Petri nets for the verification of Ubiquitous Systems with Transient Secure Association[⋆]

Fernando Rosa-Velardo

Dpto. de Sistemas Informáticos y Programación
Universidad Complutense de Madrid
fernandorosa@sip.ucm.es

Abstract. *Transient Secure Association* has been widely accepted as a possible alternative to traditional authentication in the context of Ubiquitous Computing. We develop a formal model for the *Resurrecting Duckling Policy* that implements it, called TSA systems, which is based on Petri Nets, thus obtaining amenable graphical representations of our systems. We prove that TSA specifications have the same expressive power as P/T nets, so that coverability, that can be used to specify security properties, is decidable for TSA systems. Then we address the problem of implementing TSA systems with a lower level model that only relies on the secure exchange of keys. If we view these systems as closed then our implementation is still equivalent to P/T nets. However, if we consider an open framework then we need a mechanism of fresh name creation to get a correct implementation. This last model is not equivalent to P/T nets, but the coverability problem is still decidable for them.

1 Introduction

The term *Ubiquitous Computing* was coined by Mark Weiser [18] to describe environments full of devices that compute and communicate with its surrounding context and, furthermore, interact with it in a highly distributed but pervasive way. This paradigm gives rise to new problems, in the context of mobility, coordination and security, among others [8]. The new framework of ubiquitous computing affects traditional security assumptions. Authentication is probably the major security problem for ubiquitous systems, since it is a precondition for other properties as confidentiality or integrity. But in this new context, we can no longer rely on an always online policy, due to the unreliable nature of ad hoc nets. Thus, we cannot approach the problem of authentication by assuming the existence of an online server that verifies in real time the (global) identity of principals, nor the existence of public key repositories.

In order to solve this problem, several weak forms of authentication that substitute global identity-based authentication have been proposed. One of the proposals is that of *Transient Secure Association*, implemented by the *Resurrecting Duckling Policy* [16]. This policy is based on the fact that sometimes a

[⋆] Work partially supported by the Spanish projects DESAFIOS TIN2006-15660-C02-02, WEST TIN2006-15578-C02-01 and PROMESAS-CAM S-0505/TIC/0407.

principal may not know anything at all about some concrete device (e.g., when a user has just bought a PDA), but still it wants to be the only one to use it (e.g., because she is the first one who gets to push the button). In this case, even if they have no prior knowledge of each other, they can still become associated, so that from then on they share an asymmetric relation, one being a slave and the other becoming its master. Other proposals are based on the notion of *trust* [2].

Another issue of great importance in the field of ubiquitous computing is that of Coordination. Indeed, inherent to ubiquitous computing is the continuous change of state of the different components that form a system, that need to coordinate to achieve their goals. This fact, together with the unreliable nature of ad hoc nets and the heterogeneity of the components make the orchestration approach unrealistic and little robust, so that a choreographic coordination is more advisable. Therefore, a key technology is service discovery, which nowadays is based on a heavy standardization [10] (as in Sun's Jini or in Microsoft's UPnP).

In this paper we develop a formal model for choreographic coordination in ubiquitous systems, with primitives implementing the resurrecting duckling policy. The model, called TSA, is based on Petri nets. By using Petri Nets we obtain an amenable graphical representation of our systems, together with a number of theoretical results for their analysis, which would not hold in other more expressive models. In particular, we will prove that TSA specifications can be seen as a syntactic sugar formalism for P/T nets. Moreover, we will see that we can implement TSA systems by means of Mobile Synchronizing Petri Nets, which can be seen as a class of Coloured Petri Nets [6] with a single colour of pure names, and syntactic sugar for mobility, except for the fact that they can create fresh names, in a way borrowed from π-calculus [7]. We have proved several useful decidability results for them, which are inherited by TSA systems, even though TSA systems may represent infinite state systems. In particular, decidability of coverability in MSPN system implies that all the properties of TSA systems which can be stated in terms of coverability are decidable. For an extended version of this work see [15].

2 The Resurrecting Duckling policy

Let us briefly describe Transient Secure association and the Resurrecting Duckling security policy model. Transient secure association is about creating a link between two components, that establishes a master-slave relationship (typically between a controller and a peripheral). This link needs to be secure, so that no other principal can play the role of either the master or the slave. This produces a tree topology of master-slave relationships between components. The association must also be transient, in the sense that the topology can change if desired (by the master) along the execution of the system.

