INT J COMPUT COMMUN, ISSN 1841-9836
9(3):292-304, June, 2014.

Formal Specification and Verification of Mobile Agent Systems

L. Kahloul, M. Grira

Laid Kahloul*
LINFI Laboratory, Computer Science Department,
University of Biskra, Biskra, 07000, Algeria
*Corresponding author: kahloul2006@yahoo.fr

Messaouda Grira
Computer Science Department, University of Biskra
Biskra, 07000, Algeria

Abstract: Mobile agent systems offer efficiency and flexibility as a design paradigm. These two characteristics allow to these systems to be an adequate solution for many problems. These systems are used in many critical domains. This expansion, in use, obliges designers to insure the reliability and correctness of such systems. Formal methods can be used to verify the correctness of these systems. This paper presents a formal specification and verification of mobile agent systems using the High Order π-calculus. The verification exploits the two tools UPPAAL and SPIN.
Keywords: Mobile Agent, Formal Verification, π-calculus, Promela, SPIN, UPPAAL.

1 Introduction

Formal methods are methods with mathematical background. Unlike informal methods, descriptions using formal methods are more precise, understandable, and unambiguous. There are many formal languages and methods, among these languages we recall: TL (Temporal Logics) [2], Processes Algebras (CCS [1], π-calculus [4], HOπ-calculus [19]) and State Transition Systems [3, 4]. To be formal, a language expressing a specification must comprise three components: a syntax defining the rules for forming expressions, a semantics with rules for the interpretation of formed sentences and a proof theory governing rules for inferring useful information from the specification [3].

π-calculus is a mathematical model of processes whose interconnections change as they interact. The basic computational step is the transfer of the communication link between two processes; the recipient can then use this link for further interaction with other parties. This makes the calculus suitable for modelling systems where the accessible resources vary over time [1]. Higher-Order-π-calculus (HOπ) treats another kind of mobility, where processes (called agents) themselves move. The π-calculus and its extensions are processes-algebras that focus on mobility. In these calculi, processes communicate using channels. A process sends a channel's name in the monadic π-calculus, tuples of channels names in the polyadic π-calculus, and tuples of processes and channels names in Higher-Order-π-calculus (HOπ) [4].

In software engineering, formal specification is the expression in a formal language, and at an abstract level, a set of proprieties; that a system is designed to satisfy [3]. It is recognised that good specifications should be adequate, internally consistent, unambiguous, complete, minimal and satisfied by lower-level ones [2].

System Verification is a domain appeared after the crisis witness of enterprises specialised in the development of systems, where the verification is a viable solution to these problems. During this period, verification tools were designed to verify systems. Some of the most important verification tools are:

1. UPPAAL: is the acronym of the university of Uppsala (Sweden) and Aalborg (Denmark). It is an educational tool for formal specifications and verification of systems [5];

2. SPIN: is the acronym of Simple Promela Interpreter. It is a tool for verifying the logical consistency of concurrent systems, specifically of data communication protocol. The verified systems must be described in the Promela [6] modelling language;

3. LOTOS: is the acronym of (Language of Temporal Ordering Specifications). LOTOS has been applied to describe complex systems, formally. A number of tools have been developed for LOTOS, covering user needs in the areas of simulation, compilation, test generation, and formal verification [7].

The formal specification and verification of mobile agent systems contribute to the best formalisation of these systems. Specification can be used to analyse some known MAS (mobile agent system) properties, which are: safety, accessibility, boundedness, and liveness. The objective of our work is to specify and verify an example of a MAS. This system manages service locations in particular networks. This system is called Service Location Protocol (SLP) [26].

To present this work, we organize this paper as follows: In the next section, we present briefly mobile agent systems, the section three will present HOπ-calculus, Promela and UPPAAL languages. Section four will present the modelling and verification using SPIN and UPPAAL tools. Before concluding this paper, we will show some related works in section five. Finally, a conclusion will summarize this paper and will discuss possible prospects.

