Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 186 (2007) 101–120
www.elsevier.com/locate/entcs

Automated Synthesis of Enforcing Mechanisms for Security Properties in a Timed Setting[1]

Ilaria Matteucci[2]

Istituto di Informatica e Telematica - C.N.R., Pisa, Italy
Dipartimento di Scienze Matematiche ed Informatiche, Università degli Studi di Siena

Abstract

In [21,22] we have presented an approach for enforcing security properties. It is based on the automatic synthesis of *controller programs* that are able to detect and eventually prevent possible wrong action performed by an external agent. Here, we extend this approach also to a timed setting. Under certain assumptions, we are also able to enforce several information flow properties. We show how to deal with parameterized systems.

Keywords: Partial model checking, information flow, automated synthesis of controllers.

1 Overview

Many approaches for the analysis of security properties have been successfully developed in the last two decades. More recently there also has been interest on developing techniques to study how to enforce security properties. A prominent example is the notion of security automata in [29] and some extensions proposed in [16].

In [18,20], the authors propose a methodology based on known techniques in concurrency and process logics theory for the formal analysis of several security properties (including information flow).

In [21,22], we have extended this line of research with a method for automatically enforcing the desired security properties. In particular, we have shown how to secure a system S with a possibly un-specified component, said X, through the usage of

[1] Work partially supported by CNR project "Trusted e-services for dynamic coalitions" and by EU-funded project "Software Engineering for Service-Oriented Overlay Computers"(SENSORIA) and by EU-funded project "Secure Software and Services for Mobile Systems "(S3MS).

[2] Email: Ilaria.Matteucci@iit.cnr.it

doi:10.1016/j.entcs.2007.03.025

a controller program, said $Y \rhd X$, that prevents X to express a behavior that in cooperation with S could make the overall system violate a safety property ϕ. Four different controller operators are then defined. They mimic the behavior of security automata in [6,16].Moreover in [21,22] we present a method for synthesizing a controller program for each of them.

The goal of this paper is to extend previous works by defining controller operators able to enforce other security properties as information flow ones. Moreover, here we deal with systems in a timed setting. Hence, this work represents a significant contribution to [21,22], because we not only enrich the set of security properties that can be enforced but also we deal with the elapsing of time in a system. To this aim we mainly use an appropriate extension of *partial model checking* (see [2,27]).

We show how our theory can be extended to treat with parameterized systems, $S = P_n$ where n is the parameter and $P_n = \underbrace{P \| P \| \ldots \| P}_{n}$.

Our logical approach is also able to deal with composition problems, that have been considered as an interesting issue in [6].

This paper is organized as follows. Section 2 recalls the basic theory about the analysis of timed security properties, especially non-interference, as properties of open systems. Section 3 explains our approach by dealing also with composition of properties and parameterized systems. Section 4 shows a simple example and Section 5 presents a discussion on related work. Eventually, Section 6 concludes the paper.

2 Background

In this section we briefly recall some useful technical machinery and also a logical approach for dealing with information flow properties and security properties in general.

2.1 A Timed variant of CCS

The CCS (*Calculus of Communicating Systems*) language is an algebra of processes introduced by Milner (see [23]). It is very suitable to describe concurrent and distributed systems.

Here we present a variant of CCS that permits to deal also with the elapsing of time. A lot of different languages have been developed in the literature to describe the system in a timed setting (see [3,27]). We follow a simple approach, where time is discrete, actions are durationless and there is one special *tick* action to represent the elapsing of time (see [13,27]). These are features of the so called *fictitious clock* approach of, e.g. [8,15,32]. A global clock is supposed to be updated whenever all processes of the system agree on it, by globally synchronizing on action *tick*. Hence, between two global synchronizations on action *tick* all processes proceed asynchronously by performing durationless actions.

Let $\mathcal{L}$ be a set of visible actions and let τ be a special action that models an internal computation, i.e., it is not visible by an external observer. Let $\bar{()} : \mathcal{L} \mapsto \mathcal{L}$

be a *complementation* function such that $\forall l \in \mathcal{L} : \bar{\bar{l}} = l$ and let $f : \mathcal{L} \to \mathcal{L}$ be the *relabeling* function. Let $Act = \mathcal{L} \cup \{\tau\} \cup \{tick\}$ be the set of actions, ranged over by $\alpha, \beta, \ldots$ and let $\mathcal{L} \cup \{\tau\}$ be ranged over by $a, b, c, \ldots$

The syntax of timed CCS is the following:

$$E ::= \mathbf{0} \mid A \mid \alpha.E \mid E_1 + E_2 \mid E_1 \| E_2 \mid E \backslash L \mid E[f]$$

where $\alpha \in Act$, $L \subseteq \mathcal{L}$.

The set of timed CCS *processes* is denoted with $\mathcal{E}$, ranged over by $E, F, P, Q \ldots$ We will often use some common syntactic simplifications, e.g., omission of trailing $\mathbf{0}$'s as well as omission of brackets on restriction on a single action. $Sort(E)$ is used to denote the set of actions that occurs in the term E.

Timed CCS operators have the following informal meaning:

- $\mathbf{0}$ is a process that does nothing;
- A (*agent*) is a set of processes names;
- $\alpha.E$ (*prefix*) is a process that can perform an α action and then behaves as E. In particular $tick.E$ represents a process willing to let one time unit pass;
- $E_1 + E_2$ (*choice*) represents the nondeterministic choice between the two processes E_1 and E_2; when both are able to perform a *tick* action then $E_1 + E_2$ can perform this action and reach a configuration where both summand derivatives can still be chosen;
- $E_1 \| E_2$ (*parallel*) is the parallel composition of processes that can proceed in an asynchronous way but they must synchronize on complementary actions to make a communication, represented by an internal action τ. This is the core operator for time: both components must agree on performing a *tick* action;
- $E \backslash L$ (*restriction*) is the process E when actions in $L \cup \overline{L}$ are prevented;
- $E[f]$ (*relabeling*) is the process E in which every performed action $\hat{\alpha}$ is relabeled by f;
- $i(E)$ (*idling*) allows process E to wait indefinitely. At every instant of time, if process E performs an action α then the whole system proceeds in this state, while dropping the idling operator.

The formal semantics of timed CCS processes is described by *labeled transition system* (*LTS*, for short). A *LTS* over Act is a pair $(\mathcal{E}, \mathcal{T})$ where $\mathcal{T}$ is a ternary relation $\mathcal{T} \subseteq (\mathcal{E} \times Act \times \mathcal{E})$, known as a *transition relation*. It is the least relation between timed CCS processes induced by axioms and inference rules in Table 1. Such a relation is well-defined.

2.2 Strong and weak bisimulations

It is often necessary to compare processes that are expressed using different terms but that have the same behavior. We recall some useful relations on processes.

