# A Contract-based Approach for Secure Service Discovery Systems

Ahmed Nait-Sidi-Moh, Mahmed Bakhouya, Maxime Wack

# A Contract-based Approach for Secure Service Discovery Systems

A.Nait-Sidi-Moh[1], M. Bakhouya[2], M. Wack[1]

[1]Université de Technologie de Belfort-Montbéliard, 90010 Belfort, France
ahmed.nait@utbm.fr, maxime.wack@utbm.fr
[2]High Performance Computing Laboratory, George Washington University
bakhouya@gwu.edu

*Abstract*—**Future distributed computing systems will be ubiquitous and provide accesses to a wide range of services at any time, every where, and from a variety of devices. The monitoring and controlling accesses to services are one of the fundamental challenges that must be faced in the context of ubiquitous and pervasive environments. To address services accesses issues, it is necessary to guaranty the user access to utilize the allowed and available services. More precisely, an access control must be granted to services in order to regulate their usage. In this paper, a contract-based approach for monitoring and controlling accesses to services in ubiquitous environments is presented. In this approach, client agents and server agents are bounded by contracts: the server agent expects that the client agent will not attempt to access other services than those it required, and the client agent expects that the server agent will provide the required service.**

## I. INTRODUCTION

Service discovery and access is an important issue in distributed network; given a user request, a service discovery mechanism should locate and return a set of server addresses that match the description of the requested service [2]. A fundamental aspect to locate a service in a network is done via interaction between the provider of the service and the user. A server (i.e., provider) is a computational component placed at a given site and makes available a set of services. A client (i.e., customer) component located at another site requests the execution of a service via an interaction process with the server. The server performs the required service and may produce a result that will be delivered back to the client. It should be noted that Ubiquitous Computing (UC) is often considered the same as Pervasive Computing (PC) in the literature. Gaber in [22, 23] provides a classification of these two emergent infrastructures and argued that PC is a related concept that can be distinguished from ubiquitous computing in terms of environment conditions as follows. Ubiquitous Computing aims to provide users the ability to access services and resources all the time and irrespective to their location, while the main objective in Pervasive Computing (PC) is to provide spontaneous services created on the fly by mobiles that interact by ad hoc connections. In 2000, Gaber has proposed two alternative paradigms to the traditional client to server interaction paradigm (CSP) and its variants such push, pull and peer-to-peer interaction, to design and implement Ubiquitous and Pervasive applications. According to Gaber's classification, there are three interaction paradigms: the traditional Client/Server interaction paradigm (CSP), the adaptive Services-to-Client Paradigm (SCP) and the Spontaneous Service Emergence Paradigm (SEP) [22, 23, 24]. The adaptive Services/Client Paradigm (SCP) can be considered as the opposite to CSP. Indeed, with the traditional CSP paradigm, it is the user who should initiates a request, should know a priori that the required service exists and should be able to provide the location of a server holding that service. With the alternative SCP paradigm, it is the service that comes to the user. In other words, in this paradigm, a decentralized, intelligent and self-organizing middleware should be able to provide services to users according to their availability and the network status. As pointed out in [22, 23], such a middleware can be inspired, for example, from a biological system like the natural immune system. The second alternative SEP paradigm to the Client/Server one is more adequate for pervasive applications. It involves the concept of spontaneous emergence service composition that suits pervasive environments. More precisely, spontaneous services can be created on the fly and be provided by mobiles that interact by ad hoc connections [22, 23, 24].

Today service discovery is a very active area of research and development. Challenges are to build a scalable and efficient service discovery framework. More precisely, the management of the rapid increase of available services should not incur excessive overhead and create bottleneck. Also, the time of the request resolution should be reasonably fast [2, 19, 20, 21]. However, very little has been done regarding the access control problem. More precisely, the main problem is how to manage access policies to disparate services that are not under the control of a single system designer/administrator [1, 3]. A computational contract that defines an exchange of services between a client and a provider of services is required. Contracts need to be negotiated and signed electronically too, to ensure that both the client and the provider accept their terms as binding.

In this paper, we propose an agent-based solution that meets the automatic computerized contracts for negotiating and controlling services usage in service discovery systems. Contracts express the terms under which agents representing clients and agents representing servers promise to exchange resources, such as processor time.