2 Mobile Agent Systems (MAS)

2.1 Agent Concept

Agent's term comes from two distinct domains: *the distributed systems domain* and the *multi-agent systems domain.* These last belongs in the base to intelligence artificial domain, where the programmers tent to imitate humane intelligence [8]. Russel and Norvig define the agent term as entity that broach its environment and interact on it [9].

2.2 Mobile Agents

Mobile agents have been introduced initially in 1994 with the Telescript environment [16] that permitted to processes to choose themselves to move on the sites of a network in order to work locally onto resources. A mobile agent [12,13,15] is a process that can move from a site to another site in order to achieve a task. In general, the mobility is provided using some primitive like: **move(site)** that permits the agent to move toward the site designated by the parameter. A mobile agent is composed of his corresponding code, as well as of a context including some data. This context can evolve under execution, for example while collecting some data when an agent achieves a research of information on a set of servers. The code and the agent's context are displaced with the agent when this one visits different servers. In general, a mobile agent system provides the primitive of communication allowing to the agents to interact between them, but also to the agents to interact with the services that they visit. These primitives of communications take the form of sending messages or calling procedures or methods [17].

2.3 Mobility mechanisms

Mobile agents can move between network hosts, transporting their code, data and state information to continue their execution on different environments. In literature [5], we can find

many mechanisms. In the case of remote execution, the agent is sent before it starts to be executed. When it arrives on the destination, it is executed until it finishes. In this case the agent is transferred once. When it is executing it can use the same remote execution mechanism to start the execution of other agents. In the remote execution the destination of the agent is determined by the execution starter. The mobile agent can do a weak migration by sending its data with its code. Usually, the implementations of this scheme allow choosing which part of data will be transferred to the new location of the agent. In this case, the agent programmer might design some mechanism based on the value of agent's data to resume the execution from some point. Strong migration is the highest degree of mobility. Using this scheme not only agent code and data is sent, but also the state of execution. When the agent arrives to the destination, it is fully restored and its execution is resumed from the same execution point it was just before migration. Strong migration turns to be complex, since it involves low level internal mechanisms for execution restoring that must be standard to provide migration transparency in heterogeneous environments.

2.4 Applications and limits of mobile agents

Mobile agents [1] are software abstractions that can migrate across the network representing users in various tasks. This is a contentious topic [14, 18] that attracts some researchers. Mobile agents provide a very appealing, intuitive, and apparently simple abstraction. The authors in [1] give some applications of mobile agents:

1. Distributed research of information;
2. Active Documents: Active e-mail, web page (hypertext);
3. Advanced telecommunications services: video-conferencing, mobile users (with the potential disconnections);
4. Monitoring and remote configuration of devices: industrial processes, network management;
5. Management and cooperation in the work-flow: the work-flow defines activities, sites, relationships, time for their implementation, to achieve an industrial product. Mobile agents are responsible for conveying information between co-workers in a work-flow;
6. Active networks: flexible and dynamic networks according to application needs. Two approaches are proposed: (i) Programmable switches: dynamically extend the networks. This approach is based on Code on Demand paradigm; and (ii) Capsule approach: attach codes to the transferred packets. The node that receives the packet performs the associated code to treat the data in the packet;
7. E-commerce: an agent looks in a market for catalogues, and then it returns to the laptop of a customer with the best rates available;
8. Applications deployment and maintenance of components in distributed environments.
9. Parallel processing: the agents dispatch several computing units to make parallel certain tasks.

3 Formal Languages for MAS

We are interested to present two formal languages in this section HOπ-calculus, Promela language and two tools which are SPIN and UPPAAL.

3.1 HOπ-calculus

In this higher order paradigm, mobility is achieved by allowing agent to be passed as values in a communication. HOπ-calculus [19] is an extension of the first order π-calculus introduced by D.Sangiorgi [4]. This calculus enriches the π-calculus with an explicit higher order communication. In the HOπ-calculus, not only names, but also agents of arbitrarily high order, can be transmitted. The syntax of the HOπ-calculus is an extension of the syntax of the first order π-calculus.