Definition 2.1 Let $(\mathcal{E}, \mathcal{T})$ be an LTS of concurrent processes, and let $\mathcal{R}$ be a binary

Prefixing:

$$\frac{}{\alpha.E \xrightarrow{\alpha} E}$$

Choice:

$$\frac{E_1 \xrightarrow{\alpha} E_1'}{E_1 + E_2 \xrightarrow{\alpha} E_1'} \quad \frac{E_2 \xrightarrow{\alpha} E_2'}{E_1 + E_2 \xrightarrow{\alpha} E_2'} \quad \frac{E_1 \xrightarrow{tick} E_1' \quad E_2 \xrightarrow{tick} E_2'}{E_1 + E_2 \xrightarrow{tick} E_1' + E_2'}$$

Parallel:

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 \| E_2 \xrightarrow{a} E_1' \| E_2} \quad \frac{E_2 \xrightarrow{a} E_2'}{E_1 \| E_2 \xrightarrow{a} E_1 \| E_2'} \quad \frac{E_1 \xrightarrow{l} E_1' \quad E_2 \xrightarrow{\overline{l}} E_2'}{E_1 \| E_2 \xrightarrow{\tau} E_1' \| E_2'}$$

$$\frac{E_1 \xrightarrow{tick} E_1' \quad E_2 \xrightarrow{tick} E_2'}{E_1 \| E_2 \xrightarrow{tick} E_1' \| E_2'}$$

Restriction:

$$\frac{E \xrightarrow{\alpha} E'}{E \backslash L \xrightarrow{\alpha} E' \backslash L} (\alpha \notin L \cup \overline{L})$$

Relabeling:

$$\frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$$

Idling:

$$\frac{E \not\xrightarrow{tick} \quad E \not\xrightarrow{\tau}}{i(E) \xrightarrow{tick} i(E)} \quad \frac{E \xrightarrow{tick} E'}{i(E) \xrightarrow{tick} i(E')} \quad \frac{E \xrightarrow{\alpha} E'}{i(E) \xrightarrow{\alpha} E'}$$

Table 1
Operational semantics for timed CCS.

relation over $\mathcal{E}$. Then $\mathcal{R}$ is called *timed strong simulation*, denoted by $\prec_t$, over $(\mathcal{E}, \mathcal{T})$ if and only if, whenever $(E, F) \in \mathcal{R}$ we have:

- if $E \xrightarrow{a} E'$ then there exists F' s.t. $F \xrightarrow{a} F'$ and $(E', F') \in \mathcal{R}$,
- if $E \xrightarrow{tick} E'$ then there exists F' s.t. $F \xrightarrow{tick} F'$ and $(E', F') \in \mathcal{R}$.

A *timed strong bisimulation* is a relation $\mathcal{R}$ s.t. both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are timed strong simulations. We represent with $\sim_t$ the union of all the timed strong bisimulations.

We give now the notion of *observational relations*as follows. Let $\alpha \neq \tau$ $\hat{\alpha} = \alpha$ and $\hat{\tau} = \epsilon$. $E \stackrel{\tau}{\Rightarrow} E'$ (or $E \stackrel{\epsilon}{\Longrightarrow} E'$) if $E \xrightarrow{\tau}{}^{*} E'$ (where $\xrightarrow{\tau}{}^{*}$ is the reflexive and transitive closure of the $\xrightarrow{\tau}$ relation); $E \stackrel{\hat{\alpha}}{\Rightarrow} E'$ if $E \stackrel{\epsilon}{\Longrightarrow} \xrightarrow{\hat{\alpha}} \stackrel{\epsilon}{\Longrightarrow} E'$ [3] .$Der(E)$ is the set of derivatives of E, i.e., the set of processes that can be reached through transition relations.

[3] Note that it is a short notation for $E \stackrel{\tau}{\Rightarrow} E_\tau \xrightarrow{\alpha} E'_\tau \stackrel{\tau}{\Rightarrow} E'$ where E_τ and E'_τ denote intermediate states that is not important for this framework.

It is useful to consider the class of processes that allow time proceed, the so-called *weakly time alive* processes.

Definition 2.2 A process E is *directly weakly time alive* iff $E \stackrel{tick}{\Longrightarrow}$ [4], while it is *weakly time alive* iff for all $E' \in Der(E)$, E' is directly weakly time alive.

Since $E \stackrel{\alpha}{\longrightarrow} E'$ implies $Der(E') \subseteq Der(E)$, it directly follows that if E is weakly time alive, then any derived E' of E is weakly time alive as well. Moreover, it is worthwhile noticing that the above property is preserved by the parallel composition.

Now we define the *timed weak bisimulation* relation (see [24]). This equivalence permits to abstract to some extent from the internal behavior of the systems, represented by the invisible τ actions.

Definition 2.3 Let $(\mathcal{E}, \mathcal{T})$ be an *LTS* of concurrent processes, and let $\mathcal{R}$ be a binary relation over $\mathcal{E}$. Then $\mathcal{R}$ is called *timed weak simulation*, denoted by $\preceq_t$, over $(\mathcal{E}, \mathcal{T})$ if and only if, whenever $(E, F) \in \mathcal{R}$ we have:

- if $E \stackrel{a}{\longrightarrow} E'$ then there exists F' s.t. $F \stackrel{a}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{R}$,
- if $E \stackrel{tick}{\longrightarrow} E'$ then there exists F' s.t. $F \stackrel{tick}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{R}$.

A *timed weak bisimulation* is a relation $\mathcal{R}$ s.t. both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are timed weak simulations. We represent with $\approx_t$ the union of all the timed weak bisimulations.

Every timed strong simulation is also a timed weak one (see [24]).

2.3 Equational μ-calculus and Partial Model Checking in a timed setting

Equational μ-calculus is a process logic well suited for specification and verification of systems whose behavior is naturally described using states changes by means of actions. It permits to express a lot of interesting properties like *safety* ("nothing bad happens") and *liveness* properties ("something good happens"), as well as allows to express equivalence conditions over *LTS*. In order to define recursively properties of a given system, this calculus uses fixpoint equations. Let α be in Act and X be a variable ranging over a finite set of variables V. Given the grammar:

$$A ::= X \mid \mathbf{T} \mid \mathbf{F} \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \langle\alpha\rangle A \mid [\alpha]A$$

$$D ::= X =_\nu AD \mid X =_\mu AD \mid \epsilon$$

where the symbol $\mathbf{T}$ means *true* and $\mathbf{F}$ means *false*; $\wedge$ is the symbol for the conjunction of formulae, i.e., $A_1 \wedge A_2$ holds iff both of the formulae A_1 and A_2 hold, and $\vee$ is the disjunction of formulae and $A_1 \vee A_2$ holds when at least one of A_1 and A_2 holds. The *possibility operator* $\langle\alpha\rangle A$ means that "there exists a transition labeled by α after that A holds". The *necessity operator* $[\alpha]A$ means "for all α-actions performed A holds". $X =_\mu A$ is a minimal fixpoint equation, where A is an assertion (i.e. a simple modal formula without recursion operator), and $X =_\nu A$ is a maximal fixpoint equation. Roughly, the semantics $[\![D]\!]$ of the list of equations

[4] This means that we are no interested to the final state of the transition.

D is the solution of the system of equations corresponding to D. According to this notation, $[\![D]\!](X)$ is the set of values of the variable X, and $E \models D \downarrow X$ can be used as a short notation for $E \in [\![D]\!](X)$. The formal semantics is in Table 2.

The following standard result of μ-calculus will be useful in the reminder of the paper.

