Decentralized Intrusion Detection For Secure Cooperative Multi–Agent Systems

Adriano Fagiolini, Gianni Valenti, Lucia Pallottino, Gianluca Dini and Antonio Bicchi

Abstract— **In this paper we address the problem of detecting faulty behaviors of cooperative mobile agents. A novel decentralized and scalable architecture that can be adopted to realize a monitor of the agents' behavior is proposed.**

We consider agents which may perform different independent tasks, but cooperate to guarantee the entire system's safety. Agents plan their next actions by following a set of rules which is shared among them. Such rules are decentralized, i.e. they depend only on configurations of neighboring agents. Some agents may not be acting according to this cooperation protocol, due to spontaneous failure or tampering. To detect such misbehaviors we propose a solution where each agent runs a local observer using only locally available information.

The objective of the work is the definition of a basic framework to automatically realize decentralized intrusion detectors for (hybrid) multi–agent systems where interaction is modeled through logical cooperative protocols.

I. Introduction

Multi–agent systems are more and more often employed to obtain inherently robust solutions to many robotic applications, such as exploration, surveillance, patrolling, target tracking, and intelligent transportation. In such situations, agents may perform different and possibly independent tasks, but they cooperate in order to guarantee the entire system's safety. Cooperation among agents is obtained through a shared set $\mathcal{R}$ of rules according to which all agents are supposed to plan their actions. We will refer to such rules as the cooperation protocol. Cooperative decentralized systems can often be modeled as hybrid systems whose dicrete states represent actions while guards among states depend on the configuration of neighboring agents.

We are particularly interested in decentralized control strategies for autonomous vehicles that decide on their motion based only on the configurations and velocities of neighboring vehicles, and where the main safety concern is collision avoidance. Several collision avoidance strategies for multi–agent systems have been proposed in the literature with different application domains and different sets of decentralized rules (see e.g. [1]–[4])

While in the literature the benefits of decentralized traffic management protocols are often underscored, few authors have recently highlighted the threats posed by so–called "intelligent collisions" [5]. As a matter of fact, whenever one or more agents fail to follow the common set of rules, due e.g. to spontaneous failure, tampering, or even to malicious behaviors [6].

The goal of an Intrusion Detection System (IDS) for decentralized cooperative multi–agent policies is to automatically detect possible misbehaviors, using only the information locally available to each agent, along with the knowledge of the cooperation protocol. The construction of such an IDS can build upon a rich literature on the detection of failures in Discrete Event Systems (DES) in the presence of partial observations [7]–[9]. Most methods in the literature are based on the supervisory control concepts introduced in the seminal works of Ramadge and Wonham [10], [11]. Notions of diagnosability, observability, invertibility and observability with delays for centralized and decentralized for DES have been introduced [12]–[16]. For hybrid systems, the notions of diagnosability, observability, and fault detection have been introduced more recently [17]–[21]. However, the literature in hybrid systems fault detection is much poorer, due to the intrinsic complexity of hybrid systems.

In this paper we consider mobile agents with given dynamics $\dot{q} = f(q, u)$, where u is a continuous function that drives the agent in executing its maneuvers. We assume that the protocol $\mathcal{R}$ is assigned, and that only a finite number of possible maneuvers are allowed by $\mathcal{R}$ for the agents. Based on the protocol definition, u may depend also on the current configuration $\mathbf{q}$ of agents in the environment. We also assume that, while the cooperative protocol is completely known to each agent, only a partial knowledge of the guards between nodes of the hybrid system is available, due to decentralization. Hence, $\mathbf{q}$ is not completely known to each agent, and will be decomposed in a known (observable) and an unknown (unobservable) part ${\mathbf{q}_i}^o$ e ${\mathbf{q}_i}^u$ for the i–th agent.

In the literature related to fault detection for DES, failure

A. Fagiolini, G. Valenti, L. Pallottino and A. Bicchi are with the Interdepartmental Research Center "E. Piaggio", Faculty of Engineering, University of Pisa, Italy, {a.fagiolini, l.pallottino, bicchi}@ing.unipi.it, posta@gianni.valenti.name.

G. Dini is with the Dipartimento dell'Informazione, Faculty of Engineering, University of Pisa, Italy, gianluca.dini@ing.unipi.it.

is typically modeled as a state. Hence, reachability techniques can be used to detect the failure or the diagnosability of the system. In our setting, failures correspond to agents arbitrarily misbehaving. The goal of an agent acting as a decentralized IDS is to distinguish a faulty or malicious agent in its neighborhood from a correctly cooperating agent whose actions may be influenced by other agents out of the monitor's range. Furthermore, the fact that the topology of interaction and exchange of information among mobile agents is changing and unknown, should be taken into account. These reasons make the problem we deal with quite distinct from those tackled in the current DES and hybrid systems literature, and indeed a very challenging one.