The Resurrecting Duckling policy model implements transient secure association with the following metaphor: *A duckling that emerges from its egg recognizes as its mother the first moving object that emits a sound*. From then on, the duckling obeys its mother duck until its death. Similarly, a peripheral can recognize as its mother duck the first entity that sends it a secret key, called *imprinting key*, which it will use to recognize its master. In order to make this association transient, we assume that the duckling can die and resurrect, and then it can recognize as its mother duck a different entity, to which it will be faithful from that point on. The policy can be described as a list of four principles:

1. **Two State principle.** Entities can be either *imprintable* (dead) or *imprinted* (alive). Imprintable entities can be imprinted by anyone and imprinted entities only obey their mother duck.
2. **Imprinting principle.** The step from imprintable to imprinted is called *imprinting*, and happens when an entity, from now on the mother duck, sends a key to the duckling.
3. **Death principle.** The step from imprinted to imprintable is called *death*. The default cause of death is the mother duck's order to its duckling to commit *seppuku*, though other causes are envisaged.
4. **Assassination principle.** It must be uneconomical for an attacker to kill a duckling (to cause its death).

3 TSA systems

Our aim is to define a model for ubiquitous systems, that we call TSA, based on Petri Nets and with primitives implementing the Resurrecting Duckling policy. It will be a variation of the one developed in [4]. A TSA system is essentially a set of localized nets that can move between locations and can imprint other nets, give orders to the nets they have previously imprinted, kill those nets, and been imprinted or receive orders from its master when imprinted.

We assume that the different components of a system have a type, taken from a set $\mathcal{O}$. We use a special type $\top \in \mathcal{O}$ for the type of components that need not be imprinted to be alive (e.g., users). Components can be dead or alive. Top or imprinted components are alive, otherwise they are dead. Alive components can imprint dead ones, thus creating an unique and asymmetrical link between them. From then on the imprinted component becomes a slave of the other, doing everything its master orders to it. However, when executing its code, the slave component can as well imprint other components, thus creating a hierarchical relation between components. Therefore, not only the component been killed becomes death, but also all of its slaves and so on, recursively.

We assume that components can only be imprinted within a certain range, that depends on the imprinting method being used (e.g., physical contact). We abstract from those particular methods and simply assume that the imprinting can take place whenever both components are in the same location. In particular, we assume that the key exchange that takes place when imprinting a component is done in a secure way. Besides, a component can only give orders to those slave components that are currently co-located with it.

Dually, a slave component can be killed by its owner only when they are co-located. Thus, we are implementing the default variant of the death principle, in which the death of the duckling is caused by the mother duck.

We will use the set $\mathcal{L}$, a set of location names. $\mathcal{O}$ is a set of component types, with a special element $\top \in \mathcal{O}$. We use a set S for the syntactic primitives of the standard used for coordination and $\mathcal{A}$ to denote autonomous actions. $\mathcal{A}$ contains among others the labels *go* k for every $k \in \mathcal{L}$. We denote $S? = \{s? \mid s \in S\}$ and $S! = \{s! \mid s \in S\}$. For $m > 0$, we denote by $\mathit{Labels}(m)$ the set $\mathcal{A} \cup S? \cup \bigcup_{i=1}^{m}(\{i\} \times \mathit{Labels}_{\mathit{Auth}})$, where $\mathit{Labels}_{\mathit{Auth}} = S! \cup \{\mathit{imprint}(o) \mid o \in \mathcal{O} \setminus \{\top\}\} \cup \{\mathit{kill}\}$.

Definition 1. *A TSA component is a tuple* $N = (P_N, T_N, F_N, \lambda_N, o_N)$ *such that* P_N *and* T_N *are disjoint finite set of places and transitions, respectively,* $F_N \subseteq (P_N \times T_N) \cup (T_N \times P_N)$, $\lambda_N : T_N \to \mathit{Labels}(m)$ *for some* $m > 0$ *and* $o_N \in \mathcal{O}$. *We say that* N *has type* o_N*, that* m *is the capacity of* N*, and we denote by* $\mathit{cap}(N)$ *the set* $\{1, \ldots, m\}$.