Theorem 2.4 ([30]) *Given a formula ϕ it is possible to decide in exponential time in the length of ϕ if there exists a model of ϕ and it is also possible to give an example of such model.*

Partial model checking (*pmc* for short) is a technique that was originally developed for compositional analysis of concurrent systems (see [2]). The intuitive idea underlying the *pmc* is the following: proving that $E\|F$ satisfies a formula ϕ ($E\|F \models \phi$) is equivalent to prove that F satisfies $\phi_{//_E}$, that is a modified specification of ϕ ($F \models \phi_{//E}$), where $//_E$ is the partial evaluation function for the parallel composition operator. The formula ϕ is specified by use the *equational μ-calculus*. A useful result of partial model checking is the following.

Lemma 2.5 ([2]) *Given a process $E\|F$ and a formula ϕ we have: $E\|F \models \phi$ iff $F \models \phi_{//E}$.*

The reduced formula $\phi_{//E}$ depends only on the formula ϕ and on process E. No information is required on the process F which can represent a possible enemy. Thus, given a certain system E, it is possible to find the property that the enemy must satisfy to make a successful attack on the system. It is worth noticing that partial model checking function may be automatically derived from the semantics rules used to define a language semantics. Thus, the proposed technique is very flexible.

According to [2], when ϕ is *simple*, i.e. it is of the form X, $\mathbf{T}$, $\mathbf{F}$, $X_1 \wedge \ldots \wedge X_k \wedge [\alpha_1]Y_1 \wedge \ldots \wedge [\alpha_l]Y_l$, $X_1 \vee \ldots \vee X_k \vee \langle\alpha_1\rangle Y_1 \vee \ldots \vee \langle\alpha_l\rangle Y_l$, then the size of $\phi_{//E}$ is bounded by $|\phi| \times |E|$. Referring to [1] any assertion can be transformed to an equivalent simple assertion in linear time. Hence we can conclude that the size of $\phi_{//E}$ is polynomial in the size of ϕ and E.

A lemma similar to Lemma 2.5 holds for every timed CCS operators (see [2,27]). The partial model checking function for parallel operator is given on Table 3.

2.4 Characteristic formulae

A *characteristic formula* is an equational μ-calculus formula that completely characterizes the behavior of a (state in a) *LTS* modulo a chosen notion of behavioral relation.

In Section 2.2 two different behavioral relations are described. Here we define the notion of characteristic formula for a given finite state process E w.r.t. both of those equivalences.

Definition 2.6 ([25]) Given a finite state process E, its *characteristic formula*

$$[\![\mathbf{T}]\!]'_\rho = S \quad [\![\mathbf{F}]\!]'_\rho = \emptyset \quad [\![X]\!]'_\rho = \rho(X) \quad [\![A_1 \wedge A_2]\!]'_\rho = [\![A_1]\!]'_\rho \cap [\![A_2]\!]'_\rho$$
$$[\![A_1 \vee A_2]\!]'_\rho = [\![A_1]\!]'_\rho \cup [\![A_2]\!]'_\rho \quad [\![\langle\alpha\rangle A]\!]'_\rho = \{s \mid \exists s' : s \xrightarrow{\alpha} s' \text{ and } s' \in [\![A]\!]'_\rho\}$$
$$[\![[\alpha]A]\!]'_\rho = \{s \mid \forall s' : s \xrightarrow{\alpha} s' \text{ implies } s' \in [\![A]\!]'_\rho\}$$

We use $\sqcup$ to represent union of disjoint environments. Let ρ be the environment (a function from variables to values) and σ be in $\{\mu, \nu\}$, then $\sigma U.f(U)$ represents the σ fixpoint of the function f in one variable U.
$[\![\epsilon]\!]_\rho = []$ $\quad [\![X =_\sigma AD']\!]_\rho = [\![D']\!]_{(\rho \sqcup [U'/X])} \sqcup [U'/X]$
where $U' = \sigma U.[\![A]\!]'_{(\rho \sqcup [U/X] \sqcup \rho'(U))}$ and $\rho'(U) = [\![D']\!]_{(\rho \sqcup [U/X])}$.
It informally says that *the solution to* $(X =_\sigma A)D$ *is the* σ *fixpoint solution* U' *of* $[\![A]\!]$ *where the solution to the rest of the lists of equations* D *is used as environment.*

Table 2
Equational μ-calculus

w.r.t. timed weak bisimulation [5] $D_E \downarrow X_E$ is defined by the following equation for every $E' \in Der(E)$, $\alpha \in Act$:

$$X_{E'} =_\nu \Big(\bigwedge_{\alpha; E'' : E' \xrightarrow{\alpha} E''} \langle\langle\hat{\alpha}\rangle\rangle X_{E''}\Big) \wedge \Big(\bigwedge_{\alpha} ([\alpha]\Big(\bigvee_{E'' : E' \stackrel{\hat{\alpha}}{\Rightarrow} E''} X_{E''}\Big)\Big)\Big)$$

where $\langle\langle\hat{\alpha}\rangle\rangle$ of the modality $\langle\hat{\alpha}\rangle$ which can be introduce as abbreviation (see [25]):

$$\langle\langle\epsilon\rangle\rangle\phi \stackrel{def}{=} X \text{ where } X =_\mu \phi \vee \langle\tau\rangle X \quad \langle\langle\hat{\alpha}\rangle\rangle\phi \stackrel{def}{=} \langle\langle\epsilon\rangle\rangle\langle\hat{\alpha}\rangle\langle\langle\epsilon\rangle\rangle\phi$$

These derived modalities can be equivalently expressed in equational form.
The following lemma characterizes the power of these formulae.

Lemma 2.7 ([25]) *Let* E_1 *and* E_2 *be two different finite-state processes. If* ϕ_{E_2} *is characteristic for* E_2 *then:*

(i) *If* $E_1 \approx_t E_2$ *then* $E_1 \models \phi_{E_2}$*;*
(ii) *If* $E_1 \models \phi_{E_2}$ *and* E_1 *is finite-state then* $E_1 \approx_t E_2$*.*

Now we define the notion of characteristic formula for a finite state process E w.r.t. timed weak simulation relation as follows.

Definition 2.8 Given a finite state process, its *characteristic formula w.r.t. timed weak simulation* $D_E \downarrow X_E$ is defined by the following equation for every $E' \in Der(E)$, $\alpha \in Act$:

$$X_{E'} =_\nu \bigwedge_{\alpha} ([\alpha]\Big(\bigvee_{E'' : E' \stackrel{\hat{\alpha}}{\Rightarrow} E''} X_{E''}\Big))$$

.