Our approach, which explicitly deals with the system's hybridness, consists in modeling the unknown states of agents outside the decentralized IDS's range as a disturbance to the model of each neighbor under monitoring, and in constructing an Unknown Input Observer (UIO) for such systems to reconstruct if possible the guards which have triggered certain behaviours.

II. Agent's Hybrid Architecure

In this section we describe the architecture of an agent which plans its actions, e.g. decides on its next motion, in accordance with a completely decentralized cooperative decision scheme. The architecture itself is independent of the particular agent's dynamics, and of the particular set $\mathcal{R}$ of rules describing the interaction among the agents.

Let $q_i \in \mathcal{Q}$ be a vector describing the "physical" state of the i–th agent and taking value in the configuration space $\mathcal{Q}$. Let also

$$\dot{q}_i = f_i(q_i, u_i)\,,$$

be its dynamics where $u_i \in \mathcal{U}_i$ is the control input. For the sake of simplicity, we assume all the agents to have the same generic dynamics, i.e. $f_i = f$ for all i. Then, let

$$u_i = g_i(q_i, \sigma_i)\,,$$

be a controller generating the input u_i such that the i–th agent's dynamics executes the trajectory $q_i(t)$ corresponding to the action, e.g. the motion maneuver, specified by the command σ_i. Due to the above hypothesis, all the controllers are the same: $g_i = g$ for all i.

Furthermore, let $\sigma_i \in \Sigma$ be a symbol representing the action planned by the i–th agent's supervisor, and let $\mathcal{S}_i$ be the corresponding decision making process. We will refer to $\mathcal{S}_i$ also as the agent's decision model. In the general case, the action σ_i at the generic decision time t_k is obtained as:

$$\sigma_i(t_k) = \mathcal{S}_i\left(\sigma_i(t_{k-1}), \mathbf{q}(t_k)\right)\,,$$

where $\sigma_i(t_{k-1})$ is the action planned at the previous decision time t_{k-1}, $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ is the state of the entire system, and n is the overall number of agents.

In a decentralized setting, the supervisor $\mathcal{S}_i$ is unaware of the overall number n of agents as well as of any other global information about the system, and indeed the decision making process requires only local information. Bearing this in mind, denote by ${\mathcal{Q}_i}^a$ the space of all active configurations for agent i, representing those configurations that actually affect the agent's behavior. This space is uniquely determined by the cooperation rules in $\mathcal{R}$, and can be formally defined as:

$${\mathcal{Q}_i}^a = \{q \in \mathcal{Q} \,|\, R(q_i, q)\}\,,$$

where R is a Boolean function that ensues from $\mathcal{R}$. Indeed, the structure of this set is inherently defined after assigning the decentralized cooperation rules.

Let $N_i(t)$ be the time–varying set representing the i–th agent's actual neighborhood, being the set formed by the indices of the agents actually affecting the decision making process at the current time t. Formally we have:

$$N_i = \{j \in J \,|q_j \in {\mathcal{Q}_i}^a\}\,,$$

where $J = \{1, 2, \cdots, n\}$ is the set of indices of all existing agents in the system. Examples of neighborhood N_i are given by the set of agents lying within a fixed distance, or in line of sight from the agent i. Furthermore, let $n_i = \mathrm{card}(N_i)$ be the number of agents cooperating with the i–th agent, and let $\mathcal{N}_i$ be the state of the neighborhood N_i:

$$\mathcal{N}_i = \{q_j \in Q \,|\, j \in N_i\}\,.$$

Then the decision making process $\mathcal{S}_i$ only depends on the neighborhood's state $\mathcal{N}_i$:

$$\mathcal{S}_i\left(\sigma_i(t_{k-1}), \mathbf{q}(t_k)\right) = \mathcal{S}_i\left(\sigma_i(t_{k-1}), \mathcal{N}_i(\mathbf{q}(t_k))\right)\,.$$