Let be $\{a, b, ..., x, y, ...\}$ a set of names and $\{P, Q, ...\}$ a set of processes [19]. Processes of HOπ-calculus are defined by the following grammar:

$P ::= \tilde{x}(K).P|x(U).P|P||Q|\tau.P|\upsilon xP|P + Q|[x == y]P|!P|0$

Where:

1. The output prefix $\tilde{x}(K).P$ can send the name K via the name x and continue as P;
2. The input prefix $x(U).P$ can receive any name via x and continue as P with the received name substituted for all free occurrences of U in P;
3. The Parallel Composition $P||Q$ represents the combined behaviour of P and Q executed in parallel, where P and Q can proceed independently and interact via shared names;
4. The Silent Prefix $\tau.P$ represents an agent that can evolve to P without interaction with the environment;
5. The Restriction υxP behaves as P but the scope of the name x is restricted to P. P can not interact with other process through x;
6. The Sum $P + Q$ represents an agent that can enact either P or Q;
7. The Match $[x == y]P$ can evolve as P if x and y are the same name, and will do nothing otherwise;
8. The Replication $!P$ can be seen as an infinite composition $!P = P|P|$ or, equivalently, an agent satisfying the equation $!P = P|!P = !P|P$;
9. The empty agent 0 cannot perform any action.

3.2 Promela

Promela [6] is the acronym of Process (or Protocol) Meta Language; it is the modelling language for the SPIN (Simple Promela Interpreter) [27]. Promela programs consist of processes, message channels, and variables. Processes are global objects that represent the concurrent entities of the distributed system. Message channels and variables can be declared either globally or locally in a process. It supports rendezvous and asynchronous communication between processes via channels. Processes specify behaviour, while channels and global variables define the environment in which the processes run.

3.3 UPPAAL

UPPAAL [6] is a tool to model, to validate and to verify real time systems; it is appropriated for the systems that can be modelled by timed automata or linear hybrid automata. The model-checker UPPAAL is based on the theory of timed automata [29,30] and its modelling language offers additional features such as bounded integer variables and urgency (priority between the same actions defined in two automata). The query language of UPPAAL, used to specify properties to be checked, is a subset of CTL (computation tree logic) [31].

4 Specification and Verification of the SLP Protocol

4.1 The SLP Protocol

The Service Location Protocol is an IETF standard track protocol [26]. The IETF is the Internet Engineering Task Force committee that is part of the IAB (Internet Activities Board) and determines internet standards. SLP provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks. SLP can eliminate the need for the user to know the technical features of network hosts. Using the SLP, the user needs only to know the description of the service he is interested in. Based on this description, SLP is then able to return the URL of the desired service. SLP is a language independent protocol. Thus the protocol specification can be implemented in any language. The SLP infrastructure (Figure 1) consists of four types of agents [26]:

1. User Agent (UA), which is a software entity that is looking for the location of one or more services;

2. Service Agent (SA), which is a software entity that provides the location of one or more services;

3. Directory Agent (DA), which is a software entity that acts as a centralised repository for service location information;