A TSA component is a typed-labelled Petri net. The capacity of a component represents the number of slaves it can have simultaneously imprinted. Its type o is just a syntactic category, which can be intuitively understood as the type of device it is (e.g., PDA, electronic note,...), as the imprinting protocol it uses when been imprinted or both simultaneously. The labelling of the transitions of the net establishes a partition in its transition set: Those transitions t with $\lambda(t) \in \mathcal{A}$ are autonomous transitions, that is, those that can be executed in an autonomous way, without needing to synchronize with others; If $\lambda(t) \in S?$ the transition is used to receive orders from their masters, when imprinted; Otherwise, we have that $\lambda(t) \in \{i\} \times \mathit{Labels}_{\mathit{Auth}}$. In this case the transition is used to *communicate* with its i-slave. If some net fires t with $\lambda(t) = (i, \mathit{imprint}(o))$ then some component of type o is henceforth a i-slave of it. If $\lambda(t) = (i, s!)$ then the firing of t instructs the corresponding slave to perform action $s?$. Finally, if $\lambda(t) = (i, \mathit{kill})$ then its i-slave is instructed to commit seppuku. Given a transition t we will denote by ${}^\bullet t = \{p \mid (p, t) \in F\}$, the set of preconditions of t and by $t^\bullet = \{p \mid (t, p) \in F\}$, its set of postconditions. If $\mathit{cap}(N) = \{1\}$ and $\lambda(t) = (1, \ell)$ we simply write $\lambda(t) = \ell$.

Definition 2. *A TSA system* $\mathcal{N}$ *is a set of pairwise disjoint TSA components.*

Intuitively, the nets N with $o_N = \top$ are components that need not be imprinted in order to execute their actions. We will denote by $P_{\mathcal{N}}$ the set of all places in $\mathcal{N}$ and by $T_{\mathcal{N}}$ the set of all transitions in $\mathcal{N}$, or just P and T when there is no confusion. Next we define the part of the state of the system regarding the dependency relations between its components.

Definition 3. *Given a TSA system* $\mathcal{N}$*, a dependency function of* $\mathcal{N}$ *is a partial mapping* $R : \{(N, j) \mid N \in \mathcal{N}, j \in \mathit{cap}(N)\} \to \mathcal{N}$. *A dependency function* R *induces a dependency graph* $G(R)$ *with set of nodes* $\mathcal{N}$ *and an arc from* N *to* N' *labelled by* j *if* $R(N, j) = N'$*, and we say that* N' *is a* j*-slave of* N *in* R.

We will call *directed tree* to any directed graph in which there is a node, that we call *root* of the tree, so that for every node in the graph there is a single path from the root to it. Therefore, a *directed forest* is a set of directed trees. Given two nodes s and s' of a directed forest, we will say s' is a descendant of s if there is a path in the forest from s to s'. Moreover, given a set A we will denote by $\mathcal{MS}(A)$ the set of multisets with elements in A.

Definition 4. *A marking of a TSA $\mathcal{N}$ is a tuple $\mathcal{M} = (M, loc, R)$, where $M \in \mathcal{MS}(P_{\mathcal{N}})$, $loc : \mathcal{N} \to \mathcal{L}$ and R is a dependency function of $\mathcal{N}$ such that $G(R)$ is a directed forest and for every $N \in \mathcal{N}$, if $o_N = \top$ then N is not a slave in R.*

Therefore, a marking is composed by an ordinary marking for Petri nets, a function specifying the locality of each component, and a dependency function. We say N is alive in R if $o_N = \top$ or it is a slave in R. Otherwise, we say it is dead. The components N and N' are co-located if $loc(N) = loc(N')$.

Now let us define the behaviour of TSA systems. In general, only alive components can fire transitions. Next we define the firing of autonomous transitions, that are as ordinary transitions in P/T nets, that remove tokens from precondition and add them in postconditions, except that they are restricted to alive components and can cause the movement of the executing net. We will denote by $+$ and $-$ the multiset union and the multiset difference, respectively.