The rest of the paper is structured as follows. In section 2, related work of secure service discovery systems is presented. Section 3 presents a contract-based model for service discovery systems. In section 4, a modeling study of considered process is given. Conclusion and future works are given in section 5.

## II. Secure service discovery system

Service discovery systems are systems that enable services to advertise their existence in a dynamic way, and can be discovered, configured and used by other devices with a minimum or without manual efforts. In open environment where there are many opportunities for both fraudulent services and misbehaving clients, service discovery systems are subject to a real challenge from security. In other words, to provide high confidence to users, these systems must have the capabilities of authenticating users and service providers, verifying the integrity of services, protecting the confidentiality of information, controlling the access to services based on security policies, and detecting malicious services and users.

In most known service discovery systems, trustworthy and secure accesses to such services are only critical requirements addressed. For example, Jini protects the access to the service but not discovery of the service [21]. Also, communication is done via Java RMI, which is not-encrypted. In contrast, SLP provides authentication in the local domain but not cross-domain. Indeed, no mechanism is offered for authentication of users [19]. SSDS (Secure Service Discovery System) provides a number of specific improvements in security [20]. Security is a core component of SSDS where communications are both encrypted and authenticated. SSDS help clients to determine the trustworthiness of services.

In SSDS [20], key entities in the architecture include the clients, the providers, and the discovery servers. Clients query the service discovery framework for the providers of their requested services. A client only interacts with its home discovery server. Providers provide application-level distributed services. To advertise its service, each provider sends service announcement to the service discovery framework. Similar to a client, a provider only interacts with its home discovery server. Discovery servers are key entities in the architecture. They act as brokers between service clients and providers. There is a discovery server in each domain. The discovery servers are organized as a hierarchy. More precisely, each provider server at the lowest level is called the home provider by the hosts in that lowest level domain. To perform service discovery, there will be exchange of service queries and service announcements between discovery servers. Within this architecture, servers can be dynamically added to scale up the system under heavy load. The hierarchy of servers also detects and restarts a failed server. SDS service availability is announced by periodic (authenticated) multicasts from discovery servers. The discovery server cache service information. An important distinction of the SSDS provides extremely strong mandatory security: all parties are authenticated, and all message traffic is encrypted. More precisely, SSDS uses: 1) Public Key authenticated SDS server announcements that assures authenticity of discovery service; 2) One-way encrypted service announcements (combined public and private key protocol) that assures privacy and authenticity of service descriptions; 3) Secure RMI to authenticate and encrypt remote method invocation; and 4) Certificates and a Certificate Authority structure to authenticate all principals. However, the SSDS system focuses only on communication security and secures accesses to such services but not on access control problem. In the rest of this paper a contract-based approach for negotiating and controlling services usage in service discovery systems and in particular for SSDS is presented.

## III. Access control model

The model presented in this section concerns the access control to discovered and selected computing services. A service is an abstract concept, representing a piece of computation - an application or a component - or a hardware device, such as a stereo display, processor, or a measurement instrument. More precisely, services are bound to specific hosts, representing hardware devises (e.g. disk), logical system objects (e.g. socket) or software entities, which are managed by an application [2].

### *A. Service description*

For a client - whether human or another software component to access a service, that service must be described by a clearly public interface. For example, services can be expressively described by a language description in order to obtain effective matches between their capabilities. Web service description languages should be used to describe software components. They define components identification and accessing methods that enable the discovery and the use of these components (i.e., the resources). A service description language is sufficiently expressive and flexible that it can be used not only in discovery, but also negotiation proposals and agreements. We can use any particular language description of to describe resources and permit to clients to compare different resources descriptions and choose the required. An example of service description language that can be used is WSDL (Web Services Description Language) [10]. This language would seem to provide a good basis for the definition of service syntax and forms the basic for a lot of web service composition [11]. For example, a software component can be described by the following information: identifier, type (software, hardware), description, identifier of provider, resource requirement of its execution (OS, min

RAM, min CPU, in bandwidth for transfer, etc) [10]. More precisely, in a software component description, there are three main parts: 1) the attributes, 2) the Capsule, and the Constraints and requirements [9]. Attributes include the characteristic of a software component such as the operations that can be invoked and their respective input and output parameters. The capsule gives information about the service localization, the invocation protocol and the port. The constraints and requirements give information about the required resources needed to execute the service. The following example shows a description of video service component (i.e., Web service).

