Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A Population Control Protocol for Mobile Agent Based Workflow Automation

Marina Flores-Badillo, Adrián Padilla-Duarte, Ernesto López-Mellado
CINVESTAV Unidad Guadalajara
Av. Científica 1145, Col. El Bajío, 45010 Zapopan Jal., México
{mflores,elopez}@gdl.cinvestav.mx

***Abstract*—This paper deals with agent location and loss detection in a mobile agent based workflow automation scheme. A protocol for population control of agents evolving through internet is proposed. It is supported by the handling of allowed timing in agent mission execution, allowing passive and active termination of agents, opportune localization of agents and orphan detection. The protocol is specified using stopwatch Petri nets allowing a clear definition of tasks requiring the handling of time.**

***Keywords*— Interorganizational Workflow; Mobile agents; Population control; Stopwatch Petri nets.**

I. INTRODUCTION

Workflow (WF) automation is a key factor to achieve competitiveness within an organization using current technologies. It involves both human and machine-based activities [1], for improving process performance and achieving business goals with high efficiency [2]. WF involves both processes logistics and implementation technology for developing control systems managing both task sequencing and resource allocation. Nowadays agent technologies provide flexible distributed solutions supporting business processes automation. Some works that integrate Agent technology and WF are reported in [3], [4], [5], [6], [7].

In [5] a method for agent based WF automation was proposed. In this approach, a Mobile Agent guides the process through the different organizational units (departments) where several tasks are executed according to a specific plan for the handled case. The method allows conceiving a multi agent system coping with large and complex inter-organizational WF, where the agents must travel through internet (Figure 1) in order to perform the workflow activities.

Due to the importance of the information handled by the agent, the reliability and security of WF Systems based in agent technology are crucial issues. Thus it is necessary to provide control mechanisms for mobile agents allowing the following characteristics: a) agent location: determine the site where the agent is running [8], [9]; b) termination of agents: terminate the agent execution when it is not longer needed (passive or active termination) [10]; and c) orphan detection: detect when the owner of some agent is not longer available, avoiding unnecessary use of resources. In [10] J. Baumann proposed an algorithm based on a shadow mechanism (path proxies) using an energy concept, and on a path concept for avoiding orphan agents consuming resources.

There exist several proposed protocols providing the above mentioned functionalities. They can be classed according to three concepts.

1) The energy concept which supports orphan detection for mobile agents. The protocol guaranties that an orphan agent lives a short time elapse. Each agent has a limited amount of energy to consume, every action it takes, and every resource that is used, cost energy. The protocol was first introduced by Baumann [10] and then implemented for the first time by Jochum [11]. This protocol is used in Mole [12].

2) The path concept which supports location and termination of mobile agents, this protocol is well known in the distributed systems area, some of the authors that used this concept are: R.J. Fowler in [13] for locating objects in distributed systems; A. Black and E. Jul [14], [15], for locating mobile objects; B. Awerbunch [16] for mobile users location. In a similar way, J. Baumann used this protocol for locating mobile agents in a multi-agent system [10]. Every agent leaves a path in the system, i.e. whenever it migrates, information is left behind about the agent's target place.

3) The shadow concept which supports location and termination of mobile agents, and orphan detection. In [17] it is combined the path and the energy concept for this protocol, also, some variations for this protocol are introduced. The resulting combination allows high autonomy to the agents and provides fault tolerance capabilites. The agent leaves information behind (leaves a proxy, then a path of proxies is created) but this trail is cut short at regular intervals.

Although the shadow protocol [10] is a good solution for the control of mobile agents, when it is used in applications for internet, it is possible that the whole system collapses because of the volatile proxies path that makes the path inaccessible in a certain node. This inaccessibility appears when both the shadow *ttl* and the agent *ttl* are the same and the communication delays are not considered.