Although we consider generic decentralized cooperation rules, we assume that the decision model $\mathcal{S}_i$ can be split into two parts: an event detector $\mathcal{E}_i$, that checks the occurrence of enabled events e_i based on the agent's neighborhood $\mathcal{N}_i$, and a finite state machine (automaton) $\mathcal{A}_i$, whose state represents the agent's current action σ_i and evolves in accordance with the events observed by $\mathcal{E}_i$:

$$\mathcal{S}_i = \begin{cases} e_i(t_k) = \mathcal{E}_i\left(\mathcal{N}_i\left(\mathbf{q}\left(t_k\right)\right)\right)\,, \\ \sigma_i(t_k) = \mathcal{A}_i\left(\sigma_i\left(t_{k-1}\right), e_i\left(t_k\right)\right)\,. \end{cases}$$

In particular, automaton $\mathcal{A}_i$ is defined as the 4–tuple $(\Sigma_i, E_i, \Gamma_i, \delta)$, where $\Gamma_i(\sigma_i)$ is the set of events represented by edges originating from node σ_i, and δ is the discrete state transition function.

As it is highlighted in Fig. 1, the agent's architecture is hybrid. Indeed all of its components can be arranged

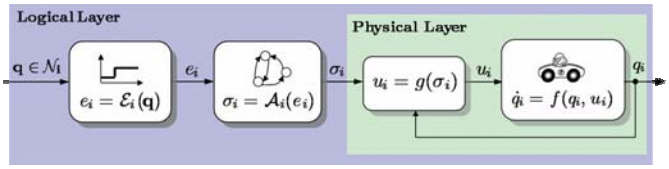

Fig. 1. Depiction of the agent's hybrid architecture that is formally represented by the hybrid system $\mathcal{H}_i = \{f_i(\cdot), g_i(\cdot), \mathcal{A}_i(\cdot), \mathcal{E}_i(\cdot)\}$.

in two layers: a time–driven physical layer, composed of the agent's dynamics f_i and the low–level controller g_i, and an event–driven logical layer, composed of the event–detector $\mathcal{E}_i$ and the automaton $\mathcal{A}_i$. Note finally that the block g_i acts as a converter from event–driven to time–driven dynamics, whereas the block $\mathcal{E}_i$ does the reverse process. Moreover, the block g_i allows us to abstract the complex agent's evolution to that of a discrete event system.

To conclude, we formally define an agent as the hybrid system:

$$\mathcal{H}_i = \{f_i(\cdot), g_i(\cdot), \mathcal{A}_i(\cdot), \mathcal{E}_i(\cdot)\}$$

where $f_i : Q_i \times \mathcal{U}_i \to Q_i$, $g_i : Q_i \times \Sigma \to \mathcal{U}_i$, $\mathcal{A}_i : \Sigma \times E_i \to \Sigma$, $\mathcal{E}_i = Q^{N_i} \to E_i$, $E_i = \{\text{true}, \text{false}\}^{\nu_i}$, and ν_i is the number of enabled events.

III. From Cooperation Rules To An Event–Driven Supervisory System

In this section we show how the i–th agent's supervisory system $\mathcal{S}_i$ is built based on the set $\mathcal{R}$ of cooperation rules. For the sake of generality, each agent may be assigned with a different kind of task — even though it cooperates with its neighbors —, and therefore it may have to use a different set $\mathcal{R}_i$ of rules. However, we focus on collaborative systems where every agent cooperates sharing the relevant subset of rules, and let $\mathcal{R} = \cap_i \mathcal{R}_i$ for all i.

The cooperation rules define all possible actions that can be executed by every agent, and at the same time they describe logical conditions on the state of the agent's neighborhood precisely stating when the agents themselves have to change their current action. Therefore, let $\Sigma = \{\sigma^1, \sigma^2, \ldots, \sigma^\kappa\}$ be the set of κ symbols representing all possible actions, and let $E_i = \{e_i^1, e_i^2, \ldots, e_i^{\nu_i}\}$ be the set of ν_i guards representing such logical conditions. Then, the i–agent's supervisory system $\mathcal{S}_i$ can be obtained as a discrete event system whose state σ_i, taking value on the set Σ, represents the agent's current action. The state itself is updated in accordance with input events $e_i \in E_i$ generated by the agent's neighborhood. In formula we have:

$$\sigma_i(t_k) = \mathcal{S}_i(\sigma_i(t_{k-1}), e_i(t_k))\,.$$

Although general cooperation rules may specify also temporal constraints on the execution of each action, we restrict

