Synthesis of local controller programs for enforcing global security properties [*]

Fabio Martinelli
Istituto di Informatica e Telematica-C.N.R., Pisa, Italy
Fabio.Martinelli@iit.cnr.it

Ilaria Matteucci
Istituto di Informatica e Telematica-C.N.R., Pisa, Italy
Dipartimento di Scienze Matematiche ed Informatiche,R. Magari-Università degli Studi di Siena
Ilaria.Matteucci@iit.cnr.it

Abstract

In this paper we present a framework based on contexts theory *and* logic *to study how, given a* partially specified system, i.e.*, a system in which there are some unspecified/ unknown components,* i.e.*, potential attackers, it is possible to enforce a global* security property *by controlling all the unspecified parts of the given system. We propose two methods to control them: A* centralized *method, in which there is a unique controller program that controls all the unspecified components, and a* decentralized *one in which each unspecified component is monitored by a controller program that forces it to behave correctly,* i.e.*, according to a local requirement found by a reduction of the global one. In both cases we show how to synthesize controller programs that solve the problem.*

1. Overview

In this paper we study how to guarantee that a *partially specified distributed system*, *i.e.*, a system in which there are some components whose behavior is unknown, is secure, *i.e.*, behaves as prescribed by a *security property* that specifies acceptable executions of a program. In particular we deal with a specific kind of security properties, the so called *safety properties* that states that "nothing bad happens".

In order to model a given partially unspecified system we recall the approach based on *context* proposed in [9]. As a matter of fact a partially specified system is modeled as a context $C(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are variables that denote unspecified components of the system.

In order to express safety properties we use a modal specification language (see [9]) that is a version of the Hennessy-Milner logic (see [8]) extended with the ability to define properties recursively.

There are a lot of significant safety properties that can be expressed by using modal specification language. For instance, *access-control policies* are safety properties since once a restricted resource has been accessed, the policy is broken. There is no way to "unaccess" the resource and fix the situation afterward. Another example of safety property that can be expressed in this logic is the *Chinese Wall policy*. It is an example of a global security properties in a distributed system.

Referring to [6, 10], our goal is describing a method to guarantee that a given distributed system with unspecified parts X_i is secure. Each unspecified component X_i is studied as a potential malicious agent. We would like to guarantee that, regardless the behavior of possible intruders, the whole system still works properly, *i.e.*, it respects the required safety property. As a matter of fact we would like to guarantee the following property:

$$\forall X_1, \ldots, X_n \quad C(X_1, \ldots, X_n) \models \phi$$

Since it is not possible to check all possible X_i behavior, in order to guarantee the previous property holds we define *controller contexts*, denoted by $\triangleright$, that allow to monitor the behavior of the unspecified part of the system by *controller programs*. In particular, we present two different strategies (centralized and decentralized approach) to use controller contexts and in both cases we show how to synthesize *controller programs* for satisfying the security requirements.

In the centralized approach, we wonder if there exists a *controller program* Y, *i.e.*, an implementation that acts as a controller, s.t.:

$$\forall X_1, \ldots, X_n \; C(\triangleright(Y, X_1 \times \ldots \times X_n)) \models \phi \qquad (1)$$

[*]Work partially supported 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).

where $\times$ denotes the product of contexts. So we synthesize an unique controller program Y that manages all the unspecified components of the system by considering the product of all X_i, $i = 1, \dots, n$, as the unique unspecified component of the system.

In order to obtain the controller program Y, we firstly evaluate the behavior of the context into the formula ϕ by the *property transformer* (see [9]). In this way we obtain a new formula ϕ' that has to be satisfied only from the unknown part of the system. As a matter of fact, the property transformer permits us to isolate which are the necessary and sufficient conditions the unknown components have to satisfy in order to make the system secure. Yet, we synthesize the appropriate controller program Y by exploiting a satisfiability procedure for temporal logic.

In the decentralized method, the given global security property is decomposed in an equivalent product of simpler formulas. We require that each unspecified component satisfies one formula of the product in order to guarantee that the whole system satisfies the global security property. For that reason we synthesize several controller programs, one for each unknown component. As a matter of fact we wonder if there exist n controller programs $Y_1, \dots, Y_n$ s.t.:

$$\forall X_1, \dots, X_n \; C(\triangleright(Y_1, X_1), \dots, \triangleright(Y_n, X_n)) \models \phi \qquad (2)$$

Also in this case we firstly evaluate the behavior of the context in the formula ϕ by the *property transformer* and we obtain a new formula ϕ'. According to [9], we can decompose the formula ϕ' as a *product formula*, $\phi_1 \times \dots \times \phi_n$. If each component X_i satisfies a formula ϕ_i of the product then we can conclude that globally the system is secure.

Hence our goal is to synthesize for each component X_i a controller program Y_i s.t. $Y_i \triangleright X_i \models \phi_i$ for all $i = 1, \dots, n$. Also in this case we exploit a satisfiability procedure for temporal logic to obtain each Y_i.

It is important to note that an advantage of our framework is that in both our approaches, even if we are able to enforce a global property, we control only the possible untrusted components and not the global system. Note that this is a crucial distinction that makes our approach more effective since usually it is not practically possible to control the whole distributed system, but only some of its critical components.