In this work we propose an algorithm for the control of mobile agents called MACOP (Mobile Agents COntrol Protocol) [18], that includes localization of agents, control of agent losses, and orphan detection in wide area networks (like internet). This protocol is based on energy and path concepts, providing fault tolerance capabilities to the agent system in the network. The protocol is formally specified using an extension to Timed Petri Nets (TPN) enhanced with stopwatch automata semantics.

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

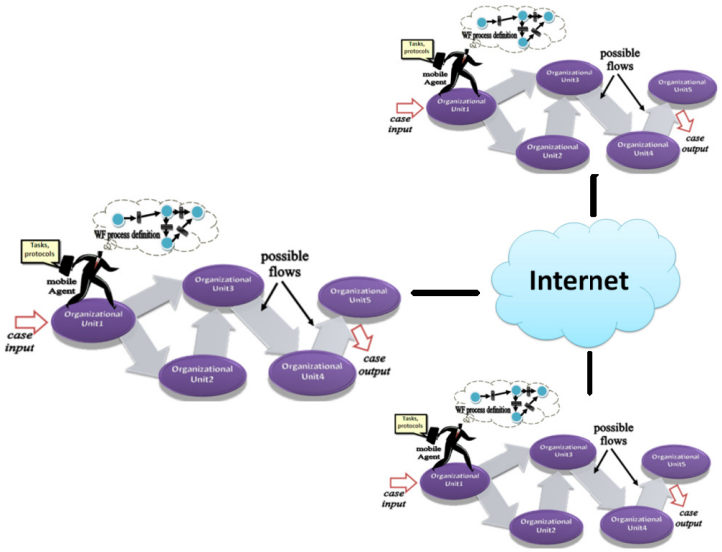

Figure 1. Workflow System as a Multi-Agent System

The remainder of this paper is organized as follows. Section II presents the MACOP protocol. Section III introduces the extension to timed Petri nets and presents the model of the proposed protocol. Finally concluding remarks are given.

II. Population Control of Mobile Agents

A. Outline

The MACOP protocol gather and extends several features included in previous proposals namely, the path proxies approach and the energy concept [17]; also it includes the spatial decoupling model [19] used for the mobile agent coordination. This protocol is composed by five functional entities showed in Figure 2 described below:

1) A *Node* is a host that provides the environment for executing mobile agents. In each node runs an instance of a special kind of agent called *Blackboard agent* (described later); each node may provide services or resources to the *mobile agents.*

2) The *Executor Agent*, also called *mobile agent*, is an active entity created with the aim to perform ordinary tasks over the network; it is controlled by a *Controller Agent*. Each mobile agent owns an amount of energy (*ttl: time to live*), which is consumed whenever the agent use resources. Also, each mobile agent has a *time reserved* (*tr*) that must be enough long for allowing the *ttl* renewal process. Whenever the energy gets low, the agent can request new energy from its *controller agent;* if it no longer exists, the mobile agent cannot get more energy and as soon as the energy is consumed, it becomes an *orphan* that can be removed. Mobile agents are allowed to communicate only through the *Blackboard Agent*.

3) The *Blackboard Agent* manages the mobile agents' input/output messages. This agent records information about the new location whenever the mobile agent migrates (a path of proxies is created). If an agent arrives to a new location it has to register with the current *Blackboard Agent*. Moreover, this agent is used to inform mobile agents about the time they took to migrate from one node to another. When the Mobile Agent receives a *delay message* it discounts that delay (*d*) from its *ttl* (*ttl-d*) and increments its *tr (tr+d).* See Figure 3.