[5] Note that the presence of *tick* actions does not influenced the definition of characteristic formula

$$
\begin{aligned}
(D\downarrow X)//t &= (D//t)\downarrow X_t \\
\epsilon//t &= \epsilon \\
(X =_\sigma AD)//t &= ((X_s =_\sigma A//s)_{s\in Der(E)})(D)//t \\
X//t &= X_t \\
\langle a\rangle A//s &= \langle a\rangle(A//s) \vee \bigvee_{s\xrightarrow{a}s'} A//s',\ \text{if } a\neq\tau \\
\langle \tau\rangle A//s &= \langle \tau\rangle(A//s) \vee \bigvee_{s\xrightarrow{\tau}s'} A//s' \vee \bigvee_{s\xrightarrow{\alpha}s'} \langle\overline{\alpha}\rangle(A//s') \\
\langle tick\rangle A//s &= \begin{cases} \langle tick\rangle A//s' & s \xrightarrow{tick} s' \\ \mathbf{F} & otherwise \end{cases} \\
[a]A//s &= [a](A//\mathrm{s}) \wedge \bigwedge_{s\xrightarrow{a}s'} A//s',\ \text{if } a\neq\tau \\
[\tau]A//s &= [\tau](A//\mathrm{s}) \wedge \bigwedge_{s\xrightarrow{\tau}s'} A//s' \wedge \bigwedge_{s\xrightarrow{\alpha}s'} [\overline{\alpha}](A\ //s') \\
[tick]A//s &= \begin{cases} [tick]A//s' & s \xrightarrow{tick} s' \\ \mathbf{T} & otherwise \end{cases} \\
A_1 \wedge A_2//s &= (A_1//s) \wedge (A_2//s) \\
A_1 \vee A_2//s &= (A_1//s) \vee (A_2//s) \\
\mathbf{T}//s &= \mathbf{T} \\
\mathbf{F}//s &= \mathbf{F}
\end{aligned}
$$

Table 3
Partial evaluation function for parallel operator $E\|(\_)$ of timed CCS.

It is easy to note that the characteristic formula of a process w.r.t. timed simulation is simpler than the formula defined in the Definition 2.6. However, also in this case we obtain an interesting result.

Lemma 2.9 *Let E be a finite-state process and let $\phi_{E,\preceq_t}$ be its characteristic formula w.r.t. weak simulation. Let F be a finite-state process s.t. $F \preceq_t E$. We have $E \models \phi_{E,\preceq_t} \Leftrightarrow F \models \phi_{E,\preceq_t}$.*

2.5 Properties expressible in equational μ-calculus

The μ-calculus is a very expressive logic (see [7]). There are a lot of (security) properties that can be expressed by the equational μ-calculus, e.g. *access control rule* as "a file f can be only read and not written", *history based access control rule* as "it is not possible to open a file a if we have already open a file b", *information flow properties* (as we will see in the next section) and so on.

In order to better explain we report some very simple examples. It is possible to find a formula to express *safety properties* as, for instance, the absence of deadlock in a system, i.e., in any state reachable from the initial one it is always possible to perform an action: $X =_\nu \langle\_\rangle\mathbf{T} \wedge [\_]X$.[6] A *liveness property* like "a state satisfying ϕ can be reached" is expressed by $X =_\mu \langle\_\rangle X \vee \phi$.

In the following section we describe more in detail a logical approach for specifying and analyzing information flow properties because in the rest of the paper we are interested to show how it is possible to define controllers to enforce these.

2.5.1 *A logical approach for specifying and analyzing security properties as information flow in a real-time setting*

Information flow is a main topic in the theoretical study of computer security. We can find several formal definitions in the literature (see [12,17,28]) for concurrent processes. To describe this property, we can consider two users, *High* and *Low* interacting with the same computer system. We wonder if there is any flow of information from *High* to *Low*. A central property is the *Non Deducibility on Composition* (*NDC*, see [12]) that has been proposed as generalization of the classical idea of *Non-Interference* to nondeterministic systems: low level users cannot infer the behavior of high level users from the system because for low level users the system is always the same. This idea can be represented as follow:

$$\forall \Pi \in \ \textit{High} \text{ users } E \| \Pi \equiv E \text{ w.r.t. Low users}$$

(where $\|$ stands for a suitable composition operator and $\equiv$ for an equivalence relation).

The natural extension of *NDC* to a timed setting is *timed NDC* (*tNDC*, for short).

We denote with *tBNDC* a security property called *Bisimulation Non Deducibility on Compositions* in a timed setting (see [12]).

Before formally introducing it, we need to discuss briefly on the nature of the admissible *High* users. In the timed *CCS* model we cannot consider all high processes for the interaction with the system. Indeed, we must restrict ourselves to *weakly time alive* processes that can perform only action in $ActH$[7] $\cup \{\tau, tick\}$. We call $\mathcal{E}_H$ the set of such processes. The reason is the following: a process Π that is not weakly time alive may prevent time from elapsing when composed in parallel with some system E. Indeed, in a compound process, time can pass if and only if all components let it pass. Hence, a high user which is not weakly time alive could block the time flow also for low users and we certainly want to avoid this unrealistic (and undesirable) possibility. The *tBNDC* property in timed *CCS* can be thus defined as follows.

Definition 2.10 $E \in tBNDC$ if and only if $\forall \Pi \in \mathcal{E}_H$ we have $(E\|\Pi)\backslash H \approx_t E\backslash H$.

[6] In writing properties, here and in the rest of the paper, we use the shortcut notations $[\_]$ means $[Act]$ and, equivalently, $\langle\_\rangle$ means $\langle Act\rangle$.

[7] It is the subset of actions in which there areall the actions of an *High* user.

Due to the presence of the universal quantification, $tBNDC$ is not very easy to check.

Let $\mathcal{H}$ be the set of high users that are composed with the system when it is checked (as done in [20]). Under certain constraints on the set $\mathcal{H}$, we can provide a method for reducing the verification of $tBNDC^{\mathcal{H}}$ membership to a validity problem in equational $\mu-$calculus, where by $tBNDC^{\mathcal{H}}$ we denoted $tBNDC$ for process in $\mathcal{H}$,

Definition 2.11 $E \in tBNDC^{\mathcal{H}}$ if and only if $\forall \Pi \in \mathcal{H} : (E \| \Pi) \backslash H \approx_t E \backslash H$.

By using the characteristic formula for $\approx_t$ of $E \backslash H$, we obtain the following characterization: [8]

$$E \in tBNDC^{\mathcal{H}} \quad \text{iff} \quad \forall \Pi \in \mathcal{H} : (E \| \Pi) \backslash H \models \phi_{\approx_t, E \backslash H} \tag{1}$$

Now, we can apply the partial evaluation function w.r.t. $E, \backslash H$ to the formula $\phi_{\approx_t, E \backslash H}$ by getting a formula ϕ'. Then the previous equation is equivalent to check that every process in $\mathcal{H}$ satisfies ϕ'. Indeed, the behavior of E has been evaluated and encoded in the formula ϕ'. Thus:

$$E \in tBNDC^{\mathcal{H}} \quad \text{iff} \quad \forall \Pi \in \mathcal{H} \quad \Pi \models \phi'$$

We expect to have decidability results only if we restrict ourselves to finite-state systems (see [18]). Let $fs = \{E | Der(E) \text{ is finite}\}$ be the set of finite state processes. We also require that the set $\mathcal{L}$ of visible actions is finite. If the membership in $\mathcal{H}$ can be defined by a formula ϕ'' then we obtain that the previous problem is equivalent to:

$$E \in tBNDC^{\mathcal{H}} \quad \text{iff} \quad \forall \Pi \in \mathcal{H} \quad \Pi \models \phi'' \Rightarrow \phi'$$

The validity problem for this logic may be shown to be decidable by using the same proof techniques of [31].