This paper is organized as follows. Section 2 presents some related works on distributed control. Section 3 recalls the theory developed by Larsen and XinXin in [9] on compositionality through an operational semantics on contexts by showing a modal language and the *property transformer* function. Section 4 shows how it is possible to synthesize both centralized and decentralized controller programs in order to guarantee that a system satisfies a global safety property. Section 5 presents a simple example and Section 6 concludes the paper.

2. Related Works

In this paper we propose a framework based on contexts and logic to deal with the problem of the synthesis of controllers for secure systems by exploiting centralized/ decentralized controller programs. In [11, 12, 13] we presented a framework based on process algebra, partial model checking and logic in order to synthesize a controller program for a system with only one possible malicious component. Also in [1] the authors have presented a method based on satisfiability procedure and game theory to synthesize controller for discrete events systems. They generate a controller for the whole system. On the contrary, in [11, 12, 13], we synthesize controllers that work by monitoring only the possible untrusted component of the system. Moreover they do not addressed any security analysis, *i.e.*, they synthesize controller for a given process that must be controlled. On the contrary we synthesize controllers that make the system secure for whatever behavior of unknown components. Our controller are synthesized without any information about the process they are going to control. In this work we deal with systems in which there are several unspecified components by applying the theory of compositionality by contexts proposed in [9]. In particular we show different control approaches.

Other works deal with partially specified system using context. In [7] is presented a framework inspired by [9] for the validation of reactive system embedded in a test environment, or isolated from their operational environment, thereby inducing a natural classification of validation strategies in different scenario. However they do not deal with security property and do not make security analysis. Moreover the authors consider context with only one unknown component, instead in our work we present results on contexts in which more than one component of the system is unspecified.

In [15], contexts are used to build *upgrade specification* from components and their interface languages. No security problem is addressed.

A lot of works deal with the problem of decentralized discrete-event control problems, as [2, 3, 4, 17, 18] the authors have studied the decentralized supervisory control problem of discrete event systems under partial observation. They do not treat this problem from a security point of view. As a matter of fact they do not do security analysis by considering generating controller for whatever possible behavior of the unspecified part, *i.e.*, they do not consider the unspecified part of the system as a potential attacker. In [4, 17, 18] the authors have investigated on the necessary and sufficient conditions for the existence of decentralized supervisors for ensuring that the controlled behavior of the system lies in a given range. The problem of the synthesis is not addressed.

On the other hand, in [2] the problem of the synthesis of controllers is studied. They start from their previous work, [3], in which they deal with the decentralized control problem of several communicating supervisory controllers, each with different information, that work in concert to exactly achieve a given legal sublanguage of the uncontrolled system's language model. In [2] the author presents a procedure for finding an optimal communication policy, if one exists, for a class of finite controllers. In our work, in addition to explicit untrusted components, we deal with the problem of the distributed control but we consider independent controllers, *i.e.*, we present a method to synthesize decentralized controllers that work independently one each other on different unspecified components of the system in such a way the whole system is secure.

It is worth also to mention approaches that directly try to build correct systems (rather than controlling potentially incorrect ones as we do). For instance in [5] the authors present an automatic synthesis procedure for distributed system having a flexible specification language and a reasonable computational complexity. They use *asynchronous automata*.

3. Composition through context

We recall here an operational theory of *context* in terms of *action transducers* as defined in [9].

First of all we give the definition of context as follows.

Definition 3.1 *A* context *system $\mathcal{C}$ is a structure*

$$\mathcal{C} = (\langle C_n^m \rangle_{n,m}, Act, \langle \rightarrow_{n,m} \rangle_{n,m})$$

where C_n^m is a set of n-to-m contexts; Act is a set of actions; $Act_0 = Act \cup \{0\}$ where $0 \notin Act$ is a distinguished no-action symbol, Act_0^k is a tuple of k actions $\in Act_0$, and $\rightarrow_{n,m} \subseteq C_n^m \times (Act_0^n \times Act_0^m) \times C_n^m$ is the transduction-relation for the n-to-m contexts satisfying $(C, \bar{a}, \bar{0}, D) \in \rightarrow_{n,m}$ iff $C = D$ and $\bar{a} = \bar{0}$ for all contexts $C, D \in C_n^m$.

For $(C, \bar{a}, \bar{b}, C') \in \rightarrow_{n,m}$ we usually write $C \overset{\bar{b}}{\underset{\bar{a}}{\rightarrow}} C'$, leaving the indices of $\rightarrow$ to be determine by the context, and we interpret this as "by consuming the action $\bar{a}$, the context C can produce the actions $\bar{b}$ and change in C'".

The operational semantics of contexts is consistent with the existing operational semantics of processes (see [16]). For instance, let C be a context $\in C_n^m$. If there are n components P_i, $i = 1, \ldots, n$, and $P_i \xrightarrow{a_i} P_i'$, is a transition of the component P_i, *i.e.*, the component P_i performs the action labeled by a_i in order to transit in P_i', then C can *consume* the actions $a_1 \ldots a_n$ while *producing* action $b_1 \ldots b_m$ and changing into C'. Thus, the context $C(P_1, \ldots, P_n)$ has the transition $C(P_1, \ldots, P_n) \xrightarrow[a_1 \ldots a_n]{b_1 \ldots b_m} C'(P_1', \ldots, P_n')$. Dually, any transition of $C(P_1, \ldots, P_n)$ ought to be derivable in this way.

In particular the set of 0-to-1 contexts C_0^1 are just processes and C_n^1 are normal n-hole contexts.