4. Memory Directory Agent (MDA), which is a software entity that memorizes the service information.

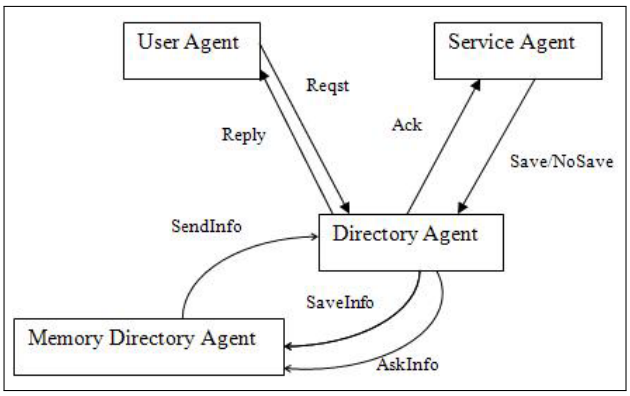

Figure 1: The SLP infrastructure

In next section, we present a formal specification of SLP in HOπ-calculus. This specification will be transformed firstly into the Promela language, then into the UPPAAL modelling language. The section six will present the verification phase. We achieve the verification using the two tools: SPIN and UPPAAL, then we discuss a comparison between the results obtained from the two tools.

5 SLP Specification

In this section, we will specify the SLP protocol by three formal specification languages: the HOπ-calculus, Promela, and UPPAAL modeling language. The formal specification of the SLP

system using the HOπ-calculus is given as the following:

$$
\begin{array}{l}
system = (Reqst,\ Reply,\ Save,\ NoSave,\ Connect,\ Ack,\ Ask_{info},\ Send_{info})| \\
UA(Reqst,\ Reply)|SA(Connect,\ Save,\ NoSave,\ Ack)| \\
DA(Reqst,\ Ack,\ Connect,\ Save,\ NoSave,\ Ask_{info},\ Send_{info})| \\
MDA(Ask_{info}, Send_{info})
\end{array}
$$

In this specification, the system is composed of one User Agent (UA), one Directory Agent (DA), one Memory Directory Agent (MDA), and one Services Agent (SA). The mobility is described through the parameters of the channel. For each new service registered by the SA within the DA, there are MDA belonging to this specific service. In our case the service will be **service**. The parameters for each agent are used for the communication between agents.
In the UA specification, we use **Request** and **Reply** as two channels for the communication between the UA and DA. The term **service** specifies the service that will be asked. In this specification, the UA sends a request to ask a service, and receives a reply that contains the service information. The User Agent (UA) Specification is as follows:

$$
\begin{array}{l}
UA(Reqst,\ Reply) = \upsilon(service) \\
\overline{Reqst}(service, Reply).Reply(service, info)
\end{array}
$$

The SA can register a service within the DA using the channel $Save$ and also remove a registered service within the DA using the channel $NoSave$, and it receives an Ack as an ok to $Save$ or $NoSave$. The Service Agent (SA) Specification is as follows:

$$
\begin{array}{l}
SA(Connect, Save, NoSave, Ack) = \upsilon(service) \\
\overline{Connect}(adrs).\overline{Save}(service).Ack|\ SA(Connect,\ Save,\ NoSave,\ Ack) \\
+\overline{Connect}(adrs).\overline{NoSave}(service).Ack|\ SA(Connect,\ Save,\ NoSave,\ Ack)
\end{array}
$$

The DA can communicate with UA and SA, it generates the service location. This DA saves the service information if it receives a $Save(service)$, and does not save it otherwise. If the DA receives a $Reqst(service)$, it asks the Memory Directory Agent (MDA) for the service information via $Ask_{info}(service)$, and it waits the reply $(Send_{info}(service))$ from this agent, before sending this service information to the UA via $\tilde{Reply}(info, service)$. The specification of the DA is as follows:

$$
\begin{array}{l}
DA(Reqst,\ Ack,\ Connect,\ Save,\ NoSave,\ Save_{service},\ NoSave_{service},\ Ask_{info}, \\
Send_{info}) = Connect(adrs)|Save(Service).\overline{Ack}.DA_{Mem}(Ask_{info}, send_{info})| \\
\overline{Save}_{service}(service)|\ Reqst(service).\overline{Ask_{info}}(service)| \\
Send_{info}(service, info).\overline{Reply}(service, info)| \\
DA(Reqst, Ack, Connect, Save, NoSave) \\
+Connect(adrs)|NoSave(service).\overline{Ack}| \\
DA_{Mem}(Ask_{info}, Send_{info}).\overline{NoSave}_{service}(service)| \\
DA(Reqst, Ack, Connect, Save, NoSave)
\end{array}
$$