Definition 5. *Let $\mathcal{N}$ be a TSA system, $\mathcal{M} = (M, loc, R)$ a marking of $\mathcal{N}$, N an alive component in R, and $t \in T_N$ such that $\lambda(t) \in \mathcal{A}$. We say that t is enabled in $\mathcal{M}$ if ${}^\bullet t \subseteq M$. In that case we say that t can be fired and then the marking (M', loc', R) is reached, where $M' = M - {}^\bullet t + t^\bullet$. If $\lambda(t) = go\ k$ then $loc'(N) = k$ and $loc'(N) = loc(N)$ for every $N' \neq N$. Otherwise, $loc' = loc$.*

Now we define the firing of imprinting transitions. The net firing any of them specifies the type it wants to imprint.

Definition 6. *Let $\mathcal{N}$ be a TSA system, $\mathcal{M} = (M, loc, R)$ a marking of $\mathcal{N}$, N and N' an alive and a dead component in R, respectively, both co-located, and $t \in T_N$ such that $\lambda(t) = (i, imprint(o_{N'}))$. We say that t is enabled in $\mathcal{M}$ if ${}^\bullet t \subseteq M$ and $(N, i) \notin Dom(R)$. Then t can be fired, to reach the marking (M', loc, R'), where $M' = M - {}^\bullet t + t^\bullet$ and R' extends R with $R'(N, i) = N'$.*

The imprinted net must be dead in order to be imprinted, and of the type required by the imprinting component. Moreover, the imprinting net must not already have an i-slave net. Now we define orders from masters to slaves.

Definition 7. *Let $\mathcal{N}$ be a TSA system, $\mathcal{M} = (M, loc, R)$ a marking of $\mathcal{N}$, N and N' two co-located alive components such that N' is a i-slave of N in R. Let us also consider $t \in T_N$ such $\lambda(t) = (i, s!)$ and $t' \in T_{N'}$ such that $\lambda(t') = s?$. We say that the pair of transitions (t, t') is enabled in $\mathcal{M}$ if ${}^\bullet t + {}^\bullet t' \subseteq M$. Then the pair (t, t') can be fired, getting (M', loc, R) where $M' = M - {}^\bullet t - {}^\bullet t' + t^\bullet + t'^\bullet$.*

Therefore, if N' is an i-slave of N then N can give orders to N', which is formalized by a synchronous firing of two transitions. Finally, let us define how a master can kill one of its slaves.

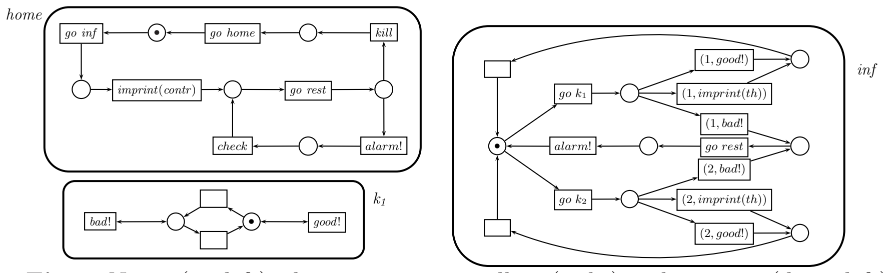

Fig. 1. Nurse (up left), thermometer controllers (right) and patients (down left)

Definition 8. *Let $\mathcal{N}$ be a TSA system, $\mathcal{M} = (M, loc, R)$ a marking of $\mathcal{N}$, N and N' two co-located alive components such that N' is a i-slave of N, and $t \in T_N$ such $\lambda(t) = (i, kill)$. We say that t is enabled in $\mathcal{M}$ if ${}^\bullet t \subseteq M$. Then t can be fired, getting (M', loc, R') where $M' = M - {}^\bullet t + t^\bullet$ and R' is the result of removing (N, i) from the domain of R, and removing $(\overline{N}, j)$ from the domain of R, for every $\overline{N}$ descendant of N' in $G(R)$ and every j.*

Reachability for TSA systems is defined as expected. We say a marking (M, loc, R) can be covered if there is a reachable marking (M', loc', R') such that $M \subseteq M'$, $loc = loc'$, $Dom(R) \subseteq Dom(R')$ and for every $(N, i) \in Dom(R)$, $R(N, i) = R'(N, i)$. The coverability problem consists on deciding if a given marking can be covered.