4) The *Controller Agent* approves *ttl* renewals; this feature is similar to the *dependency object* in the energy protocol; in this way, each mobile agent depends on such agent and not on the application itself. Similarly to the mobile agent, this agent also has a *tr* and a *ttl.*

5) The *Application* creates *controller agents* and *mobile agents*; also it assigns *mobile agents* to a specified *controller agent.*

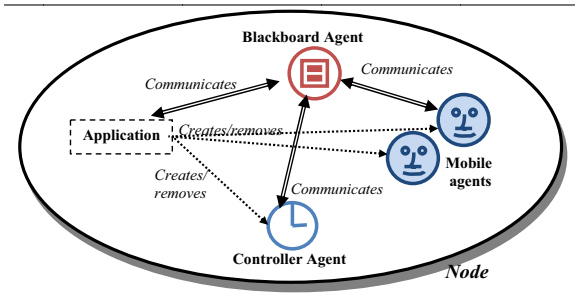

Figure 2. Elements for the MACOP protocol.

B. Agent Migration

When an agent migrates from one *Node* to another, the agent has to consider if it has enough *ttl* to support the migration process; if it is enough, the agent leaves information about the

destination node (proxy) and then starts the migration process. When the agent arrives to a new location, it sends a control message to the *blackboard agent* located in the previous node (the last visited) to inform that it has reached its new destination (see Figure 4).

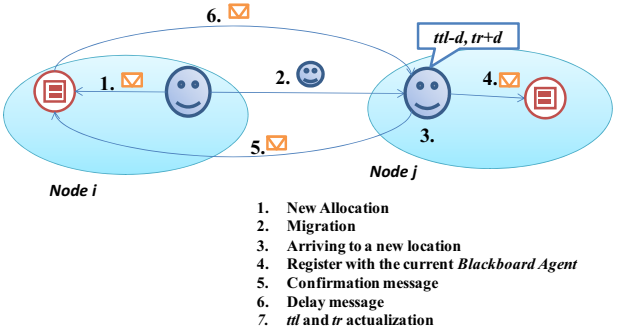

Figure 3. Migration process from $Node_i$ to $Node_j$

C. Ttl renewal

When an agent (*Controller Agent or Mobile Agent*) consumes its *ttl*, it suspends the execution of its tasks and begins the process of renewing its *ttl* while it is consuming its *tr*. If it is done by a Mobile Agent, it sends to the current *Blackboard Agent* a message with a *ttl_request*, and after a short time, if it has not received a *ttl_renewal* message, it sends another *ttl_message*; when its *tr* is totally consumed, it is considered as an Orphan Agent and ends with its execution.

When a *Blackboard Agent* receives a *ttl_request*, it sends the message to the *Controller Agent* and waits for its answer; when the answer arrives, the *Blackboard Agent* sends the right message to the requester. If the *Controller Agent* consumes all its *ttl*, it sends directly a *ttl_request message* to the Application and waits the approval. Using this energy concept we support the control of Orphan Agents (see Figure 5).

D. Agent Elimination

Whenever an Agent leaves a proxy in the *blackboard agent* a path of proxies is created. This path of proxies can be followed if we need to locate an agent.

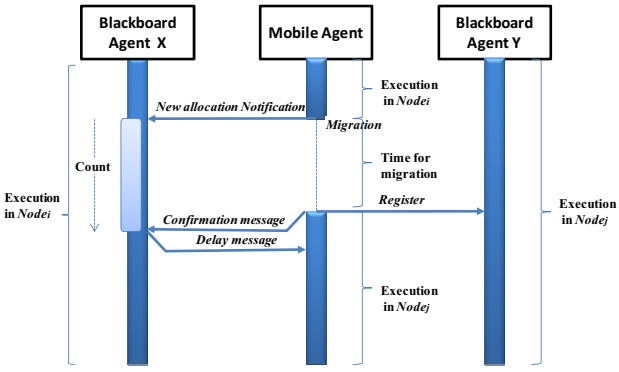

Figure 4. Interchange of messages between the *Blackboard Agent* with the *Mobile Agent* during the migration process.