***Attributes***
service-identifier = "Video-1-12";
num instances = 3 ;
type = "Wservice";
isMobile = "No"; //
description = "Video service";
provider-identifier = "Video-1";
input-parameters = {Type of film, Name of film, etc};
output-parameters = {XML Doc video Details};
price = 5;
***Capsule***
location = "video.domain.com";
protocol = https;
port = 80;
***Constraints and Requirements***
diskfree > 100; //Kbytes
memoryfree >= 128; //Kbytes
OpSys = "SOLARIS 2.5"

The Requirements part includes execution needs of the component towards the available resources. For example, a software component requires more than 100 Kbytes of free disk space and at least 128 Kbytes of free memory to be executed. Furthermore, the software component can be executed under SOLARIS 2.5 environment.

In resources that are bound to specific hosts, like service description, there are three main parts in the description: attributes, capsule, and constraints. The Attributes part includes characteristics of a resource, such as location, CPU usage, and free memory. These values can change over time. The Constraints part includes constraint expressions defined by the resource provider for the allocation of this resource [9]. The following example shows a description of a workstation. In this example, the workstation has 228 megabytes free memory, 10 gigabytes disk free, and running Solaris 2.5. The workstation is provided by CLUS and located at "clus.univ.edu.com".

***Attributes:***
resource-identifier = " CLUS-M1";
type = "Machine";
description = "workstation of seas group at univ";
provider-identifier = "CLUS001";
diskfree = 10; //gigabytes
memoryfree = 228; //megabytes
cache = 8; //megabytes
OpSys = "SOLARIS 2.5";
price = 5;
***Capsule***
location = "clus.univ.edu.com ";
***Constraints:***
untrusted = {"NA"}

*B. Negotiation process*

In this section, a lifecycle model to help us understand the interactions which take place between a client and a server engaged in the service discovery and access is presented. In this model each client is represented by an agent called client agent and each server is represented by an agent server agent. The lifecycle of an interaction between the client agent and the server agent is as follows: service localization, negotiation of contract, formation and examining of the contract, signature of the contract, activity, and termination. In the first phase, a client agent wishing to purchase access to a service must locate potential service providers able to meet its requirement. This is done by the client agent and the server agent. This phase depends on the protocol used in each service discovery system [19]. In the second phase, a client agent enters into the negotiation with one or more of these potential server agents, to see if they can agree mutually acceptable terms of the required service, i.e., the provider with the highest offer. More precisely, the client agent seeking to use the service must negotiate with the server agents offering the service matching its needs. The outcome of this phase is an agreement specifying the terms that both the client agent and the server agent consider acceptable such as the price, the delivery date, etc. In other words, the client agent and the server agent establish binding contract that provides both principals agree to their terms. All participants must agree and examine the contract in advance. This agreement could be expressed using digital signatures to prove contract acceptance.

The fourth phase permits to the client agent and the server agent to sign the contract; a copy of the contract together with both signatures proves that it was accepted. In other words, the server agent can express its authorization as an explicit contract to supply services. Conversely, the client agent should also enter into the contract, promising to abide by the service access, and eventually to provide payment or other services in exchange for those of the server agent [4]. This contact represents the granted permissions for accessing the required service. It defines an exchange of services between the server agent and the client agent. Thus a contract specifies the server agent promises to offer services through the contract and service requirements that specify the minimum levels of services the server will offer. It specifies also that the client agent will be responsible for paying the server agent for services used under the contract. For example, the following example presents a contract between a client agent desired a video service described above. There are three main parts: 1) Contract Information, 2) Server Information and 3) client Information. Contract information part includes all information that concerns the contract such as the identifier, the type of the contract, etc. For example, the contract-id is the identifier of the contract in order to distinguish the case where there are multiple contracts for the same service. The Server Information part includes information concerning the server such as the identifier of the server agent, the location, etc. Provider-Identifier and client-Identifier represent the actors who are part in the contract. Duration represents the maximum time the server agent can take to finish the service. Start-time and end-time represent the time during which the contract is valid.