Example 3.1 *CCS process algebra (see [14]) can be seen as a context system with the following contexts:* prefix *$a^* \in C_1^1$ for $a \in Act$,* restriction *$\backslash L \in C_1^1$ where $L \subseteq Act$.* Choice *and* parallel *context $+, \| \in C_2^1$;* inactive *$\mathcal{O}_n^m \in C_n^m$ for any n and m, with Nil that denotes the context $\mathcal{O}_0^1$. There are also the* identity *context $I_n \in C_n^n$ and the* projection *$\Pi_n^i \in C_n^1$. The semantics definition of CCS context is in Table 1.*

Inaction: $C \overset{\bar{0}}{\underset{\bar{0}}{\rightarrow}} C$ for all C Prefix: $a^* \overset{a}{\underset{\bar{0}}{\rightarrow}} I_1$

Restriction: $\backslash_L \overset{a}{\underset{a}{\rightarrow}} \backslash_L \quad a \notin L$

Choice: $(1)\ + \xrightarrow[(a,0)]{a} \Pi_1^1 \quad (2)\ + \xrightarrow[(0,a)]{a} \Pi_2^1$ for $a \in Act$

Projection: $\Pi_n^i \xrightarrow[i(a)]{a} \Pi_n^i$ Identity: $I_n \overset{\bar{a}}{\underset{\bar{a}}{\rightarrow}} I_n$

Parallel: $(1)\ \| \xrightarrow[(a,\hat{a})]{\tau} \| \quad (2)\ \| \xrightarrow[(a,0)]{a} \| \quad (3)\ \| \xrightarrow[(0,a)]{a} \|$

where $i(a) \in Act_0^n$ with the i^{th} component being a and all the others being 0.

Table 1. Semantics of CCS context system.

3.1 Operations between contexts

Different operations between contexts can be defined. First of all, we show how the behavior of two contexts can be compared by recalling the following definition of *simulation* and *bisimulation* equivalence (see [14]).

Definition 3.2 *Let $\mathcal{C} = (\langle C_n^m \rangle_{n,m}, Act, \langle \rightarrow_{n,m} \rangle_{n,m})$ be a context system. Then an n-to-m* simulation *$\mathcal{R}$ is a binary relation on C_n^m s.t., whenever $(C, D) \in \mathcal{R}$ and $\bar{a} \in Act_0^n$, $\bar{b} \in Act_0^m$, then the following holds:*

$$\textit{if } C \overset{\bar{b}}{\underset{\bar{a}}{\rightarrow}} C', \textit{ then } D \overset{\bar{b}}{\underset{\bar{a}}{\rightarrow}} D' \textit{ for some } D' \textit{ with } (C', D') \in \mathcal{R}.$$

We write $C \prec D$ in case $(C, D) \in \mathcal{R}$ for some n-to-m simulation $\mathcal{R}$.

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

Composition. Contexts can be composed. $C(P_1, \ldots, P_n)$ is a composed context. In order to facilitate *composition*, it is allowed that contexts produce a number m of actions, w.r.t. the consumption of n actions, with, possibly, $n \neq m$. Moreover, in order to cater for *asynchronous* contexts, it is not required that all the components $P_1, \ldots, P_n$ contribute in a transition of the combined process $C(P_1, \ldots, P_n)$.

Definition 3.3 *Let $\mathcal{C} = (\langle C_n^m \rangle_{n,m}, Act, \langle \rightarrow_{n,m} \rangle_{n,m})$ be a context system. A* composition *on $\mathcal{C}$ is a dyadic operation $\circ$ on contexts such that whenever $C \in C_n^m$ and $D \in C_m^r$ then $D \circ C \in C_n^r$. furthermore, the transductions for a context $D \circ C$ with $C \in C_n^m$ and $D \in C_m^r$ are fully characterized by the following rule:*

$$\frac{C \overset{\bar{b}}{\underset{\bar{a}}{\rightarrow}} C' \quad D \overset{\bar{c}}{\underset{\bar{b}}{\rightarrow}} D'}{D \circ C \overset{\bar{c}}{\underset{\bar{a}}{\rightarrow}} D' \circ C'}$$

where $\bar{a} = (a_1, \ldots, a_n)$ is a vector of actions.

Products. In order to represent a *partial specification* of a system, *i.e.*, a system with n holes, we use an n-to-1 context $C \in C_n^1$. To allow the expansion of the n holes to be carried out independently, it is defined an *independent combination* of n contexts as $D_1 \times \ldots \times D_n$.

Definition 3.4 *Let $\mathcal{C} = (\langle C_n^m \rangle_{n,m}, Act, \langle \rightarrow_{n,m} \rangle_{n,m})$ be a context system. A* product *on $\mathcal{C}$ is a dyadic operation $\times$ on contexts, s.t. whenever $C \in C_n^m$ and $D \in C_r^s$ then $C \times D \in C_{n+r}^{m+s}$. Furthermore the transduction for a context $C \times D$ are fully characterized by the following rule:*

$$\frac{C \overset{\bar{b}}{\underset{\bar{a}}{\rightarrow}} C' \quad D \overset{\bar{d}}{\underset{\bar{c}}{\rightarrow}} D'}{C \times D \overset{\bar{b}\bar{d}}{\underset{\bar{a}\bar{c}}{\rightarrow}} C' \times D'}$$

where juxtaposition of vectors $\bar{a} = (a_1, \ldots, a_n)$ and $\bar{c} = (c_1, \ldots, c_r)$ is the vector $\bar{a}\bar{c} = (a_1, \ldots, a_n, c_1, \ldots, c_r)$.