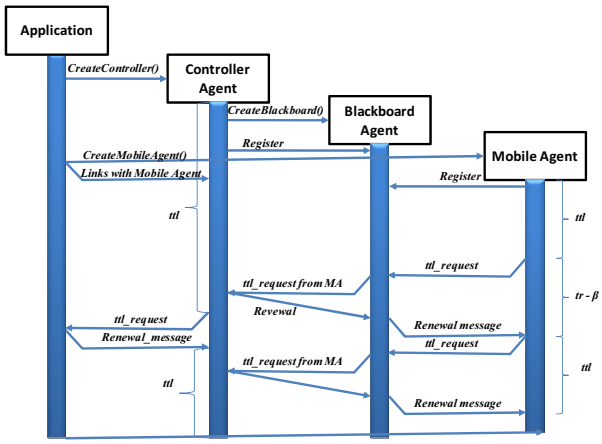

Figure 5. ttl renewal process

The *Controller Agents* also contains information about the last knowing location of the associated agents, this information is updated each time that these agents receive a *ttl_request* from a Mobile agent.

When the Application decides to *actively* eliminate an Agent, it uses the created path for locating the right agent and sends it the elimination message. Finally both *Application* and *Controller Agent*, erase all references to the target agent. When the target agent receives the *active termination_message*, it ends with its execution (see Figure 6).

The *Passive Elimination* is supported by the Orphan Agents control.

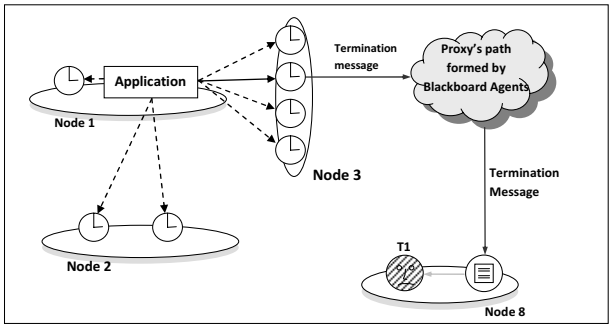

Figure 6. Active elimination of agents

III. Protocol Modeling

A. Stopwatch Petri Nets

The Stopwatch Petri Nets (SWPN) is an extension of Time Petri Nets (TPN) for addressing the modeling of the preemptive scheduling of tasks, i.e. actions that can be suspended and resumed [20], [21].

In this work we proposed an extension to TPN based on the SWPN defined in [20] and similar semantics of Stopwatch Automata [22] for allowing the use of Stopwatches (SW)

associated to several transitions, and the use of transitions constrains over the SW to restrict the firing of transitions.

Definition 1. A Stopwatch Petri Net as a tuple (P, T, Pre(.), Post(.), Mo, X, ϕ, G, INI), where:

- $P = \{p_1,p_2,...,p_n\}$ and $T = \{t_1,t_2,...,t_m\}$ are finite sets of vertices named places and transitions respectively
- *Pre (Post):* $P \times T \rightarrow \mathbb{Z}^+$ is a function representing the weighted arcs going from places to transitions (transitions to places); $\mathbb{Z}^+$ is the set of nonnegative integers.
- $Mo \in (\mathbb{Z}^+)^{|P|}$ is the initial marking of the net, associating with each place p the number of tokens it contains initially. $M(p_i)$ is the number of tokens in the place p_i.
- $X = \{x_i | i \in \mathbb{N}\}$ is a finite set of positive real-valued stopwatches, which has the following attributes:
  - $\dot{X} = \{(\dot{x}_i = c) | c \in \{0,1\}\}$ is a set of derivatives of the stopwatches with respect to time
  - $C(X) = \{(x_i \; op \; v_j) | \; op \in \{=,<,>,\leq,\geq\} \; and \; v_j \in \mathbb{R}^{\geq 0}\}$ is the set of constrains over X
- $\phi \; \phi : P \rightarrow 2^{\dot{X} \cup C(X)}$ is a function that associates to places a subset of derivatives of the stopwatches and watch constrains (invariants); in this way $\phi(p_i) = \{(\dot{x}_1 = c_1), \ldots, (\dot{x}_k = c_k), (x_1 \; op_1 \; v_1), \ldots, (x_j \; op_j \; v_j)\}$ where $p_i \in P$
- $G: T \rightarrow Exp_G(C(X))$ is a function that associates to transitions a Boolean expression (guards) defining the conditions that must be fulfilled to fire the transition.
- $INI : T \rightarrow Exp_{INI}(X)$ is a function that associates to transitions a set of expressions in the form $(x_i := v_j)$.