***Contract Information***
Contract-id= Cont-10
Contract-type: on-demand
Duration: 60 minutes
Start-time=9h:00
End-time=10h:00
***Server Information***
service-identifier = " Video-1-12";
type = "software";
description = "Video service";
provider-identifier = " Video-1";
input-parameters = { Type of film, Name of film, etc };
output-parameters = { XML Doc video Details};
price = 5;
location = " video-domain.com";
protocol = https;
port = 80;
***Client information***
Client-identifier=Id-client

In the fifth phase, the contract is eventually regulated through payment messages and the transaction process of using the resource or the service can start running. The discovery server expects that the client agent will not attempt to access other services than those it required, and expects that the server agent will provide the required service. In addition, the transaction may be automatically monitored by the discovery server, and parties would be warned if any behavior outside the agreed terms of the contract takes place. It should be noted that the contract can also be modified on the fly, to rectify a contract violation such as a resource loss, with a new contract exchange. More precisely, when a client agent attempts to use the corresponding service, the permission for accessing the service is granted accordingly. If the client agent tries to perform an action that is not performed by the contract the access is denied.

The final phase consists of an optional termination of the request or a final termination of the contract. In addition, the contract that ensures the service is reserved from the time the contract is signed until its expiration. Once the contract is expired, it is not valid and a new contract must be signed between the client agent and the server agent. All these phases are controlled by asynchronous messages for requesting actions and exchanging signatures between the client agent and the server agent.

Electronic signature and associated processes are issues arousing an ever-increasing interest in many researches in the field of security in information systems [5, 6, 7, 8]. They become a vital component of the emerging electronic business infrastructures. The main goal is to provide users with time management and signed- electronic content protection. In the rest of this paper, we address the purpose and the usage of signature and time-stamps on negotiated electronic contracts between server agents and client agents in service discovery systems. The specification and modeling of electronic signature process using Petri net together with the validation of some properties are presented.

## IV. Electronic signature process modeling

Electronic signature and associated processes are managed with synchronization, parallelism and concurrency phenomena [15]. So, variety of these phenomena makes the study of such protocols difficult and requires the use of several complementary theories for their description and analysis. As it can be performed for telecommunication networks, computer systems or transportation systems, we show how the Petri net tool can be used to model electronic signatures and associated processes. Such tool has attracted the attention of many researchers [13, 14, 16, and 17]. Petri net model enables us to obtain some performance indices and to derive easily some graphical properties of the studied protocol [12].

In order to check and validate the contract signing process described above, we represent its behavior by a graphical model using Petri net model. In this model, transitions model the events (e.g. signature, authenticity, hash coding, time-stamping…) and their firings model the occurrence of these events. Recall that a PN is a graph composed of two nodes: places and transitions. The oriented arcs connect certain places to certain transitions, or conversely. With each arc, we associate a weight (non negative integer). In a formal way, a PN is a 5-uple PN = (P, T, A, W, M0) where:

- P = {P1, …, Pn} is a finite set of places;
- T = {T1, …, Tm} is a finite set of transitions (represented with line segments);
- $A \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs;
- $W = A \rightarrow \{1, 2 \ldots\}$ is the weight function associated with arcs;
- $M0 = P \rightarrow \{0, 1, 2 \ldots\}$ is the initial marking of graph.