3 Enforcing security properties

Let S be a system, and let X be one component that may be dynamically changed (e.g., a downloaded mobile agent). We say that the system $S \| X$ enjoys a security property expressed by a logical formula ϕ if and only if for every behavior of the component X, the behavior of the system S enjoys ϕ, i.e.:

$$\forall X \quad (S \| X) \backslash H \models \phi \tag{2}$$

where H is the set of high actions as before. By using the partial model checking approach proposed in [18], we can focus on the properties of the possibly un-trusted component X, i.e.:

$$\forall X \quad X \models \phi_{//_{S, \backslash H}} \tag{3}$$

[8] Actually, this is true only if we consider finite-state processes.

Thus, we may study whether a potential enemy could exists and, in particular, which are necessary and sufficient conditions that an enemy should satisfy to alter the correct behavior of the whole system. In order to protect the system we may simply check the correctness of each process X before it is executed or, if it is not possible, we may define a controller that, in any case, forces each process to behave correctly. We may distinguish several situations [9] depending on the control we may have on the process X:

(i) if X performs an action we may detect and intercept it;

(ii) in addition to (i), it is possible to know which are the possible next steps of X;

(iii) X whole code is known and we are able to model check it [10].

In the scenarios (i) and (ii) we may imagine to develop some controllers that force the intruder to behave correctly, i.e., as prescribed by the formula $\phi_{//_{S,\backslash H}}$.

3.1 Synthesis of controller programs

We wish to provide a framework where we are able to enforce specific security properties defining a new operator, said $Y \triangleright^* X$, that can permit to control the behavior of the target X, given the behavior of a control program Y. We can image different behaviors for controller operators. For instance in [21,22] security automata defined in [6,29] are modeled by process algebra operators. These operators permit to enforce safety properties as the automata they model. Here we develop a theory for enforcing also information flow properties, specified as $tBNDC$, and we give some examples of controllers that do this.

First of all we note that, by introducing a controller operator the Formula (2) becomes

$$\exists Y \quad \forall X \quad (S \| Y \triangleright^* X) \backslash H \models \phi \tag{4}$$

Equivalently, by *pmc*, we get:

$$\exists Y \; \forall X \; (Y \triangleright^* X) \models \phi' \text{ where } \phi' = \phi_{//(S \backslash H)} \tag{5}$$

While the Formula (5) should be the property to manage, it might not be easy. Considering the following additional assumption:

Assumption 3.1 *For every X and Y, we have: $Y \triangleright^* X \approx_t Y$.*

If the controller operator satisfies the Assumption 3.1, the Formula (5) is equivalent to:

$$\exists Y \; Y \models \phi' \tag{6}$$

As a matter of fact, the previous assumption permits us to conclude that $Y \triangleright^* X$ and Y are timed equivalent. It is possible to reduce the Formula (5) to the Formula (6) by resorting to the concept of characteristic formula for timed equivalence (Definition 2.6).

[9] The last two pose several decidability issues.

[10] We do not consider here the possibility of manipulating the code.

It is important to note that the Assumption 3.1 is a sufficient condition to enforce some properties and in particular to enforce $tBNDC$ (although in the scenario (i) it would not be very useful, since it could often override the high user instructions). To force $tBNDC$ we have also to require not only that Y satisfies the formula ϕ' but also that is weakly time alive. Since the w.t.a. property can be expressed by a μ-calculus formula, $\phi_{w.t.a.}$ for short, we have to find a model for $\phi' \wedge \phi_{w.t.a.} = \phi''$.

While designing such a process Y could not be difficult in principle, we can take advantage of our logical approach and obtain an automated procedure. As matter of facts, exploiting the Theorem 2.4, it is possible to decide if there exists a model Y for ϕ'' and find it. For the semantics of conjunction, if Y satisfies ϕ'' it satisfies ϕ. Hence Y is suitable for Formula (6). Unfortunately, the satisfiability procedure has complexity that is, in the worst case, exponential in the size of the formula. Hence, since we express security property by a formula, in the worst case, the procedure to find Y has exponential complexity.

3.2 Example of controller operators

In this section we give two examples of possible semantics definition of a controller operator.

Let E and F be two processes. We define the controller operator $\triangleright'$ by the following rules.

$$\text{(a)}\ \frac{E \xrightarrow{\alpha} E' \quad F \xrightarrow{\alpha} F'}{E \triangleright' F \xrightarrow{\alpha} E' \triangleright' F'}\ \alpha \neq \tau \qquad \text{(b)}\frac{E \xrightarrow{a} E'}{E \triangleright' F \xrightarrow{a} E' \triangleright' F} \qquad \text{(c)}\frac{F \xrightarrow{\tau} F'}{E \triangleright' F \xrightarrow{\tau} E \triangleright' F'}$$

This operator forces the system to make always the right action also if we do not know what action the agent X is going to perform. Whereas we are interested to the observational equivalence between processes, X can also perform the action τ. Under this hypothesis and the additional one that X is weakly time alive, this controller operator is able to wait an action of a possible intruder, then, after timeout [11] expires, performs the right action. It is possible to note that $\triangleright'$ is *tick*-deterministic (the action *tick* can be performed only if the rule (a) can be applied.).

Proposition 3.1 *Let E and F be two finite-state processes. If both E and F are weakly time alive, also $E \triangleright' F$ is weakly time alive.*

Proposition 3.2 *The operator $\triangleright'$ enjoys Assumption 3.1.*

Another controller operators $\triangleright''$ can be defined as follows.

$$\text{(a)}\ \frac{E \xrightarrow{\alpha} E' \ F \xrightarrow{\alpha} F'}{E \triangleright'' F \xrightarrow{\alpha} E' \triangleright'' F'}\ \alpha \neq \tau \qquad \text{(b)}\ \frac{E \xrightarrow{a} E' \ F \overset{a}{\not\Rightarrow} F'}{E \triangleright'' F \xrightarrow{a} E' \triangleright'' F} \qquad \text{(c)}\ \frac{F \xrightarrow{\tau} F'}{E \triangleright'' F \xrightarrow{\tau} E \triangleright'' F'}$$

[11] Referring to [14] the notion of timeout can be formally defined through the combination of process algebra operators

This operator looks at the action performed by F and, if E and F perform the same action α then the whole system performs it. On the contrary, the whole system performs the action performed by E. The τ action can be always performed by both of the processes. This controller is *tick*-deterministic and we can prove the following propositions.

Proposition 3.3 *Let E and F be two finite-state processes. If both E and F are weakly time alive, also $E \triangleright'' F$ is weakly time alive.*

Proposition 3.4 *The operator $\triangleright''$ enjoys Assumption 3.1.*

3.2.1 Feasibility issues for our controllers

The introduction of a controller operator helps to guarantee a correct behavior of the entire system.

We discuss in this section the feasibility of our controller operators, i.e. how and also if, these controllers $\triangleright'$, $\triangleright''$, can be implemented.

For the first controller operator, $\triangleright'$, we can note that it may in any moment neglect the external agent X behavior, unless X performs τ. The behavior of the system may simply follow the behavior of the controller process. In particular, the controller may always choose to perform its correct action, rather than waiting for an action by the target. Thus, it would be easily implementable in all three scenarios.