Feed-back. In order to deal with *recursion*, a construction of *feed-back* on contexts is defined, s.t. whenever $C \in C_n^n$ then $C^\dagger \in C_0^n$ with the behavioral equation $C^\dagger \sim C \circ C^\dagger$ being satisfied. Formally, we have the following definition.

Definition 3.5 *Let $\mathcal{C} = (\langle C_n^m \rangle_{n,m}, Act, \langle \rightarrow_{n,m} \rangle_{n,m})$ be a context system. A* feed-back *on $\mathcal{C}$ is a unary operation †, on contexts of $\mathcal{C}$ s.t., whenever $C \in C^{n,n}$ then $C' \in C_0^n$. Furthermore, the transduction for a context $C^\dagger$ with $C \in C_n^n$ is fully characterized by the rule:*

$$\frac{C \overset{\bar{b}}{\underset{\bar{a}}{\rightarrow}} C' \quad C^\dagger \xrightarrow{\bar{a}} D}{C^\dagger \xrightarrow{\bar{b}} C' \circ D}$$

For this operations on contexts the following result holds.

Theorem 3.1 ([9]) $\sim$ *is preserved by composition, product and feed-back of context.*

By following a reasoning similar to the one are made in [9] to prove the previous theorem, it is possible to note that the same result holds also for $\prec$.

3.2. Modal language and property transformer for contexts

In order to express properties of contexts we consider the modal specification language introduced in [9].

Definition 3.6 *Let L be a set of labels (or actions) and let V be a set of variables. The* formulas *and* declarations *over V relative to L, $\mathcal{F}_{V,L}$ and $\mathcal{D}_{V,L}$ are built up according to the following abstract syntax*

$$\begin{aligned} &\phi ::= \mathbf{T} | \mathbf{F} | X | \phi_1 \wedge \phi_2 | \phi_1 \vee \phi_2 | \langle a \rangle \phi | [a]\phi | \\ &\text{LET MAX } D \text{ IN } \phi | \text{ LET MIN } D \text{ IN } \phi \\ &D ::= X_1 = \phi_1 \ldots X_n = \phi_n \end{aligned}$$

The above logic is a propositional modal logic with $\langle a \rangle \phi$ and $[a]\phi$ providing the two relativized modalities. The declaration in the LET-constructs, introduces simultaneous recursively specified properties. The concepts of free and bound variables are defined as usual; in particular we call a formula *closed* if it contains no free variables. We shall use standard notation $\phi[\psi/X]$ to describe the substitution of ψ for all free occurrences of the variable X in ϕ.

$$\begin{aligned} &[\![\mathbf{T}]\!]_\rho = \Gamma \quad [\![\mathbf{F}]\!]_\rho = \emptyset \quad [\![X]\!]_\rho = \rho(X) \\ &[\![\phi_1 \vee \phi_2]\!]_\rho = [\![\phi_1]\!]_\rho \cup [\![\phi_2]\!]_\rho \; [\![\phi_1 \wedge \phi_2]\!]_\rho = [\![\phi_1]\!]_\rho \cap [\![\phi_2]\!]_\rho \\ &[\![\langle a \rangle \phi]\!]_\rho = \{\gamma \in \Gamma | \exists \gamma' : \gamma \xrightarrow{a} \gamma' \wedge \gamma' \in [\![\phi]\!]_\rho\} \\ &[\![[a]\phi]\!]_\rho = \{\gamma \in \Gamma | \forall \gamma' : \gamma \xrightarrow{a} \gamma' \Rightarrow \gamma' \in [\![\phi]\!]_\rho\} \\ &[\![\text{LET MAX} D \in \phi]\!]_\rho = [\![\phi]\!](D_\nu [\![D]\!]_\rho) \\ &[\![\text{LET MIN} D \in \phi]\!]_\rho = [\![\phi]\!](D_\mu [\![D]\!]_\rho) \\ &\text{and} \\ &\quad D_\nu [\![X_1 = \phi_1 \ldots X_n = \phi_n]\!]_\rho = \nu\rho' \\ &\qquad \rho\{[\![\phi_1]\!]_{\rho'} \backslash X_1, \ldots, [\![\phi_n]\!]_{\rho'} \backslash X_n\} \\ &\quad D_\mu [\![X_1 = \phi_1 \ldots X_n = \phi_n]\!]_\rho = \mu\rho' \\ &\qquad \rho\{[\![\phi_1]\!]_{\rho'} \backslash X_1, \ldots, [\![\phi_n]\!]_{\rho'} \backslash X_n\} \end{aligned}$$

Table 2. Semantics clauses.

The interpretation of the introduced logic is given relative to *Labeled Transition System* that is a structure $(\Gamma, L, \rightarrow)$, where Γ is a set of states and L is a set of actions and $\rightarrow \subseteq \Gamma \times L \times \Gamma$ is the *transition relation*. The interpretation of a closed formula is given as the set of states *satisfying* the formula. The semantics of formulas is given w.r.t. an environment $\rho : V \rightarrow \mathcal{P}(\Gamma)$. The semantics definition is given inductively on the structure of formulas and declaration as in Table 2, with ν and μ being the *maximum* respectively *minimum* fixed point operator.

For this logic the following theorem holds.