B. Transition enabling and firing

A transition $t_j \in T$ is said to be enabled by the marking M if $M(p_i) \geq Pre(p_i, t_j) \; \forall \; p_i \in P$ and the invariants in each $\phi(p_i)$ must be true. For example, in Figure 7, t_1 is enabled when both *P1* and *P4* are marked (when the invariant is omitted there is no time restrictions to keep the transition enabled).

If the time elapse defined by the guard of one transition t_j does not overlap with the time induced by the invariants of the input places of t_j then it never will be fired.

If the invariant of one place (p_i) is never true or is not longer true due to the over time, then the output transitions will never be enabled.

If t_i is enabled, then t_i may fire in M if the guard associated with t_i is fulfilled and the invariants in its input places are still true. For example, in Figure 7, none of the transitions are enabled until the communication place *P4* gets a mark.

When a transition t_i is fired the stopwatches expressed in the fired transition are initialized to the specified values. For example, in Figure 7, when transition t_2 is fired, the stopwatch x_1 is reset to 0. The firing of transitions is instantaneous.

C. Stopwatch handling

In a SWPN every stopwatch x_i may be interrupted($\dot{x}_i = 0$) and then resumed later($\dot{x}_i = 1$). This is declared for every place in the net.

The function ϕ associates a subset of stopwatch derivatives and a set of watch constrains to each place; this represents the active or interrupted stopwatches when the place is marked, and the time invariant where the transition is enabled. For example $\phi(p_1) = \{(\dot{x}_1 = 1), (x_1 \leq 6)\}$ means that when p_1 is marked, the stopwatch x_1 is running, and while its value is less or equal to 6, the transitions t_j in *Post(p_1, t_j)* can be enabled.

We assume that all stopwatches in the initial marking are stopped unless it is explicitly indicated in a marked place. Once the state of a stopwatch is updated it remains in the same state even if the derivative is not explicitly specified in some place. The stopwatch changes its state only when it is indicated in a marked place.

The function G declares some constrains to the firing of transitions, or gives a condition where the transition may fire. For example $G(t_1) = \{(x_1 == 6)\}$ means that when t_1 is enabled and the guard x_1==6 is true, t_1 can be fired. If a guard for a transition is omitted, then the transition can be fired immediately.

Consider the SWPN in Figure 7; it has a stopwatch x_1 that runs only when the token is either in *P2* or *P5*. When t_1 is fired, x_1 is reset and starts running. When t_3 is fired then x_1 is suspended, and then resumed after the firing of t_4; once x_1 is resumed it returns in the same value it had when it was stopped (see Figure 7b). In this way, each stopwatch is reset or initialized only in the initial marking or when it is specified in the firing of a transition, for example at the firing of t_2 in Figure 7a.

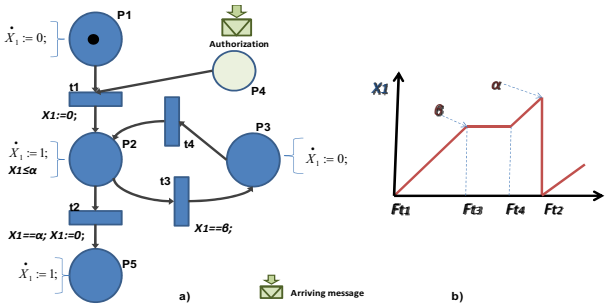

Figure 7. a) Example of a Stopwatch Petri Net supposing α>β, b) time scale of x_1 according to some transitions firings (F_{t_i})

D. Communication places

A special kind of place, named communication place, is included to represent the message exchange between agents. An output message in an agent PN model corresponds to an arriving message in other agent PN model. Having this couple of places is equivalent to have a simple place connecting two different nets as showed in Figure 8, where the dashed lines indicates that we can represent the lost of the message.