4 An application example

In this section we present a simple application example to illustrate the definitions in the previous section. Let us consider a scenario with several patients in two rooms of a hospital, called k_1 and k_2. Each of the patients has an electronic thermometer [17] that reads the temperature every once in a while, and willing to send (e.g., using RFID) a message stating whether everything is all right or not. We assume that these thermometers are inactive (that is, dead according to our terminology), except when a thermometer controller takes over them. Intuitively, we assume that every room has one of these controllers, that reads the messages output by thermometers, and sends an alarm message to a nurse whenever the temperature is not correct. For simplicity, we assume that all these controllers are in fact part of a single component that moves between rooms in the hospital and that it can only simultaneously control two thermometers, so that it has capacity two. Finally, there is a night shift nurse, that must check the temperature of the patients. For that purpose, it may imprint a controller, so that she is the only one to receive alarm messages from it. Therefore, the life cycle of a night-shift nurse consists on going from her home to the infirmary, imprinting a thermometer controller, going to rest, checking the patient in case she receives an alarm message, or killing the controller and going back home.

We model this scenario by means of a TSA system with several components (see Fig. 1). The first one represents the nurse, that we assume has type $\top$. Initially, it is located in location *home*. Then it can fire *go inf*, thus moving to the infirmary, where it can imprint the thermometer controller and then go to

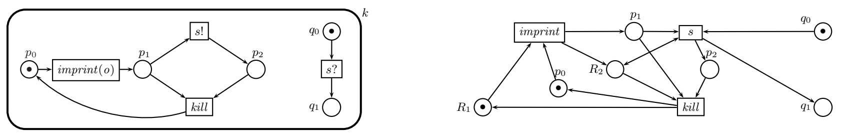

Fig. 2. Example of TSA system (left) and P/T net that simulates it (right)

rest. Notice that, since the nurse has capacity one, we use simple names as labels of its synchronizing transitions, such as *kill*, instead of $(1, \mathit{kill})$.

The thermometer controller, with type *contr*, is initially in location *inf*. The controller, however, does not have a $\top$ type, so that it must wait to be imprinted to perform any action. After been imprinted, it can go either to location k_1 or to k_2. In these locations there are patients. Now the controller has a choice between *good*!, *bad*! or $\mathit{imprint}(\mathit{th})$. However, since the patient thermometers are still dead (in our sense), all the controller can do at the present time is imprinting them. Then it can again choose between any of the locations. If it goes back again to the same place then it can read the thermometer and, if it reads *bad*, that is, if it synchronizes with the thermometer in transition *bad*!, then it goes to the resting room and sends an alarm message to the nurse.

Eventually, the system will reach a dependency state in which both thermometers are imprinted by the controller. At any point, the nurse can fire its *kill* transition, thus killing the controller (and, recursively, the controller killing its thermometers). Notice that the controller does not have any killing transition, so that it only kills its thermometers when the nurse kills it.

5 Verification of TSA systems

In the previous sections we have presented a model that allows us to specify systems using primitives that implement the resurrecting duckling policy, which are correct by definition. In this section we will see how we can use the known decidability results for the verification of the obtained TSA systems.

First of all, we will prove that the locality component of TSAs is not essential in the model. More precisely, we will say a TSA system is centralized if none of its components have movement transitions and all of them are initially located in the same location. Under these assumptions, we can ignore the locality component of centralized TSA systems, thus writing just (M, R) instead of (M, loc, R).

Lemma 1. *Every TSA system can be simulated by a centralized TSA system.*

Therefore, in the following we can assume that every TSA system is centralized. Now, we prove that every TSA system can be simulated by a P/T net.

Proposition 1. *Every TSA system can be simulated by a P/T net.*

The proof is based on the fact that, since the TSA system is considered as closed, the number of components is fixed in the initial state and, therefore, the number of dependency functions is finite. Figure 2 shows the P/T net (right) that simulates the TSA system in the left.