Theorem 3.2 ([19]) *Given a formula ϕ it is possible find a model of ϕ in exponential time in the length of ϕ.*

Example 3.2 *By using this logic it is possible to define several security properties,* e.g., *the* Chinese Wall *policy. This policy says that, let A and B two sets of elements. Once one accesses to an element in A, he cannot access to B and viceversa. Here we consider that A and B are sets of files and we consider the action* open. *This can be expressed by the formula $\phi = \phi_1 \vee \phi_2$ where ϕ_1 and ϕ_2 are the following two formulas respectively:*

$$\begin{array}{lll}\phi_1 = \text{LET MAX} & W = & [open_A]W \wedge [open_B]\mathbf{F}\text{IN}W \\ \phi_2 = \text{LET MAX} & V = & [open_B]V \wedge [open_A]\mathbf{F}\text{IN}V\end{array}$$

As a matter of fact ϕ is a disjunction between two different formulas ϕ_1 and ϕ_2 that cannot be both true at the same time. Indeed ϕ_1 permits to open only file in A, on the other hand ϕ_2 allows the access to elements in B.

Property transformer function. According to the definition in [9] it is possible to define the property transformer function $\mathcal{W}$ for contexts as in Table 3. The property transformer is introduced in order to isolate which is the necessary and sufficient conditions that the unspecified part of the system has to satisfy.

$$\begin{array}{l}
\mathcal{W}(C,\mathbf{T}) = \mathbf{T} \quad \mathcal{W}(C,\mathbf{F}) = \mathbf{F} \quad \mathcal{W}(C,X) = X^C \\
\mathcal{W}(C,\phi_1 \wedge \phi_2) = \mathcal{W}(C,\phi_1) \wedge \mathcal{W}(C,\phi_2) \\
\mathcal{W}(C,\phi_1 \vee \phi_2) = \mathcal{W}(C,\phi_1) \vee \mathcal{W}(C,\phi_2) \\
\mathcal{W}(C,\langle \bar{b} \rangle \phi) = \bigvee_{C \overset{\bar{b}}{\underset{\bar{a}}{\to}} D} \langle \bar{a} \rangle \mathcal{W}(D,\phi) \\
\mathcal{W}(C,[\bar{b}]\phi) = \bigwedge_{C \overset{\bar{b}}{\underset{\bar{a}}{\to}} D} [\bar{a}] \mathcal{W}(D,\phi) \\
\mathcal{W}(C,\text{LET MAX } D \text{ IN } \phi) = \text{ LET MAX } D^T \text{ IN } \mathcal{W}(C,\phi) \\
\mathcal{W}(C,\text{LET MIN } D \text{ IN } \phi) = \text{ LET MIN } D^T \text{ IN } \mathcal{W}(C,\phi) \\
\text{where} \\
D^T = \{X^C = \mathcal{W}(C,\phi) | C \in C_n^m, X = \phi \in D\}
\end{array}$$

Table 3. Definition of property transformer.

The following theorem holds.

Theorem 3.3 ([9]) *Let $\mathcal{C} = (\langle C_n^m \rangle_{n,m}, Act, \langle \rightarrow_{n,m} \rangle_{n,m})$ be a context system. Let ϕ be a closed formula and let $C \in C_n^m$. Then for any $Q \in C_0^n$ the following equivalence holds:*

$$C(Q) \models \phi \Leftrightarrow Q \models \mathcal{W}(C,\phi)$$

Here we recall an example in order to explain how the property transformer works.

Example 3.3 *We consider again the formula that expresses the Chinese Wall policy given in the Example 3.2. We consider a distributed system $S = \|(X_1, X_2)$. Hence we calculate $\phi' = \mathcal{W}(\|,\phi)$. Let us consider the following abbreviation:*

$$\begin{array}{ll} W^{\|} = \mathcal{W}(\|, W) & V^{\|} = \mathcal{W}(\|, V) \\ \phi_1' = \mathcal{W}(\|, \phi_1) & \phi_2' = \mathcal{W}(\|, \phi_2) \end{array}$$

Then we obtain $\phi' = \phi_1' \vee \phi_2'$ where ϕ_1' and ϕ_2' are the following:

$$\begin{array}{l}
\phi_1' = \text{LET MAX}W' = [(0, open_A)]W' \wedge [(open_A, 0)]W' \\
\wedge[(0, open_B)]\mathbf{F} \wedge [(open_B, 0)]\mathbf{F})\text{IN}W' \\
\phi_2' = \text{LET MAX}V' = [(0, open_B)]V' \wedge [(open_B, 0)]V' \\
\wedge[(0, open_A)]\mathbf{F} \wedge [(open_B, 0)]\mathbf{F}\text{IN}V'
\end{array}$$

3.3. Decomposition of property

According to [9], in order to deal with the problem of decomposition of a joint property into properties that can be satisfied by the components $P_1, \ldots, P_n$ of a context, the set of formulas $\mathcal{F}^n$ is extended with *product formulas* of the form $\phi_1 \times \ldots \times \phi_n$ where $\phi_1, \ldots, \phi_n$ are closed, unary formulas, *i.e.*, they belong to $\mathcal{F}^i$. This extended set is denoted by $\mathcal{F}_\times^n$. The semantics of $\mathcal{F}_\times^n$ is obtained by adding to the clauses of Table 2 the following rule:

$$Q \in [\![\phi_1 \times \ldots \times \phi_n]\!]_\rho \Leftrightarrow \exists P_1 \in [\![\phi_1]\!] \ldots \exists P_n \in [\![\phi_n]\!] \quad Q \sim P_1 \times \ldots \times P_n$$

For a product formula $\phi \in \mathcal{F}_\times^n$, we write $\models \phi$ if ϕ is satisfied by all n-ary context system. In this case ϕ is *valid*. Moreover $\models_\times \phi$ if ϕ is satisfied by all n- product process $P_1 \times \ldots \times P_n$ of any context system. In this case ϕ is *weakly valid*.