The operator $\triangleright''$ cannot be implemented in the scenario (i): if we are not able to decide a priori which are possible next steps that the external agent is going to perform we cannot implement the second rule of (11). In the scenario (ii) $\triangleright''$ operator would be implementable. It would be also possible in this scenario to give priority to the first rule in order to allow always the correct action of the target. Thus, controller $\triangleright''$ would be our favorite, if we could consider scenario (ii) because it leaves that the external agent executes correct action, if the first rule can be applied, and denies the unwanted situation checking hem by the second rule. Also in this case internal actions performed by X are permitted without any action of Y.

3.3 Composition of properties

Our logical approach is able to struggle successfully with composition problems. If we should force many different security policies, we have only to force the conjunction of them. In formulas: let $\phi_1, \cdots, \phi_n$ be n different security policies, S be our system and X be a target, we have:

$$\forall X(S\|X)\backslash H \models \phi_1 \quad \ldots \quad \forall X(S\|X)\backslash H \models \phi_n$$

Hence, for all X the system $(S\|X)\backslash H$ have to satisfy ϕ_i for $i = 1, \ldots, n$. This is equivalent to satisfy the conjunction of these, $\bigwedge_{i=1,\cdots,n} \phi_i = \phi$. Hence the problem is reduced to:

$$\forall X(S\|X)\backslash H \models \phi \tag{7}$$

that is the same situation that we have described by the Formula (5).

3.4 Analysis of parameterized systems

A parameterized system describes an infinite family of (typically finite-state) systems; instances of the family can be obtained by fixing parameters. Consider a parameterized system $S = P_n$ defined by parallel composition of processes P, e.g. $\underbrace{P\|P\|\ldots\|P}_{n}$. The parameter n represents the number of processes P present in the system S.

Example 3.5 Consider the system with one consumer process C and n producer processes P. Each process P is defined $P \stackrel{def}{=} a.P$ where $a \in Act$, and the process C is $\bar{a}.C$. The entire system is $(P_n\|C)\backslash\{a\}$ and the processes communicate by synchronization on $\bar{a}$ and a actions.

Referring to the Formula (2) we have

$$\forall n\, \forall X \quad P_n\|X \models \phi \tag{8}$$

It is possible to note that in the previous equation there are two universal quantifications; the first one on the number of instances of the process P and the second one on the possible unknown agents.

In order to eliminate the universal quantification on the number of processes, first of all, we define the concept of *invariant formula w.r.t. partial model checking for parallel operator* as follows.

Definition 3.6 A formula ϕ is said an *invariant w.r.t. partial model checking* for the system $P\|X$ iff $\phi \Leftrightarrow \phi_{//_P}$.

It is possible to prove the following result.

Proposition 3.7 *Given the system $\forall i\ P_i\|X$. If ϕ is an invariant formula for this system then*

$$\forall X\ \ (\forall n\ \ P_n\|X \models \phi \quad \textit{iff} \quad X \models \phi)$$

In order to apply the theory developed in Section 3.1, we show a method to find the invariant formula. According to [5], let ψ_i be defined as follows

$$\psi_i = \begin{cases} \phi'_1 & \text{if } i = 1 \\ \psi_{i-1} \wedge \phi'_i & \text{if } i > 1 \end{cases}$$

By definition of ψ_i and by Lemma 2.5, $\forall j$ s.t. $1 \leq j \leq i$ $(X \models \phi'_j) \Leftrightarrow X \models \psi_i$. Hence $X \models \psi_i$ means that $\forall j$ s.t. $1 \leq j \leq i$ $P_j\|X \models \phi'$. We say that ψ_i is said to be *contracting* if $\psi_i \Rightarrow \psi_{i-1}$. If $\forall i$ $\psi_i \Rightarrow \psi_{i-1}$ holds, we have a chain that is a said a *contracting sequence*. If it is possible to find the invariant formula ψ_ω for a chain

of μ-calculus formulae, that is also said *limit of the sequence*, then the following identity holds.

$$\forall X \;\; X \models \psi_\omega \Leftrightarrow \forall n \geq 1 \quad P_n \| X \models \phi' \tag{9}$$

Now we can apply the reasoning made in Section 3.1. Hence we are able to define a controller operator that forces each process to behave correctly and synthesize a controller program.

In some cases it could not be possible to find the limit of the chain. However there are some technique that can be useful in order to find an approximation of this limit (see [5,9]).

4 A simple example

Consider the process $S = l.\mathbf{0} + h.h.l.\mathbf{0}$. The system S where no high level activity is present is timed weakly bisimilar to $l.\mathbf{0}$. Consider the following equational definition (please note that X_S is a variable here):

$$X_S =_\nu ([\tau]X_S) \wedge [l]\mathbf{T} \wedge \langle\langle l \rangle\rangle \mathbf{T}$$

It asserts that a process may and must perform the visible action l. As for the study of $tBNDC$-like properties we can apply the partial evaluation for the parallel operator we obtain after some simplifications:

$$(X_S)_{//S} =_\nu ([\tau](X_S)_{//S}) \wedge [\overline{h}]\langle\langle \overline{h} \rangle\rangle \mathbf{T}$$

which, roughly, expresses that after performing a visible $\overline{h}$ action, the system reaches a configuration s.t. it must perform another visible $\overline{h}$ action. The information obtained through partial model checking can be used to enforce a security policy which prevents a system from having certain information leaks. In particular, if we use the definition of the controller as $\triangleright''$, we simply need to find a process that is a model for the previous formula, say $Y = \overline{h}.\overline{h}.\mathbf{0}$. Then, for any component X, we have $(S \| (Y \triangleright'' X)) \setminus \{h\}$ satisfies $(X_S)_{//S}$.

For instance, consider $X = \overline{h}.\mathbf{0}$. The system

$$(S \| (Y \triangleright'' X)) \setminus \{h\} \xrightarrow{\tau} (h.l.\mathbf{0} \| (\overline{h} \triangleright'' \mathbf{0})) \setminus \{h\}$$

Thus, using the second rule the controller may force to issue another $\overline{h}$ and thus we eventually get

$$(h.l.\mathbf{0} \| (\overline{h} \triangleright'' \mathbf{0})) \setminus \{h\} \xrightarrow{\tau} (l.\mathbf{0} \| (\mathbf{0} \triangleright'' \mathbf{0})) \setminus \{h\} \approx l.\mathbf{0}$$

and so the system still preserves its security since the actions performed by the component X have been prevented from being visible outside. On the contrary, if the controller would not be there would be a deadlock after the first internal action.

5 Discussion on related work

In [19], a preliminary work has been provided that is based on different techniques for automatically synthesizing systems enjoying a very strong security property, i.e., SBSNNI (see [12]). That work did not deal with controllers.

Much of prior works are about the study of enforceable properties and related mechanisms. In [11] authors deal with a safety interface that permits to study if a module is safe or not in a given environment. Here is checked all system, instead in our approach, through the partial model checking function, we are able to monitor only the necessary/untrusted part of the system. This is an advantage of our approach because often not all the system need to be checked (or it is simply not convenient to check it as a whole). Some components could be trusted and one would like to have a method to constrain only the untrusted ones (e.g. downloaded applets). Similarly, it could not be possible to build a reference monitor for a whole distributed architecture, while it could be possible to have it for some of its components.