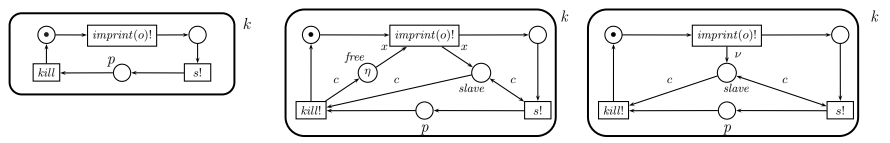

Fig. 3. TSA system (left), implementation (center) and open implementation (right)

By applying this result, we can use all the existing machinery for ordinary P/T nets in the analysis of our systems. For instance, coverability and home state problems are known to be decidable for P/T nets (for a survey see [3]). Then, in principle, we could algorithmically check any property that can be stated in terms of them. For example, regarding the example in the previous section, we could check that, whenever a thermometer is imprinted by a doctor, then whenever needed it eventually raises the alarm, and only to the right doctor, since that is a home state property. Another property that could be interesting to analyze, and can be stated in terms of coverability (adding a control point to the specification), is checking that it is not possible for a controller to (for example) send out an alarm message without first been imprinted. Assuming that we have several nurses and controllers, we could also specify in terms of coverability the property that says that any nurse has at most one controller.

So far, we are regarding our TSA systems as the specification of systems using the resurrecting duckling policy, since our primitives assume that the imprinting and killing mechanisms are always correct. However, it is also desirable to perform a lower level analysis, with weaker assumptions. For that purpose, we will prove that every TSA system can be implemented by an MSPN system. An MSPN system intuitively consists on a set of Petri nets, each of them at some location. Those nets can move between location and, synchronize between them. Moreover, and this is more important, they can manage pure names, that we call *identifiers*, and use them for authentication purposes, that is, as imprinting keys. Therefore, apart from ordinary tokens, MSPN components can have identifiers as tokens. For instance, a component could offer a synchronization only to components exhibiting a determined identifier. This is formalized by using variables as labels on arcs. MSPN components can also create fresh identifiers by means of a special variable ν, which can only be instantiated to fresh identifiers. However, they do not need to use this variable to implement TSA systems.

Proposition 2. *TSA systems can be simulated by MSPNs without name-creation.*

Figure 3 (center) shows the implementation of the TSA system in the left of that figure. MSPN systems without name-creation are still equivalent to P/T nets, so that we can take advantage of all the existing analysis methods for P/T nets even in the implementation of TSA systems as closed systems. The previous proof profits from the fact that we are regarding TSA systems as closed systems, so that we can extensively and statically analyze all of their participants. In particular, we can check (and force) that the components being imprinted, that is, receiving their masters imprinting key, throw away those keys when they are killed. Therefore, components can reuse those keys in a new imprinting.

However, it is not very realistic to deal with closed systems in the context of ubiquitous computing. Here we want our services to be offered in a global and uniform way, so that some of the principals that imprint or some of the devices that are imprinted may not be part of the system, but part of the *environment* of the system. In this case, we can no longer check that they erase the imprinting keys after been killed so that, if we reuse them, then the principals of the environment could use old keys to illegitimately interact with a component.

Proposition 3. *Every TSA system can be simulated by an open MSPN system.*

This result can be proved similarly to the previous one, with one crucial difference: Instead of having a predefined set of identifiers to imprint other components, each component will create a fresh identifier for every imprinting. The right of Fig. 3 shows the open implementation of the TSA system in its left, which is similar to that in the center, but it does not have a place *free* for free identifiers and it has the special variable ν labelling the arc to the place called *slave*. As we have proved in [12, 14], the general class of MSPN system (in which we allow name creation) is not equivalent to P/T nets anymore, but are not Turing-complete either. In particular, we have proved that coverability is still decidable, even in the open case.

6 Conclusions and Future Work

In this paper we have defined a model based on Petri Nets that addresses the problem of Secure Transient Association in ubiquitous systems. The fact of using Petri Nets gives us an amenable and easy to grasp model.

Then we study whether we can approach the problem of the verification the properties of these systems. As a first step, we prove that TSA systems, seen as specification of systems using the primitives of the Resurrecting Duckling Policy, are equivalent to P/T nets, so that we have many decidable properties for them as reachability, coverability or home state. Then we show how these primitives can be implemented solely relying on a mechanism that manages pure names, that can be seen as imprinting keys. These implementations can be studied from two different points of view: first assuming that our systems work in a closed environment, and then without this assumption. In the first case, we have proved that this implementation is still equivalent to P/T nets. However, this is no longer true in the second case, because we need to allow the creation of fresh identifiers, but even so we can approach the verification of their properties.