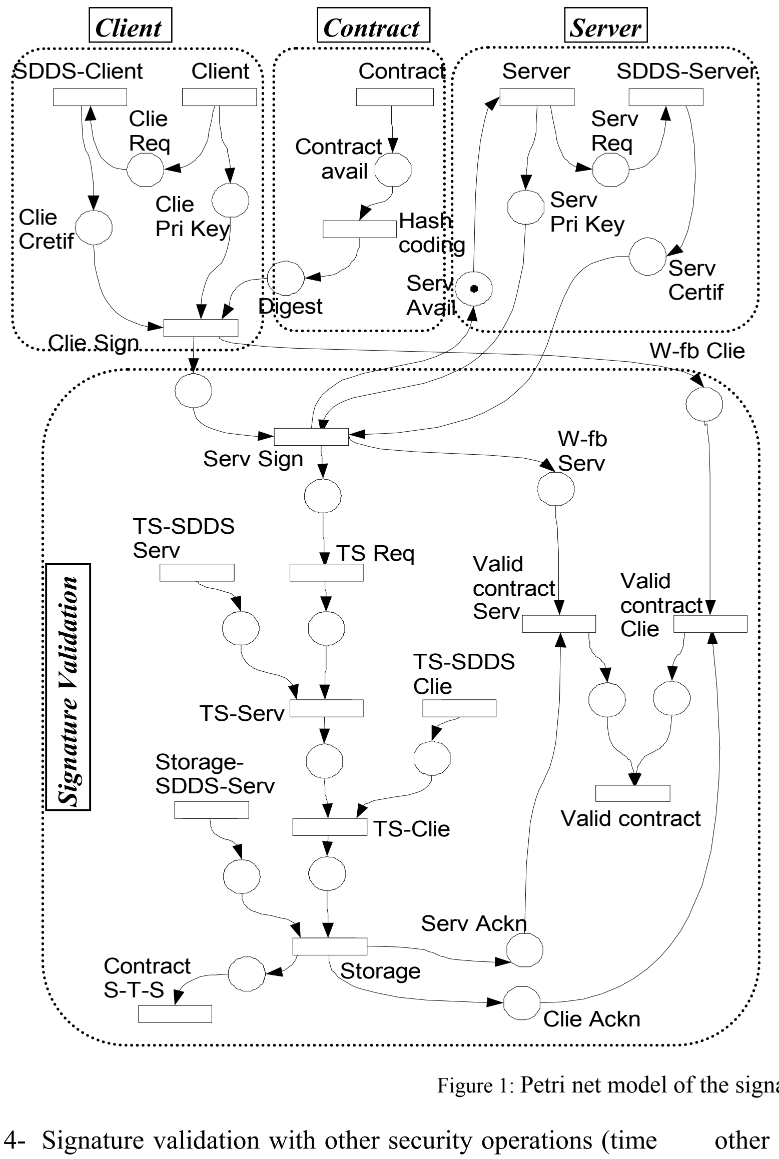
An important class of PN is timed event graphs (TEG). For this class, each place has only one input and one output transition. In addition, each place is associated with time interval which means that a token may remain in a place for this time interval. The considered protocol will be modelled by TEG model. The aim of using this tool is to be able to describe, in a simple and effectiveness way, the behaviour of dynamic systems by mathematical and linear equations in (max, plus) algebra [15] which will be the subject of our next contribution.

The figure 1 presents the Petri net of the Electronic signature and associated processes applied to SSDS system described above. The graph is composed of four sub-graphs:

1- Contract with its digest: after creation and negotiation of contract between a client agent and a server agent, the firing of transition Contract means that the content of the contract is valid (token in place Contract-avail). To sign this contract electronically one can use its digest obtained using a hash-coding function (token of place Digest). We note that several hash-coding functions exist in the literature. Among these functions an alternative method for generating digests of electronic document is developed in [17, 18]. Using this method, one can grasp the original contract ideas from its hash-code.

2- Client agent part with its certificate authority SSDS-Client: in this part of the model, the client agent, which needs a service, sends its request to the certificate authority SSDS-Client for asking a valid certificate (token in place Clie-Req) to sign the contract with a sever token in place (Clie-Pri-Key). The firing of transition SSDS-Client means that the certificate authority delivers a valid certificate to the client agent (token in place Clie-Certif). Once the three places Clie-Certif, Clie-Pri-Key and Digest are occupied each one by a token, then the client agent could sign (firing of transition Clie-sign) the contract and transmit it to the server agent to affix its signature, which means that the client agent sends its request to the server agent and wait for the response. This is modelled by adding a token to the situated place between transitions Clien-Sign and Serv-Sign, and another token in the place W-fb-Cli.

3- Server agent side with its certificate authority SSDS-Server: like for the client agent side, the server agent asks for a valid certificate from its certificate authority (token in place Serv-Req). Once this certificate is obtained and the server agent is free to sign (presence of tokens in places Serv-Pri-Key and in Serv-Certif), the transition Serv-Sign is validated and fired.