We wonder if there exists a single product formula $\phi_1 \times \ldots \times \phi_n$ s.t. $\models_\times \phi_1 \times \ldots \times \phi_n \Leftrightarrow \mathcal{W}(C,\phi)$. However usually there is not a unique possible decomposition. Whenever ϕ is a *finite* formula, *i.e.*, it is neither LET MAX nor LET MIN, there exists a finite decomposition, *i.e.*, a finite collection of product formulas $\langle \phi_1^i \times \ldots \times \phi_n^i \rangle_{i \in I}$, where I is and index set, s.t.

$$\models_\times \bigvee_{i \in I} \phi_1^i \times \ldots \times \phi_n^i \Leftrightarrow \mathcal{W}(C,\phi)$$

The following theorem holds. [1]

Theorem 3.4 ([9]) *Let ϕ be a finite formula. Then there exists a weakly equivalent disjunctive product formula $\bigvee_{i \in I} \psi^i \times \varphi^i$ s.t.*

$$\models_\times \phi \Leftrightarrow \bigvee_{i \in I} \psi^i \times \varphi^i$$

[1] The interested reader can find all the theory in [9].

4. Synthesis of controllers

We study how and when it is possible to enforce safety properties in a distributed system by centralized/decentralized monitors. As a matter of fact we consider a partially specified system in which more than one component behavior is unknown. We represent systems like this as context in C^1_n where n is the number of unspecified components of the system.

Our problem is to guarantee that a given partially specified system satisfies a safety property expressed as a modal logic formula ϕ whatever the behavior of the unspecified components is, *i.e.*,

$$\forall X_1, \ldots, X_n \quad C(X_1, \ldots, X_n) \models \phi \tag{3}$$

To do this, we introduce *controller context*, denoted by $\triangleright \in C^n_{2n}$, that forces the system to behave correctly, *i.e.*, as prescribed by ϕ. It acts on two components, in particular it combines a *controller program* $Y \in C^n_0$ and an unknown component $X \in C^n_0$ in such a way Y forces X to behave correctly.

It is possible to give several semantics definitions for $\triangleright$. It depends on which kind of properties we are going to enforce or in which way we want to enforce them (*e.g.*, [11, 12, 13]).

A possible semantics definition of the controller context $\triangleright(Y, X)$ is the following:

$$\frac{Y \xrightarrow{\bar{b}} Y' \quad X \xrightarrow{\bar{b}} X'}{\triangleright(Y,X) \xrightarrow[(\bar{b},\bar{b})]{\bar{b}} \triangleright(Y', X')} \tag{4}$$

This means that $\triangleright$ works with specified context in such a way if Y and X do not perform the same actions than the execution halts.

4.1. Two approaches for enforcing safety properties in a distributed system

A controller context can be applied in different ways in order to solve the problem in Formula (3).

Centralized method. We synthesize a unique controller program Y that enforces ϕ by monitoring the product of all components as a unique context. The Formula (3) becomes:

$$\exists Y \quad \forall X_1, \ldots, X_n \quad C(\triangleright(Y, X_1 \times \ldots \times X_n)) \models \phi \tag{5}$$

where, being each X_i is in C^1_0, $X_1 \times \ldots \times X_n$ is in C^n_0. First of all, by applying the property transformer function, we find the weakest condition the unknown component has to satisfy in order to guarantee the whole system satisfies ϕ. Hence the problem of Formula (5) becomes as follows:

$$\exists Y \quad \forall X_1, \ldots, X_n \quad \triangleright(Y, X_1 \times \ldots \times X_n) \models \phi' \tag{6}$$

where $\phi' = \mathcal{W}(C, \phi)$. Now we want to synthesize Y. Looking at the semantics definition of $\triangleright$, it is possible to prove that the following simulation relation holds:

$$\triangleright(Y, X) \prec Y$$

Let Fr_μ be a sublanguage of the modal language in Section 3.2 in which there are not the diamond formulas. This class of formulas expresses safety properties. It is easy to prove that Fr_μ formulas is closed under the property transformer function. Moreover, following a reasoning similar to the one made in [12] for processes it is possible to give the following result.

Proposition 4.1 *Let E and F be two contexts and $\phi \in Fr_\mu$. If $F \prec E$ then $E \models \phi \Rightarrow F \models \phi$.*

At this point in order to satisfy the Formula (6) it is sufficient to find a controller program Y s.t.:

$$Y \models \phi'$$

Since $Y \in C^n_0$ it performs a vector of actions. Without lost of generality, we consider that each transition is labeled by a vector as $\bar{a} = (a_1, \ldots, a_n)$ and we synthesize Y by Theorem 3.2. So we are able to prove the following result.