The MDA is the Memory Directory Agent; it saves information about services. It communicates only with the DA. The specification of the MDA is as follows:

$$
\begin{array}{l}
MDA(Ask_{info}, Send_{info}, Save_{service}, \overline{NoSaveservice}) = \\
Save_{service}(service)|Ask_{info}(service).\overline{send}_{info}(service)|MDA(Ask_{info}, Send_{info}) \\
+NoSave_{service}(service).MDA(Ask_{info}, Send_{info})
\end{array}
$$

Formal methods may be used to specify the system behaviour and to verify that the designed and implemented system satisfies the expected properties. The higher-order π-calculus is able to describe the mobile systems behaviour and, because it possesses formal semantics, it is capable of verification as well. To insure verification of the HOπ-calculus, one is obliged to use some automated tools. We have found two tools that can be used: SPIN and UPPAAL. To realise verification using this two tools, we were obliged to apply some transformation on the former specification. The two next sections will present the specification in SPIN and in UPPAAL.

5.1 Modelling SLP Using Promela

To allow the verification of the SLP protocol, we propose (firstly) to use the SPIN tool. The SPIN tool uses Promela as a specification language. Requirements and properties are specified as LTL (Linear Temporal Logic) [2]. These LTL formulae can be entered as assertions in the Promela specification, and so be used in simulation, or they can be verified using the model-checker of SPIN. The following paragraphs present the Promela specification of the SLP protocol.

The messages declarations are formalised as follows:

$$\#define\, msgtype\, 10$$
$$mtype = service, name, ok, info$$

The channels between the entities (agents) are formalised as follows:

$$chan\, Reqst = [1]\, of\, mtype, byte$$
$$chan\, Reply = [1]\, of\, mtype$$
$$chan\, Save = [1]\, of\, mtype$$
$$chan\, NoSave = [1]\, of\, mtype$$
$$chan\, Ack = [1]\, of\, mtype$$
$$chan\, Connect = [1]\, of\, byte$$
$$chan\, Ask_info = [1]\, of\, mtype$$
$$chan\, Send_info = [1]\, of\, mtype$$
$$chan\, Save_service = [1]\, of\, mtype$$
$$chan\, wait = [1]\, of\, byte$$
$$bool free = true,\ souscribe = false;$$
$$byte adrs;$$

We declare four processes. The first one is a process named *User Agent*, specified as follows:

$$active\, proctype\, UA()\{$$
$$xsReqst;\ xrReply;$$
$$again1:$$
$$Reqst!service,\ Reply;$$
$$if\ \ :: Reply?service \rightarrow free = false;\ wait!msgtype(10)$$
$$fi;$$
$$goto\, again1\}$$

The second one is a process named *Service Agent*, specified as follows:

```
active proctype SA(){
xs Save, NoSave; xr Ack;
again2;
Connect!adrs;
if  :: wait?(10) → free = false
fi
if
:: free == true → Save!service
:: else → NoSave!service
fi;
Ack?ok;
goto again2}
```

The third one is a process named *Directory Agent*, specified as follows:

```
active proctype DA(){
xs Ack, Ask_info, Reply;
xr Save, NoSave, Send_info, Reqst;
if  :: Connect?adrs → subscribe = true
fi
assert((free == true) & &(subscribe == true));
if
:: Save?service → Ack!ok; Save_service!service
:: NoSave?service → Ack!ok
:: Reqst?service → Ask_info!service
fi;
if Send_info?service → Reply!service, info
fi}
```

The four one is a process named *Memory Directory Agents* and it is specified as follows:

```
active proctype MDA(){
xs Send_info; xr Ask_info, Save_service;
if
:: Save_service?service;
:: Ask_info?service → Send_info!service
fi}
```

5.2 Modelling SLP by UPPAAL

Figure 2 presents the graphic modelling of the whole system specification as a timed automaton. The agents SA, DA, and MDA are presented respectively in the Figures: 3, 4, and 5.