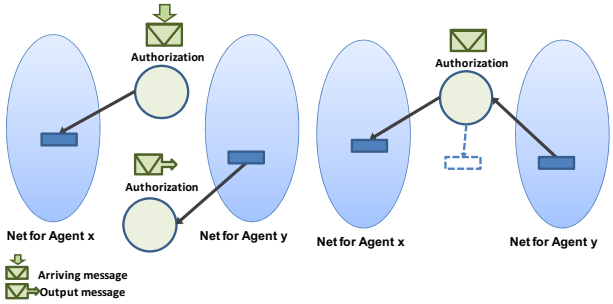

Figure 8. Message representation and its equivalence

E. Protocol Modeling

Now, two of the components of the MACOP protocol are presented.

In Figure 9 the behavior of the *Executor agent* is described. On the left side of the figure it is described the normal behavior of the agent, i.e. the tasks that the agent must execute to complete its goal. The right side describes the tasks regarding the agent control activities inherent to the MACOP protocol. When transition t_1 is fired, the two different behaviors of the agent are enabled. If *P4* is marked, the normal tasks of the agent are executed and during this time (stopwatch *X1*) the agent consumes its *ttl*; while t_8 is not enabled (it will be enabled when the *ttl* of the agent is *null*), the agent can migrate from one site of the net to another site and execute tasks.

If t_8 is enabled (*X2==ttl*), i.e. when the *ttl* of the agent is consumed, the activity in *P4* is interrupted after having started; in this way the clock *X1* is stopped representing that the agent is blocked while is renewing its *ttl*. If the request for a new *ttl* is answered (*P10* is marked), the tasks of the agent and the clock *X1* are resumed. If after *tr* units of time the agent has not received the *New_ttl* message (when clock *X3==tr*) then the agent ends its execution (t_{12} is fired).

Due that we are dealing with reactive agents, several communication places are included in the model. These places correspond to that included, also as communication places, in the model of the *Blackboard Agent* showed in Figure 10. Notice that that model does not require the use of stopwatches.

The *controller agent* has a model similar to the executor agent, because both agents perform the process of *ttl* renewal. Furthermore the model must include the process of authorize/reject *ttl* renewals from the associated agents. This agent communicates with the *blackboard agent;* so, the model includes communication places (Figure 10).

Finally, the model for the application may only include the most important tasks, such as agent creation, agent association, the periodical updating of the list of agents that must continue alive, and the termination of agents.