**Legend:**

***Server part***
**SDDS-Sever:** Service Discovery DS (Server side)
**Serv Req**: Server request
**Serv Certif**: Server certificate
**Serv Pri Key**: Server Private Key
**Serv Sign**: Server signature
**W-fb Serv**: Server waiting feedback
**TS Req**: Time stamping request
**TS-SDDS Serv**: Time stamping SDDS server
**TS-Serv:** Time stamping (server side)
**Serv Akno**: Server Acknowledgement
**V contract-Serv**: Valid contract (server side)

***Client part***
**SDDS-Client**: Service Discovery DS (Client side)
**Clie Req:** Client request
**Clie Pri Key**: Client Private Key
**Clie Certif**: Client certification
**Clie Sign**: Client signature
**W-fb Clie:** Client waiting feedback
**TS-SDDS clie**: Time stamping SDDS client
**TS-Clie**: Time Stamping (Client side)
**Clie Ackn**: Client Acknowledgement
**V contract-Clie:** Valid contract (client side)

***Contract***
**Contract avail**: Contract available
**Hash coding**: Contract hash coding
**Digest:** contract digest that will be signed by private key
**Contract S-T-S:** Contract signed time-stamped and stored

Figure 1: Petri net model of the signature protocol.

4- Signature validation with other security operations (time stamping, storage and validation). Once the signature is made by the client agent and the server agent (presence of a token in the place situated between Serv-Sign and Ts-Req), other security operations will be done on the contract. The sub-model “signature validation” represents the three steps by which the contract transits before its validation. These steps are: time-stamping (firing of transitions TS-SSDS-

Serv, TS-Serv for the server, and TS-SSDS-Clie and TS-Clie for the client), storage which is only done by the server agent certificate authority (firing of transitions Storage-SSDS-Serve and Storage). Finally, the validation of the contract is done by both the server agent (firing of transition Valid-contract-Serv) and the client agent (firing of transition Valid-contract-clie) and Valid-contract.

In the legend (figure 1), significances of Petri net model elements are mentioned.

By analyzing the Petri net model presented in the figure 1, the following properties that concern the evolution of the signature process between client agents and a server agent are verified to validate the contract signature process:

a- Existence of a marking enabling to fire any sequence of transitions

b- Monotony: which means that the growing of number of tokens in model places of a marking preserve the firing possibility of a sequence of transitions

c- Boundness: a place P is called k-bounded if for all marking M reachable from M0 (initial marking), $M(p) \leq k$. To ensure this property in our model, we suppose that the holding time of a token in certain places (W-fb Serv and W-fb Clie, see figure 1) is bounded by a fixed limit. If this constraint is respected the contract will be valid. Otherwise, the client request can not be satisfied.

d- Livness: this means that the firing of each transition, independently of the net evolution from its initial marking, is possible. We say that the transition is alive and each process event can be reached.

e- Absence of blocking: this property means that there is no marking from which a transition is not fired.

## V. Conclusion

In this paper an access control model based on negotiation and establishing contract is presented. Contract links computations to the resource model, by expressing agreements between clients and provides as promises to exchange resources, and binding them to the underlying computations that use the resources. The resource description allows efficient assessment and prediction of contracts' impact. The second part of the paper addresses the modeling and proves formally the proposed process, using Petri net. This model enables to have appropriate properties of safety, liveness and absence of blocking of the considered contract signature protocol. More precisely, contract signing schemes should be analyzed in more detail to establish efficient contract transformations that participants can use to establish a mutually satisfactory contract. Future research addresses the enhancement of this model by using a (max, plus) algebra.

## VI. References

[1] Y. Gidron, I. Ben-Shaul, O. Holder, and Y. Arior, "Dynamic Configuration of Access Control for Mobile Components in FARGO". *Concurency and Computation*, V 13, N° 1, pp. 5-21, January 2001.

[2] M. Bakhouya, J. Gaber, "Adaptive Approaches for Ubiquitous computing". In *Mobile networks and wireless sensor networks*. Eds. Hermes Science, ISBN 2-7462-1292-7, pp. 129-163, 2006.

[3] B. Firozabadi, M. Sergot, "Contractual access control". In Procs. of 10th *International Workshop of Security Protocols*, Cambridge (UK), 2002, citeseer.ist.psu.edu/firozabadi02contractual.html.

[4] B. Ninham Shand, "Trust for Resource Control: Self-enforcing Automatic Rational Contracts between Computers", *UCAM-CL-TR-600*, ISSN 1476-2986.