As future work, we plan to study in a more systematic way the kind of properties of TSA systems that can be studied. It would also be interesting to follow a dual approach, namely that in which we want to verify that a given system does indeed implement the resurrecting duckling policy, which would certainly lead us to approaches like proof carrying code [9] or dynamic typing [5]. We also plan to consider the studies about transactions in the context of Petri nets [1], to apply them to killing of slaves in our model, which are clearly transactions, though we do not treat them explicitly as such in the open implementation.

In [13] we presented a tool for the verification of MSPN system. We are currently extending this prototype to make it cope with the primitives presented in this paper, so that it automatically performs the translations we have described and checks the properties that we have proved to be decidable.

References

[1] R. Bruni and U.Montanari. *Executing Transactions in Zero-Safe Nets.* 21st Int. Conf. on Application and Theory of Petri Nets. LNCS vol. 1825. Springer, 2000.

[2] M. Carbone, M. Nielsen and V. Sassone. *A Calculus for Trust Management.* 24th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science. LNCS vol. 3328. Springer, 2004.

[3] J. Esparza and M. Nielsen. *Decidability issues for Petri Nets - a survey.* Bulletin of the EATCS 52:244-262 (1994).

[4] D.Frutos-Escrig, O.Marroquín-Alonso and F.Rosa-Velardo. *Ubiquitous Systems and Petri Nets.* Ubiquitous Web Systems and Intelligence,LNCS vol.3841.Springer,2005.

[5] M. Hennessy and J. Riely. *Resource Access Control in Systems of Mobile Agents.* High-Level Concurrent Languages. ENTCS 16:3–17. Elsevier, 1998.

[6] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.*. Volume 1, Basic Concepts. Monographs in TCS, Springer, 1997.

[7] R. Milner, J. Parrow and D. Walker. *A calculus of mobile processes I.* Information and Computation 100(1):1–40. Academic Press Inc., 1992.

[8] R. Milner. *Theories for the Global Ubiquitous Computer.* Foundations of Software Science and Computation Structures, LNCS vol.2987. Springer, 2004.

[9] G.C. Necula and P. Lee. *Safe, Untrusted Agents Using Proof-Carrying Code.* Mobile Agents and Security. LNCS vol. 1419. Springer, 1998.

[10] G.G. Richard. *Service Advertisement and Discovery: Enabling Universal Device Cooperation.* IEEE Internet Computing archive 4(5):18–26. IEEE Educational Activities Department, 2000.

[11] F. Rosa-Velardo, O. Marroquín-Alonso and D. Frutos-Escrig. *Mobile Synchronizing Petri Nets: a choreographic approach for coordination in Ubiquitous Systems.* MTCoord'05. ENTCS 150.

[12] F. Rosa-Velardo, D. Frutos-Escrig and O. Marroquín-Alonso. *On the expressiveness of Mobile Synchronizing Petri Nets.* 3rd Int. Workshop on Security Issues in Concurrency. ENTCS (to appear).

[13] F. Rosa-Velardo. *Coding Mobile Synchronizing Petri Nets into Rewriting Logic.* 7th Int. Workshop on Rule-based Programming. ENTCS (to appear).

[14] F. Rosa-Velardo, and D. Frutos-Escrig. *Symbolic Semantics for the Verification of Security Properties of Mobile Petri Nets.* 4th Int. Symposium on Automated Technology for Verification and Analysis. LNCS vol. 4218. Springer, 2006.

[15] F. Rosa-Velardo. *Petri nets for the verification of Ubiquitous Systems with Transient Secure Association.* Technical Report 2/07, Dept. Sistemas Informï¿$\frac{1}{2}$ticos y Computacïï¿$\frac{1}{2}$n, Universidad Complutense de Madrid, 2007.

[16] F. Stajano. Security for Ubiquitous Computing. Wiley Series in Communications Networking & Distributed Systems. John Wiley & Sons, 2002.

[17] R. Want. *Enabling Ubiquitous Sensing with RFID.* Computer vol.37(4), pp.84-86. IEEE Computer Society Press, 2004.

[18] M. Weiser. *The Computer for the 21st Century.* In Human-computer Interaction: Toward the Year 2000, pp.933-940. Morgan Kaufmann Publishers Inc, 1995.