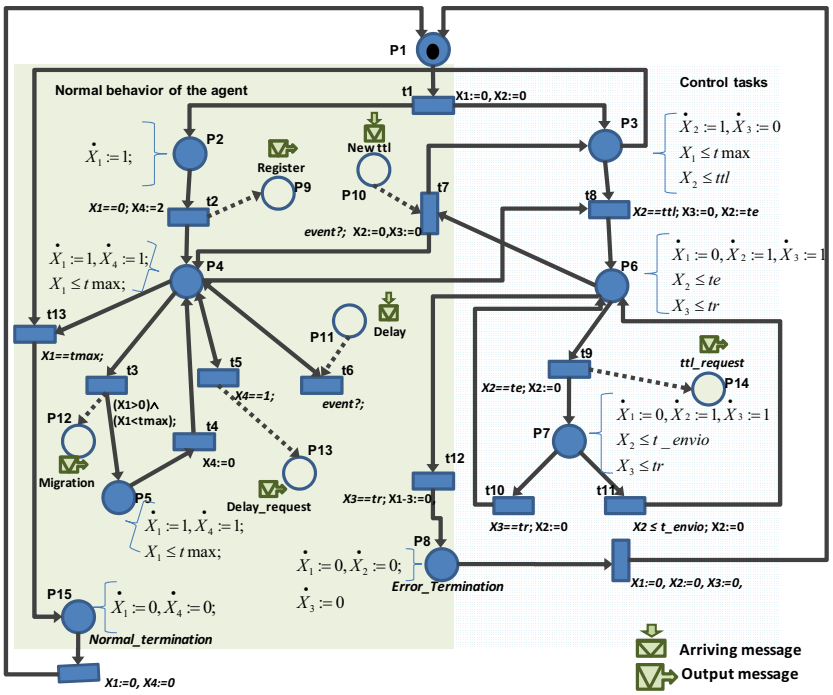

Figure 9. SWPN of the Executor Agent

F. *Modeling issues using SWPN*

The modeling using SWPN allows defining easier and formally important features of the protocol such as the preemptive behavior of agents' tasks. A SWPN model helps to analyze the timing constraints of interactive tasks performed by the protocol components.

During the model building, stopwatches, invariants and guards must be carefully defined. On one hand on must to observe the places that have associated the same stopwatch; in general such places must be exclusively marked. On the other hand invariant and guards have to reasonably overlap.

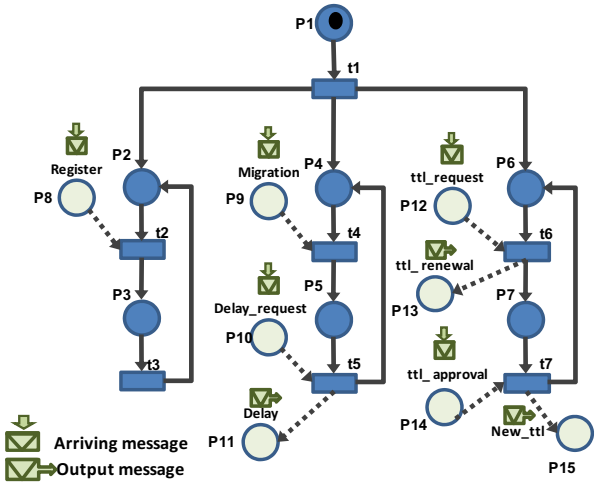

Figure 10: PN for the Blackboard Agent

IV. CONCLUSIONS

The proposed protocol is based in the path concept and in the energy concept applied to the mobile agent paradigm. The time control is supported by assigning a quantum of life to each mobile agent since its creation, which is consumed during the agent activity and renewed when it is totally consumed. The protocol also uses a special agent called *Blackboard Agent* to record information about the target agent location that is left behind whenever t he agent migrates from one host to another. The records where used for agent location. Some of the advantages provided for this proposed protocol include: the passive and active termination of agents, location of agents at any time, the detection of orphan agents in a local manner. The modeling using SWPN allows a clear understanding of the protocols and makes easier the process of analysis of properties.

ACKNOWLEDGMENT

M. Flores-Badillo was sponsored by CONACYT, Grant No. 191083

REFERENCES

[1] W.M.P. van der Aalst, and K. Hee, "Workflow Management: Models, Methods and Systems". London, MIT Press. 2002

[2] H.A. Reijers, and W.M.P. van der Aalst, "The Effectiveness of Workflow Management Systems: Predictions and Lessons Learned". International Journal of Information Management, Vol. 25, No. 5. pp. 458-472. 2005.

[3] M. Repetto, M. Paolucci, and A. Boccalatte, "A design tool to Develop Agent-Based Workflow Management Systems", Proc. Italian Workshop, from Objects to Agents: Intelligent Systems and Pervasive Computing (WOA2003), Villasimius, Italy. 2003

[4] B.T.R. Savarimuthu, and M. Purvis, "A Collaborative Multi-Agent Based Workflow System", Knowledge-Based Intelligent Information and Engineering Systems, Lecture Notes in Artificial Intelligence (LNAI), vol. 3214, 2004.