[5] B. Preneel, A. Bosselaers, H. Dobbertin, "The Cryptographic Hash Function RIPEMD-160". *CryptoBytes*, vol. 3, No. 2, pp. 9-14. 1997.

[6] D. Rieupet, M. Wack, N. Cottin, D. Assossou, "Signature Electronique Multiple". *Atelier Sécurité des Systèmes d'Information*, XXIIème Congrès INFORSID, Mai 2004.

[7] J. Linn, "RFC 1421: Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", IAB IRTF PSRG, IETF PEM WG, February 1993.

[8] National Institute of Standards and Technology: Digital Signature Standard (DSS). Federal Information Processing Standards Publication, FIPS PUB 186-2, January 2000.

[9] Z. Maamar, Q. Z Zheng, and B. Benatallah, "On Composite Web Services Provisioning in an Environment of Fixed and Mobile". *Information Technology and Management Journal*, Special Issue on Workflow and e-Business, Kluwer Academic Publishers, 2004

[10] P. Kellert, F. Toumani, "Les Web Services Sémantiques". Research Report LIMOS/RR 03-15, 11 Juillet 2003.

[11] S. Higels, D. Lewis, "State of art: Service composition". M-Zones Deliverable 1, pages 88-136, www.m-zones.org/deliverables/d1_1/papers/

[12] A. Nait-Sidi-Moh, M. Wack, "Modelling of Process of Electronic Signature with Petri Nets and (max, plus) Algebra". *Computer science and its applications* (LNCS), Editeur Springer, Vol. 3483, Issue IV, pp. 792-801, ISBN: 3-540-25863-9. 2005.

[13] A. Di Febbraro, S. Sacone, "Hybrid Modelling of Transportation Systems by means of Petri Nets". *IEEE International Conference on Systems, Man and Cybernetics* (SMC'98) 1998; Vol.°1; 131-135.

[14] E. Castelain, K. Mesghouni, "Regulation of a Public Transport Network with consideration of the passenger flow: modeling of the system with High-level Petri Nets". *IEEE International Conference on Systems, Man and Cybernetics* (IEEE SMC'02). Tunisia: Hammamet; 2002; Vol. 6; 250-254.

[15] F. Baccelli, G. Cohen, G-J. Olsder, J-P. Quadrat, "Synchronisation and linearity. Algebra for discrete Event Systems". John wiley et sons; 1992.

[16] R. David, H. Alla, "Du grafcet aux réseaux de Petri". Book, France : Paris; 1992.

[17] M. Wack, A. Nait-Sidi-Moh, S. Lamrous, N. Cottin, "Meaningful Electronic Signatures and Associated Processes Formalization Proposals''. *Journal of Artificial Intelligence and Law*, Editeur Springer. ISSN 0924-8463 (Print) 1572-8382 (Online), URL: http://www.springerlink.com/content/0626g21x25634128/

[18] M. Wack, A. Nait-Sidi-Moh, S. Lamrous, N. Cottin, J. Gaber, "Procédé de Signatures Signifiantes à Base de Mots Clefs", Office européen des brevets, Patentlaan 22280 HV Rijswiijk (ZH), N° 07 00847.

[19] E. Guttman, "Service Location Protocol: Automatic Discovery of IP Network Services". *IEEE Internet Computing*, 3(4), 71-80, ISSN: 1089-7801, 1999.

[20] D. Xu, K. Nahrstedt, D. Wichadakul, "Qos-aware Discovery of Wide-area Distributed Services". In *First IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid), Brisbane, Australia, pp. 92–99, 2001.

[21] R. Robert, "Discovery and Its Discontents : Discovery Protocols for Ubiquitous Computing". *UIUCDCS-R-99-2132*, March 25 2000.

[22] Gaber J. "New paradigms for ubiquitous and pervasive computing". *White paper*. Universite de Technologie de Belfort-Montbéliard, France, 2000.

[23] Gaber J. "New paradigms for ubiquitous and pervasive applications". *Proc. of First Workshop on Software Engineering Challenges for Ubiquitous Computing*. Lancaster, UK, 2006.

[24] Bakhouya, M., Gaber, J., "Ubiquitous and Pervasive Application Design". *Encyclopedia of Mobile Computing & Commerce*, Eds. D. Taniar, Idea Group Pub, Fb. 2007.