Schneider in [29] deals with *enforcement mechanisms* able to enforce safety properties. Ligatti and al. in [6,16] have described four different kinds of *security automata*. In these articles there is the idea that safety properties can be enforced and information flow cannot. Starting from these works, in [21,22] security automata have been modeled by process algebra operators. Exploiting satisfiability procedure for μ-calculus and results of process algebra theory, authors are able to synthesize appropriate controller program for a given controller operator. Our work represents an extension of [21,22] because here we enforce also information flow properties. Moreover we study parameterized systems and all the theory is developed for a timed setting.

Also in their paper, Bartoletti, Degano and Ferrari (see [4]) refer to [29] saying that while safety properties can be enforced by an execution monitor, liveness properties cannot. In this paper they deal with the problem of modeling composition of services in the presence of security constraints. In this paper is presented typed extension of the λ-calculus in order to describe services as program expressions, and to compose them under security constraints. In order to enforce safety and liveness properties, they enclose security-critical code in *policy framings*, in particular *safety framings* and *liveness framings*, that enforce respectively safety and liveness properties of execution histories. The analysis is a static analysis that over-approximates behavior *history expressions*.

We use controller synthesis in order to force a system to verify security policy. The synthesis of controllers is, however, studied also in other research areas. We describe here two papers that deal with synthesize of controller in real-time. In [10] the author describes an algorithm for synthesize controller from real-time specification. He presents an algorithm for specified in a subset of the internal temporal logic duration calculus. The synthesized controllers are given as PLC-Automata. These are an abstract representation of a machine that periodically polls the input and has the possibility of measuring time. In [26] the authors tackle the following

problem: given a timed automaton restrict its transition relation in a systematic way so that all remaining behaviors satisfy certain properties. The problem is formulated using the notion of real-time game. A strategy for a given game is a rule that tells the controller how to choose between several possible actions in any game position. A strategy is winning if the controller, by following these rules, always wins (according to a given definition of winning) no matter what the environment does. There is the definition of game automata and the authors gives a relation and, by using it, it is able to define a winning strategy for the game. We are going to study the relationship with our approach.

6 Conclusion and Future Works

We illustrated some results towards a uniform theory for enforcing security properties based on a process calculi and logical formalization of security properties.

With this work, we contribute to extend a framework based on process calculi and partial model checking that has been shown to be very suitable to specify and verify security properties also to the synthesis of secure systems also in a timed setting.

We give a technique to enforce information flow properties by using process algebra controller operators. As a matter of fact we find out sufficient conditions to enforce some properties as $tBNDC$. However, as the Assumption 3.1, is strong we are working to refine our method in order to deal with problem.

Acknowledgement We thank the anonymous referees of ICS06 for valuable comments that helped us to improve this paper.

References

[1] Andersen, H., "Verification of Temporal Properties of Concurrent Systems," Ph.D. thesis, Department of Computer Science, Aarhus University, Denmark (1993).

[2] Andersen, H. R., *Partial model checking*, in: *LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science* (1995), p. 398.

[3] Asarin, E. and C. Dima, *Balanced timed regular expressions.*, Electr. Notes Theor. Comput. Sci. **68** (2002).

[4] Bartoletti, M., P. Degano and G. L. Ferrari, *Enforcing secure service composition.*, in: *CSFW* (2005), pp. 211–223.

[5] Basu, S. and C. R. Ramakrishnan, *Compositional analysis for verification of parameterized systems*, in: *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science **2619** (2003), pp. 315–330.

[6] Bauer, L., J. Ligatti and D. Walker, *More enforceable security policies*, in: I. Cervesato, editor, *Foundations of Computer Security: proceedings of the FLoC'02 workshop on Foundations of Computer Security* (2002), pp. 95–104.

[7] Bradfield, J. C., *On the expressivity of the modal mu-calculus.*, in: *STACS*, 1996, pp. 479–490.

[8] Corradini, F., D. D'Ortenzio and P. Inverardi, *On the relationships among four timed process algebras.*, Fundam. Inform. **38** (1999), pp. 377–395.

[9] Cousot, P. and R. Cousot, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in: *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (1977), pp. 238–252.

[10] Dierks, H., *Synthesising controllers from real-time specifications*, in: *ISSS '97: Proceedings of the 10th international symposium on System synthesis* (1997), pp. 126–133.

[11] Elmqvist, J., S. Nadjm-Tehrani and M. Minea, *Safety interfaces for component-based systems.*, in: R. Winther, B. A. Gran and G. Dahll, editors, *SAFECOMP*, Lecture Notes in Computer Science **3688** (2005), pp. 246–260.

[12] Focardi, R. and R.Gorrieri, *A classification of security properties*, Journal of Computer Security **3** (1997), pp. 5–33.

[13] Gorrieri, R., R. Lanotte, A. Maggiolo-Schettini, F. Martinelli, S. Tini and E. Tronci, *Automated analysis of timed security: a case study on web privacy.*, Int. J. Inf. Sec. **2** (2004), pp. 168–186.

[14] Gorrieri, R. and F. Martinelli, *A simple framework for real-time cryptographic protocol analysis with compositional proof rules*, Sci. Comput. Program. **50** (2004), pp. 23–49.

[15] Hennessy, M. and T. Regan, *A temporal process algebra*, in: *FORTE '90: Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols* (1991), pp. 33–48.

[16] Ligatti, J., L. Bauer and D. Walker, *Edit automata: Enforcement mechanisms for run-time security policies*, International Journal of Information Security **4** (2005), pp. 2–16.

[17] Lowe, G., *Semantic models for information flow*, Theor. Comput. Sci. **315** (2004), pp. 209–256.

[18] Martinelli, F., *Partial model checking and theorem proving for ensuring security properties*, in: *CSFW '98: Proceedings of the 11th IEEE Computer Security Foundations Workshop* (1998).

[19] Martinelli, F., *Towards automatic synthesis of systems without informations leaks*, in: *Proceedings of Workshop in Issues in Theory of Security (WITS)*, 2000.

[20] Martinelli, F., *Analysis of security protocols as* open *systems*, Theoretical Computer Science **290** (2003), pp. 1057–1106.

[21] Martinelli, F. and I. Matteucci, *Modeling security automata with process algebras and related results* (2006), presented at the 6th International Workshop on Issues in the Theory of Security (WITS '06) - Informal proceedings.

[22] Martinelli, F. and I. Matteucci, *Through modeling to synthesis of security automata* (2006), accepted to STM06. To appeare in ENTCS.

[23] Milner, R., *Synthesis of communicating behaviour*, in: *Proceedings of 7th MFCS* (1978).

[24] Milner, R., "Communicating and mobile systems: the π-calculus," Cambridge University Press, 1999.

[25] Müller-Olm, M., *Derivation of characteristic formulae*, in: *MFCS'98 Workshop on Concurrency*, Electronic Notes in Theoretical Computer Science (ENTCS) **18** (1998).