6 SLP Verification

This section is dedicated to present the verification phase in this work. We consider three important properties (i.e. requirements), that we will verify:

- **Safety**: Every agent SA which registers a service within DA is recognised by the DA and the service becomes available;

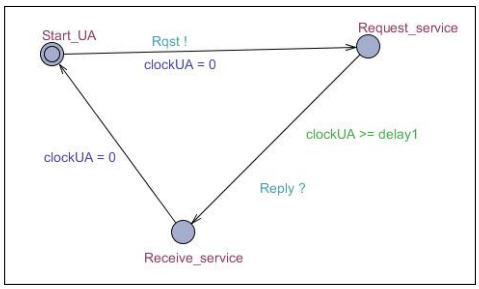

Figure 2: Diagram of the system

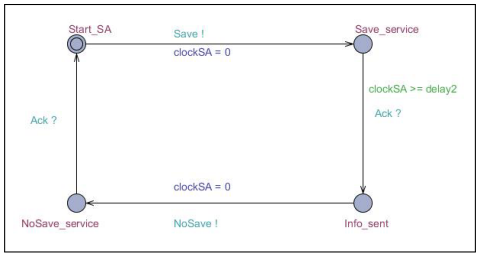

Figure 3: Service agent diagram

- **Reachability**: If UA asks for the service "service" and if SA subscribes this service at the DA, then UA obtains the service;

- **Bounded Liveness**: During the registration period within the DA, a service must be accessible.

6.1 SLP Verification by the SPIN Tool

We use the SPIN tool to verify the Safety, Reachability and Liveness properties. This tool allows verifying properties during the simulation using the assert function (assertion): $assert((free == true)\ \&\ \&(subscribe == true))$, or using the following never claims:

$never\{$
$TO\_init:$
if
$::(!(ask)\ \&\ \&(frr)\ \&\ \&(subscribe)) \rightarrow goto\, accept\_all$
$::(!ask) \rightarrow goto\, TO\_S3$
$fi;$
TO_S3
if
$((free)\ \&\ \&(subscribe) \rightarrow goto\, accept\_all$
$(1) \rightarrow goto\, TO\_S3$
$fi;$
$accept\_all$
$skip\}$

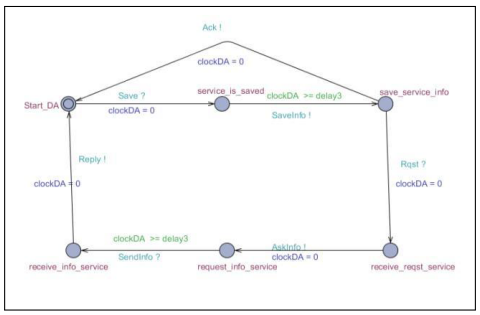

Figure 4: Directory agent diagram

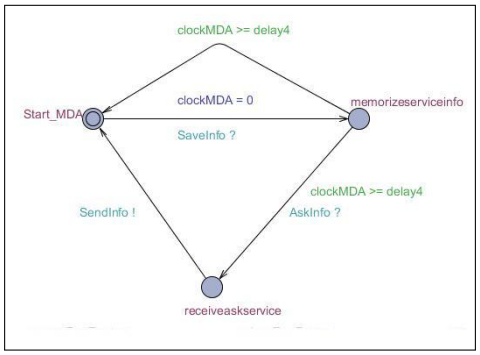

Figure 5: Memory directory agent diagram

6.2 SLP Verification by the UPPAAL Tool

We use the SPIN tool to verify the Safety and Liveness properties using the CTL (Computational Tree Logic) [31] as following:

$E<>Directory.receive\_reqst\_service$: This expression allows us to verify the reachability of some sate in the directory agent behaviour.