Theorem 4.1 *The problem described in Formula (5) is decidable.*

Decentralized method. We synthesize several controller contexts Y_i, one for each unspecified components X_i of the system. Hence our main problem can be formalized as follows:

$$\begin{gathered}\exists Y_1, \ldots, Y_n \; \forall X_1, \ldots, X_n \\ C(\triangleright(Y_1, X_1), \ldots, \triangleright(Y_n, X_n)) \models \phi\end{gathered} \tag{7}$$

In this case all Y_i are processes. By exploiting the property transformer on context, we can reduce the previous problem as follows:

$$\begin{gathered}\exists Y_1, \ldots, Y_n \; \forall X_1, \ldots, X_n \\ \triangleright(Y_1, X_1) \times \ldots \times \triangleright(Y_n, X_n) \models \mathcal{W}(C, \phi)\end{gathered} \tag{8}$$

For the class of finite formulas, by applying the Theorem 3.4, the problem in (8) becomes

$$\begin{gathered}\exists Y_1, \ldots, Y_n \; \forall X_1, \ldots, X_n \\ \triangleright(Y_1, X_1) \times \ldots \times \triangleright(Y_n, X_n) \models_\times \bigvee_{i \in I} \psi^i_1 \times \ldots \times \psi^i_n\end{gathered} \tag{9}$$

We have to choose which formula of the disjunction we are going to enforce, *i.e.*, we reduce the problem in Formula (9) as follows:

$$\begin{gathered}\exists Y_1, \ldots, Y_n \; \forall X_1, \ldots, X_n \\ \triangleright(Y_1, X_1) \times \ldots \times \triangleright(Y_n, X_n) \models_\times \psi^k_1 \times \ldots \times \psi^k_n\end{gathered} \tag{10}$$

We can solve the problem above by solving n problems that have the following form:

$$\exists Y_j \; \forall X_j \; \triangleright(Y_j, X_j) \models \psi_j^k$$

We are able to synthesize n controller programs able to enforce safety properties, *i.e.*, properties described as formulas of Fr_μ, by exploiting the Theorem 3.2. As a matter of fact we are in a case similar to the previous one. Instead to generate a controller program $Y \in C_0^n$ that control the product of n components, we generate n controller programs in C_0^1 each of them controls only one component X_i. So we are able to prove the following result.

Theorem 4.2 *The problem described in Formula (7) is decidable.*

5. An Example: The *Chinese Wall* policy

Now we present how we enforce the *Chinese Wall* policy, that is a very common security property, in a distributed system. We consider as distributed system $S = \|(X_1, X_2)$.

As in the Example 3.2 the Chinese Wall policy is expressed by the formula $\phi = \phi_1 \vee \phi_2$ where

$$\begin{array}{lll} \phi_1 = \textsc{let max} & W = & [open_A]W \wedge [open_B]\mathbf{F}\textsc{in}W \\ \phi_2 = \textsc{let max} & V = & [open_B]V \wedge [open_A]\mathbf{F}\textsc{in}V \end{array}$$

Hence, as in the Example 3.3, we calculate $\phi' = \mathcal{W}(\|, \phi) = \phi'_1 \vee \phi'_2$ as follows:

$$\begin{array}{l} \phi'_1 = \textsc{let max} W' = [(0, open_A)]W' \wedge [(open_A, 0)]W' \\ \wedge[(0, open_B)]\mathbf{F} \wedge [(open_B, 0)]\mathbf{F})\textsc{in}W' \\ \phi'_2 = \textsc{let max} V' = [(0, open_B)]V' \wedge [(open_B, 0)]V' \\ \wedge[(0, open_A)]\mathbf{F} \wedge [(open_B, 0)]\mathbf{F}\textsc{in}V' \end{array}$$

Now we synthesize a controller program for enforcing ϕ in both cases.

Centralized control. In this case we generate a controller program $Y \in C_0^2$ s.t. it is a model for the formula ϕ' and a controller program for $\triangleright(Y, X_1 \times X_2)$. We take

$$\begin{array}{lll} Y & = + & (Y_1, Y_2) \\ Y_1 & = + & ((0, open_A)^*, (open_A, 0)^*)^\dagger \\ Y_2 & = + & ((0, open_B), (open_B, 0)^*)^\dagger \end{array}$$

It is possible to note that such Y, at the beginning, permits whatever possible behavior of unspecified components. Indeed both X_1 and X_2 are allowed to perform $open_A$ or $open_B$ action. However, after the first step, only one behavior is allowed. Let us consider, for instance, $X_1 = (open_A)^* \circ X_2$ and $X_2 = (open_B)^* \circ X_1$. In this case Y_1 becomes $((open_A, 0)^*)^\dagger$ and $Y_2 = ((0, open_B)^*)^\dagger$ since the other choice case never happens. Hence $Y = +(((open_A, 0)^*)^\dagger, ((0, open_B)^*)^\dagger)$. Let us consider that the first step is performed by X_2 then we have the following derivation tree:

$$\frac{\triangleright \xrightarrow[(0,\, open_B,\, 0,\, open_B)]{(0, open_B)} \triangleright \qquad \| \xrightarrow[(0,\, open_B)]{open_B} \|}{\| \circ \triangleright \xrightarrow{open_B} \| \circ \triangleright}$$

Hence:

$$\|(\triangleright(Y, X_1 \times X_2)) \xrightarrow{open_B} \|(\triangleright(Y_2, X_1 \times X_1))$$

Looking at the transition rule of $\triangleright$ we can note that, at the beginning, both the possibility, executing the action $open_A$ as well as executing the action $open_B$, are allowed. Since the first step is performed by X_2, the action $open_B$ is done. Hence the controller program chooses the component Y_2 that allows only $open_B$ actions. However, after the transition, the target system can perform only action $open_A$ so the system halts.