[26] Pnueli, A., E. Asarin, O. Maler and J. Sifakis, *Controller synthesis for timed automata*, in: *Proc. System Structure and Control* (1998).
URL citeseer.ist.psu.edu/asarin98controller.html

[27] R.Focardi, R.Gorrieri and F.Martinelli, *Real-time Information Flow Analysis*, IEEE JSAC (2003).

[28] Ryan, P. Y. A. and S. A. Schneider, *Process algebra and non-interference*, in: *CSFW '99: Proceedings of the 1999 IEEE Computer Security Foundations Workshop* (1999), p. 214.

[29] Schneider, F. B., *Enforceable security policies*, ACM Transactions on Information and System Security **3** (2000), pp. 30–50.

[30] Street, R. S. and E. A. Emerson, *An automata theoretic procedure for the propositional μ-calculus*, Information and Computation **81** (1989), pp. 249–264.

[31] Streett, R. S. and E. A. Emerson, *The propositional mu-calculus is elementary*, in: *Proceedings of the 11th Colloquium on Automata, Languages and Programming* (1984), pp. 465–472.

[32] Ulidowski, I. and S. Yuen, *Extending process languages with time*, in: *AMAST '97: Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology* (1997).

A Technical proofs

Lemma 2.9 *Let E be a finite-state process and let $\phi_{E,\preceq_t}$ be its characteristic formula w.r.t. weak simulation. Let F be a finite-state process s.t. $F \preceq_t E$. We have $E \models \phi_{E,\preceq_t} \Leftrightarrow F \models \phi_{E,\preceq_t}$.*

Proof: In order to prove the following proposition we give the following chain:

$$F \preceq_t E \Leftrightarrow \forall \alpha F \stackrel{\alpha}{\rightarrow} F' \exists E' E \stackrel{\alpha}{\Rightarrow} E' \wedge F' \preceq_t E' \Leftrightarrow$$
$$\forall \alpha F \stackrel{\alpha}{\rightarrow} F' \Rightarrow F' \models \bigvee X_{E'} \Leftrightarrow \forall \alpha F \models [\alpha] \bigvee X_{E'} \Leftrightarrow$$
$$F \models \bigwedge([\alpha](\bigvee X_{E'}))$$

□

Proposition 3.1 *Let E and F be two finite-state processes. If both E and F are weakly time alive, also $E \rhd' F$ is weakly time alive.*

Proof: We want to prove that for all $(E \rhd' F)' \in Der(E \rhd F)$ $(E \rhd' F)' \stackrel{tick}{\Rightarrow}$. E and F are weakly time alive so

- for all $E' \in Der(E)$ $E' \stackrel{tick}{\Rightarrow}$ i.e., $E' \stackrel{\tau*}{\rightarrow} E_1 \stackrel{tick}{\rightarrow} E'' \stackrel{\tau*}{\rightarrow}$
- for all $F' \in Der(F)$ $F' \stackrel{tick}{\Rightarrow}$ i.e., $F' \stackrel{\tau*}{\rightarrow} F_1 \stackrel{tick}{\rightarrow} F'' \stackrel{\tau*}{\rightarrow}$

So $\exists E', F'$ such that $(E \rhd' F)' = E' \rhd' F'$ and, referring to the semantics rules of $\rhd'$ $E' \rhd' F' \stackrel{\tau*}{\rightarrow} E' \rhd' F_1 \stackrel{\tau}{\rightarrow} E_1 \rhd' F_1 \stackrel{tick}{\rightarrow} E'' \rhd' F'' \stackrel{\tau*}{\rightarrow}$ □

Proposition 3.2 *The operator $\rhd'$ enjoys Assumption 3.1*

Proof : We have to prove the following sentence:

$$\text{For every } E \text{ exists an } F \text{ such that: } E \rhd' F \approx_t E.$$

In order to do that, we show that the following relation is a timed weak bisimulation:

$$\mathcal{R} = \{(E \rhd' F, E) \mid E, F \in \mathcal{E} \text{ and } F \text{ is weakly time alive}\}$$

We distinguish two cases: action a and the action $tick$.

- Assume that $(E \rhd' F, E) \in \mathcal{R}$ and $(E \rhd' F) \stackrel{a}{\rightarrow} (E \rhd' F)'$. According to given semantics rules, if the first rule is applied $(E \rhd' F)' = E' \rhd' F'$. Looking at premises of the rule guarantees that exists $E \stackrel{a}{\Rightarrow} E'$. If the second rule is applied $(E \rhd' F)' = E' \rhd' F$. Also in this case in premises there is $E \stackrel{a}{\Rightarrow} E'$. If F performs the action τ $(E \rhd' F)' = E \rhd' F'$. Since we have the reflexive and transitive closure of $\stackrel{\tau}{\rightarrow}$, $E' = E$ by the action τ.
  Assume that $(E, E \rhd' F) \in \mathcal{R}$ and $E \stackrel{a}{\rightarrow} E'$. Using a) rule or b) rule of $\rhd'$, we can have two different options for $(E \rhd' F)'$. In both cases exists $(E \rhd' F)'$ s.t. $(E \rhd' F) \stackrel{a}{\Rightarrow} (E \rhd' F)'$ and $(E', (E \rhd' F)') \in \mathcal{R}$. It is possible to note that in this case τ does not produce any problem.
- Assume that $(E \rhd' F, E) \in \mathcal{R}$ and $(E \rhd' F) \stackrel{tick}{\rightarrow} E' \rhd' F'$. We can note that this transition is possible by application of the first rule, i.e., $E \stackrel{tick}{\rightarrow} E'$ and $F \stackrel{tick}{\rightarrow} F'$. So we have obviously E' such that $E \stackrel{tick}{\Rightarrow} E'$.

Assume that $(E, E \rhd' F) \in \mathcal{R}$ and $E \stackrel{tick}{\rightarrow} E'$. We have to prove that exists $(E \rhd' F)'$ such that $(E \rhd' F) \stackrel{tick}{\Rightarrow} (E \rhd' F)'$. Considering that F is weakly time alive, we consider, without loss of generality, that $F \stackrel{tick}{\rightarrow} F'$ [12]. Applying the first rule we obtain $E' \rhd' F'$.□

Proposition 3.3 *Let E and F be two finite-state processes. If both E and F are weakly time alive, also $E \rhd'' F$ is weakly time alive.*
Proof: It is the same proof of the proposition 3.1. □
Proposition 3.4 *The operator $\rhd''$ enjoys Assumption 3.1.*
Proof:We have to prove the following sentence:
For every E exists an F such that: $E \rhd'' F \approx_t E$. In order to do that, we show that the following relation is a timed weak bisimulation:

$$\mathcal{R} = \{(E \rhd'' F, E) \mid E, F \in \mathcal{E} \text{ and } F \text{ is weakly time alive}\}$$

With similar arguments of the proof of proposition 3.2, we can prove that $E \rhd'' F \approx_t E$.□

[12] In fact, being F w.t.a. we know that for all $F' \in Der(F)$ $F' \stackrel{tick}{\Longrightarrow}$. This means that F may perform a τ action a certain number of time. This is not a problem because it is sufficient to apply the third rule of $\rhd''$ as much time as the number of τ actions performed.