$A[]notdeadlock$: This expression allows us to verify that the system has no deadlock.

7 Related Works

Mobile agent paradigm attracts researchers in the domain of software engineering and artificial intelligence. Many interesting researches on formal specification and verification of mobile code system can be found. In the present work, we have presented firstly a specification of the service location protocol system in the HOΠ-calculus. This specification was then verified using the two tools: SPIN and UPPAAL. This verification validates the safety propriety. In the literature, we found some similar works that tended to specify and verify formally mobile code systems using other specification languages and other verification tools, as examples:

- In [1], the SLP is specified in HOPi-calculus and verified by UPPAAL tool. In this work, the HOPi-calculus specification is then translated to an automaton model, where the mobile aspect of data is not respected. In our work, we have used a specification with only four agents. Another difference is that we have used the SPIN tool and Promela.

- In [3], a formal specification and verification of secure routing protocols for mobile ad hoc networks (MANETs) is presented. In this work, authors used the Promela as specification languages and SPIN as verification tool.

- Formal specification of standards for distance victor routing protocol is presented in [10]. This project presents how to use an interactive, HOL (High Order Logic), together with a model checker, SPIN, to prove key proprieties of distance victor routing protocols. The formal verification techniques in this project are suited to routing protocol generally.

8 Conclusion

A mobile agent is an entity that can change its location during its execution. It can migrate from one host to another, in a network. The use of mobile agents knows an expansion in several domains. This inclusion of mobile agents in critical domains obliges the designer and developer to make attention during the elaboration of these agents. The reliability and the correctness of these programs (mobile agents) are important. Formal methods can help the designer to specify systems then to prove the correctness of these systems. Some of these methods are: π-calculus, HOπ-calculus.

In this paper, we have presented the use of the HOπ-calculus and Promela language as specification languages for mobile agent systems. We have presented a specification of the SLP protocol (Services Location Protocol), and we have done verification using the two tools: UPPAAL and SPIN.

The use of formal methods meets some difficulties in software development. These difficulties are argued by the lack in experiments of these methods and the requirement of some mathematical background by the developer. In this work and some other previous works [32–34], we have tried to show the power of formal methods to insure the reliability of a mobile agent systems. In our future work, we will be interested to apply formal methods on more examples of mobile agent systems. These experiments will establish an approach on the use of formal methods, in software engineering process.

Bibliography

[1] Milner, R. (1989). *Communication and Concurrency*, Prentice Hall, International Series in Computer Science, ISBN 0-13-115007-3.

[2] Rescher,N., Garson, J. (1968). Topological Logic, *Journal of Symbolic Logic*, 33(4):537-548.

[3] Wagner, F. (2006). *Modeling Software with Finite State Machines: A Practical Approach*, Auerbach Publications, ISBN 0-8493-8086-3.

[4] Murata, T. (1989). Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, 77(4): 541-580.

[5] Fuggetta, A., Picco, G.P., and Vigna, G. (1998). Understanding Code Mobility, *IEEE transactions on software engineering*, 24(5): 342-361.

[6] Bengtsson, J., Larsen, k.G., Larsson,F.,Pettersson, P., Yi W. (1995). Uppaal - a Tool Suite for Automatic Verification of Real-Time Systems, in *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, 22-24 October, 1995.

[7] Barbu, A. (2005). *Developing Mobiles Agents Through a Formal Approach*, Thesis, Paris XII, 12 September 2005.

[8] Lamsweerd, A. V. (2000). Formal Specification: a Roadmap, *ICSE '00 Proceedings of the Conference on The Future of Software Engineering*, 147-159.

[9] Gurdag, A.B., Caglayan, M.U. (2008). A Formal Security Analysis of SAODV using Model Checking, *International Symposium on Computer Networks (ISCN)*, June 2008.

[10] Sangiorgi, D., Walker, D.(2003). *The Pi-calculus: A Theory of Mobile Process*, Cambridge University Press.