Decentralized control. It is easy to note that the formula ϕ' is not finite. Hence a finite decomposition for it may not exists. In this case it may be difficult to find a formula to enforce.

Moreover is that the formula ϕ' requires a strict interaction between processes. As a matter of fact, the behavior of the second context is conditioned by the behavior of the first one and viceversa. Indeed if the first context performs an action $open_A$ ($open_B$) the second cannot performs an action $open_B$ ($open_A$) and viceversa. For this reason it is not possible to find two independent controller operators that enforce the Chinese Wall policy in a distributed way at runtime.

On the other hand, we could enforce the Chinese Wall policy in a distributed way by establishing a priori that one of the two context must not perform any action. For instance we can consider the following two controller programs:

$$Y_1 = Nil \qquad \begin{array}{lll} Y_2 & = + & (Y'_2, Y''_2) \\ Y'_2 & = & (open_A^*)^\dagger \\ Y''_2 & = & (open_B^*)^\dagger \end{array}$$

In this way, the first unknown component cannot perform any action and the second has to respect the Chinese wall policy. For instance, let $X_1 = (open_A)^* \circ X_2$ and $X_2 = (open_B)^* \circ X_1$ the we have that $\triangleright(Nil, X_1)$ is equivalent to the context Nil because does not perform any action. Hence the system is $\|(Nil, (\triangleright(Y_2, X_2)))$. Thus we have the following transition:

$$\|(Nil, (\triangleright(Y_2, X_2))) \xrightarrow[(0,\, open_B)]{(open_B)} \|(Nil, (\triangleright(Y''_2, X_1)))$$

since, according to the semantic definition of $\triangleright$, the transduction of $\triangleright(Y_2, X_2)$ is the following:

$$\triangleright(Y_2, X_2) \xrightarrow[(open_B,\, open_B)]{(open_B)} \triangleright(Y''_2, X_1)$$

The execution halts because the second contexts, by calling the first one, tries to performs an action $open_A$ that is forbidden.

6. Conclusion

In this paper we have studied how it is possible to guarantee that a distributed system with more than one unspecified component is secure by using controllers. We describe two different methods to do this, a centralized one and a decentralized one, for enforcing safety property. In the first one all component are combined and considered as a unique component and we synthesize a controller program that forces such combination to behave correctly. In the second approach we have studied when it is possible to synthesize local controller context that, by monitoring the behavior of the local component, enforces a given formula that has to be satisfied locally in order to guarantee that a global security property is satisfied by the whole system.

Acknowledgement. We thank the anonymous referees of APE08 for valuable comments that helped us to improve this paper.

References

[1] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[2] G. Barrett and S. Lafortune. On the synthesis of communicating controllers with decentralized information structures for discrete-event systems. In *Proceedings of the 37th IEEE international conference on Decision and Control*, volume 3, pages 3281–3286, December 1998.

[3] G. Barrett and S. Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Trans. Automatic Control*, 45(9):1620–1638, 2000.

[4] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete event processes with partial observations. *IEEE, Trans. Automat. Contr.*, 33:249–260, 1988.

[5] A. Ştefănescu. Automatic synthesis of distributed systems. In *ASE '02: Proceedings of the 17th IEEE international conference on Automated software engineering*, page 315, Washington, DC, USA, 2002. IEEE Computer Society.

[6] R. Focardi, R. Gorrieri, and F. Martinelli. Classification of security properties - part ii: Network security. In R. Focardi and R. Gorrieri, editors, *FOSAD*, volume 2946 of *Lecture Notes in Computer Science*, pages 139–185. Springer, 2002.

[7] L. Heerink and E. Brinksma. Validation in context. In *PSTV*, pages 221–236, 1995.

[8] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[9] K. G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, Dec. 1991.

[10] F. Martinelli. Analysis of security protocols as open systems. *Theoretical Computer Science*, 290(1):1057–1106, 2003.

[11] F. Martinelli and I. Matteucci. An approach for the specification, verification and synthesis of secure systems. *Electr. Notes Theor. Comput. Sci.*, 168:29–43, 2007.

[12] F. Martinelli and I. Matteucci. Through modeling to synthesis of security automata. *Electr. Notes Theor. Comput. Sci.*, 179:31–46, 2007.

[13] I. Matteucci. Automated synthesis of enforcing mechanisms for security properties in a timed setting. *Electr. Notes Theor. Comput. Sci.*, 186:101–120, 2007.

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

[15] M. Müller-Olm, B. Steffen, and R. Cleaveland. On the evolution of reactive components: A process-algebraic approach. In *FASE '99: Proceedings of the Second Internationsl Conference on Fundamental Approaches to Software Engineering*, pages 161–175, London, UK, 1999. Springer-Verlag.

[16] G. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI-FN-19, Aarhus University, 1981.

[17] P. Ramadge and W. Whoman. Supervision of discrete event processes. In *Proceedings of 21th IEEE conference Decision Contr.*, volume 3, pages 1228–1229, December 1982.

[18] K. Rudie and W. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE transactions on automatic control*, 37(11):1692–1708, Nov. 1992.

[19] R. S. Street and E. A. Emerson. An automata theoretic procedure for the propositional μ-calculus. *Information and Computation*, 81(3):249–264, 1989.