[5] M. Flores-Badillo, and E. López-Mellado, "Mobile Agent based Automation of Distributed Workflow Processes", in Proc. of the IEEE International Conference on System of Systems Engineering, (SOSE'08) June 2nd - 4th, 2008, Monterey, California, USA.

[6] L. Xu, H. Liu, S. Wang and K. Wang, "Modelling and analysis techniques for cross-organizational workflow systems", Systems Research and Behavioral Science Volume 26, Issue 3, Pages367 – 389. 2009.

[7] L Xu, WA Tan, H Zhen and W Shen. "An approach to enterprise process dynamic modeling supporting enterprise process evolution", Information Systems Frontiers, Vol. 10, Issue 5, pp. 611-624, November 2008.

[8] E. Chen Wen-Shyen, R. Leng Chun-Wu, and Lien Yao-Nan, "A novel mobile agent search algorithm", in Proc. of the First Int. Workshop on Mobile Agents '97, Lecture Notes in Computer Science 1219, Springer-Verlag, Berlin, Germany, pp. 162 - 173. 1997.

[9] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, et al, "MASIF: the OMG mobile agent system interoperability facility", in Proceedings of the First Int.Workshop on Mobile Agents '98.1998.

[10] J. Baumann, "Control Algorithms for Mobile Agents", tech. report of Stuttgart University,1999, pp.59-79.

[11] E. Jochum, "Konzeption und Implementierung eines Orphan-Detection Mechanismus nach dem Energiekonzept", Student Thesis Nr. 1642, Faculty of Computer Science, University of Stuttgart. 1997.

[12] "Mole Project Pages" (1999), University of Stuttgart, web page, URL: http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.htm

[13] R. J. Fowler, "Decentralized object finding using forwarding addresses", Ph.D. Thesis, Technical Report 85-12-1, Department of Computer Science, University of Washington, USA. 1985.

[14] A. Black, N. Hutchinson, E. Jul, H. Levy, and L. Carter, "Distribution and abstract types in Emerald", IEEE Transactions on Software Engineering 13, 1, 1986. pp. 65 - 76.

[15] E. Jul, H. Levy, N. Hutchinson, and A. Black, "Fine-grained mobility in the Emerald system", in ACM Transactions on Computer Systems 6, 1, 1988. pp. 109 - 133.

[16] B. Awerbuch, and D. Peleg, "Concurrent online tracking of mobile users", Journal of the ACM 42, 5, 1995. pp. 221 – 233

[17] J. Baumann, "A protocol for orphan detection and termination in mobile agent systems", Technical Report Nr. 1997/09, Faculty of Computer Science, University of Stuttgart, Germany. 1997.

[18] A. Padilla-Duarte, "A Mobile Agent Population Control Protocol" (in Spanish), MSc. Thesis, Cinvestav Guadalajara Mexico. December 2007.

[19] G. Cabri, L. Leonardi, and F. Zambonelli, "Coordination infrastructures for mobile agents", Microprocesors and Microsystems 25 2001. pp. 85-92.

[20] M. Magnin, P. Molinaro, and O. Roux, "Decidability, Expressivity and State-Space Computation of Stopwatch Petri Nets with Discrete-time Semantics", in Proc. of the 8th International Workshop on Discrete Event Systems, Ann Arbor, Michigan, USA, July 2006.

[21] A. Allahham, and H. Alla, "Post and Pre-initialized Stopwatch Petri Nets", In Proc. of the 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07), France, 2007.

[22] F. Cassez, and O. Roux, "Structural Translation from Time Petri Nets to Timed Automata", Journal of Systems and Software, 79(10), 2006.