[11] Faez CHARFI. (2003). *Une approche dinterfaage de CoD UPPAAL pour la spcification et la vrification des sysmes temps el*, Thesis, September 2003.

[12] Ruys, T. (2006). *SPIN and Promela*, January 18, 2006.

[13] Sighireanu, M., *LOTOS NT User Manual*, February 21, 2008.

[14] Gray,R.S., Cybenko,G., Kotz, D., Peterson, R.A., Rus, D. (2002) *D'agents: Application and performance of a mobile-agent system*, 2002.

[15] Akhil Sahai, A., Morin, C. (1998). Mobile agents for enabling mobile user aware application, *Proceedings of the 2nd International Conference on Autonomous agents*, 205-211.

[16] Bhargavan,K., Obradovic, D., Gunter, C.A. (2002). Formal Verification of Standards for Distance Vector, *Journal of the ACM*, Vol. 49(4):538-576.

[17] Picco, G.P. (1998). *Understanding, Evaluating, Formalizing, and Exploiting Code Mobility*, Ph.D. Thesis, Politecnico di Torino, Italy, February 1998.

[18] Satoh, I. (2002). Physical mobility and Logical mobility in ubiquitous computing environments, *Proceeding MA '02 Proceedings of the 6th International Conference on Mobile Agents*, 186-202.

[19] Johansen, D. (2004). Mobile agents: Right concept, wrong approach, *Proc. IEEE, Mobile Data Management-MDM*, 300-301.

[20] Robles S. (2002). *Mobile Agent Systems and Trust, a Combined View Toward Secure Sea-of-Data Applications*, PhD Thesis, Barcelona, July 2002.

[21] Kotz, D., Gray, R. S. (1999). Mobile agents and the Future of internet, *ACM Operating Systems Review*, 7-13.

[22] Roth, V. (2004). Obstacles to adoption of mobile agents, *IEEE International Conference on Mobile Data Management*, 296-297.

[23] Milojicic, D. (1999). Trend Wars: Mobile agent applications. *IEEE Concurrency*, 80-90.

[24] Sangiorgi, D. (1993). From π-calculus to Higher-Order-π-calculus- and back. *TAPSOFT '93 Proceedings of the International Joint Conference CAAP/FASE on Theory and Practice of Software Development.*

[25] Milner, R., Parrow, J., Walker, D. (1992). *A calculus of mobile processes*, part I/II. 1992.

[26] Perkins, C., Guttman, E., *Service Location Protocol (SLP)*, online: http://www.ietf.org/rfc/rfc2608.txt.

[27] Kaliappan, P.S., *Simple Promela Interpretor (SPIN)- Model Checker.*

[28] Guillaume, S. (2008). *A Promela front-end for Spot*, 20 p., May 2008.

[29] Alur R., Dill, D.L. (1990). Automata for modeling real-time systems, *Colloquium on Algorithms, Languages, and Programming*, 443:322-335.

[30] Hopcroft, J.E., Ullman, J.D. (2001). *Introduction of Automata Theory, Languages*, Addison Wesley.

[31] Huth, M., Ryan, M. (2004). *Logic in Computer Science*, (Second Edition), Cambridge University Press. p. 207. ISBN 0-521-54310-X. 2004.

[32] Kahloul, L., Chaoui, A. (2008). Coloured Reconfigurable Nets for Code Mobility Modeling, *Int J Comput Commun*, ISSN 1841-9836, Suppl. issue, 3(S):358-363.

[33] Kahloul, L., Chaoui, A., Djouani, K. (2009). *Code Mobility Modelling: A Formal Study*, *International Review on Computer and Software*, http://www.praiseworthyprize.com/IRECOS.htm.

[34] Kahloul, L., Chaoui, A., Djouani, K. (2010). Modeling Reconfirgurable Systems Using Flexible Petri Nets, *4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, August 24 - 27, 2010, Taipei, Taiwan.