and future challenges in mobile agents programming. Microsystems, 25: 121-129.
35. Ramchandani, 1973. Analysis of asynchronous concurrent systems with Timed Petrinet. Ph.D. Thesis, MIT.
36. Choi, S., M. Baik, C. Hwang, 2004. Location Management and Message Delivery Protocol in Multi-region Mobile Agent Computing Environment. Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), 1063-6927/04, 2004 IEEE.
37. Milojicic, S., W. LaForge and D. Chauhan, 1998. Mobile Objects and Agents (MOA). Dist. Syst. Eng., 5(4).
38. Li, T.Y. and J.F. Zhang, 2002. An Optimal Location Update and searching Algorithm for Tracking Mobile Agent. AAMAS'02, July 15-19, 2002, Bologna, Italy.
39. The software and instruction manuals for TINA, OLC http://www.laas.fr/OLC/OLC.html.en group of LAAS/CNRS http://www.laas.fr/, http://www2.laas.fr/tina/distribution.php.
40. Roth, V. and J. Peters, 2001. A Scalable and Secure Global Tracking Service for Mobile Agents. In MA'2001, LNCS 2240, pp: 169-181.
41. Tao, X. and J. Lul, 2000. Communication Mechanism in Mogent System. J. Software, 11: 1060-1065.