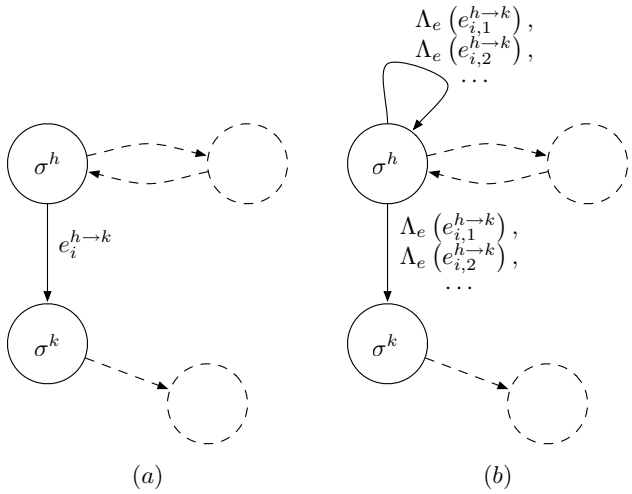

Fig. 2. Depiction of a generic automaton (a): nodes represent different supervisory actions, and edges represent transitions triggered by logical input events. Depiction of the predictor automaton (b).

for simplicity with logical rule sets where no temporization is present. In such a case, a supervisor $\mathcal{S}_i$ can be formally represented as a finite state machine (automaton). See for reference Fig. 2–a where nodes represent different actions, and edges represent transitions between nodes labelled with the input events that trigger such transitions. More precisely, input event $e_i^{h\to k}$ encodes the logical condition on the state of the agent's neighborhood requiring supervisor $\mathcal{S}_i$ to update its state σ_i from action σ^h to action σ^k.

The obtained supervisor is capable of coping with any neighborhood $\mathcal{N}_i$. Nevertheless, at a particular instant, each agent considers a decision model where the set E_i of possible events is computed w.r.t. the current neighborhood. By doing so, the agent virtually decomposes all events $e_i^{h\to k}$ as the disjunction of sub–events,

$$e_i^{h\to k} = \bigvee_l e_{i,l}^{h\to k}(\mathbf{q}_l)\,, \tag{1}$$

where is $\mathbf{q}_l \in 2^{\mathcal{N}_i}$, and $e_{i,l}^{h\to k}$ is the l–th sub–event depending only on variable $\mathbf{q}_l$ and requiring $\mathcal{S}_i$ to update its state σ_i from action σ^h to action σ^k. Note finally that all these logical expressions are mutually exclusive, and thus represent actual events.

IV. Detecting Non–Cooperative Behaviors Under Partial Knowledge

In this section we introduce the problem of detecting non–cooperative agents, referred to as intruders. To this aim, we first define the physical behavior of agent i as the trajectory $q_i(t)$ of its time–driven dynamics during any

observation period $[t_{k-1}, t_k]$. Then, we consider the generic agent h — the observer — trying to establish whether the neighboring agent i is acting according to the rules in $\mathcal{R}$ or not. For the sake of space, we will omit the case of an omniscent observer, i.e. an agent having complete knowledge of the i–th agent's neighborhood state $\mathcal{N}_i$. Instead, we directly consider an observer with exact but partial knowledge of $\mathcal{N}_i$, since this case already encompasses the previous one.

Under this circumstance, the observer may be unable to measure the time–driven state q_j of some of the agents laying within neighborhood N_i. This can happen either when they are too distant and out of sensors' range, or when they are masked by other agents or obstacles. In this perspective, observer h can split the configuration space $\mathcal{Q}$ of any agent as the union of an observable space ${\mathcal{Q}_h}^o$, and an unobservable space ${\mathcal{Q}_h}^u$ ($\mathcal{Q} = {\mathcal{Q}_h}^o \cup {\mathcal{Q}_h}^u$), and consequently the state of N_i in a known part ${\mathbf{q}_i}^o$ and an unknown part ${\mathbf{q}_i}^u$:

$$\mathbf{q}_i = ({\mathbf{q}_i}^o, {\mathbf{q}_i}^u)\,. \tag{2}$$

Denote with O_h the set of observable agents being those agents in the observable space ${\mathcal{Q}_h}^o$ of observer h:

$$O_h = \{j \in J \,|\, q_j \in {\mathcal{Q}_h}^o\}\ .$$

To verify the correctness of any observed behavior $q_i(t)$, the observer itself has to compute the set $\hat{q}_i(t)$ of all behaviors expected from agent i according to the only available information. To achieve this, first the number n_i of agents actually interacting with agent i must be inferred, and then, for each of these agent, their induced input sequence must be reconstructed. In this vein, our problem requires first a model identification, and then an unknown–input observation for a hybrid system.

Furthermore, let $\hat{n}_i$ be the observer h's estimate of the number of agents interacting with agent i. It should be clear that such a number has always to satisfy the following relation:

$$0 \leq \hat{n}_i - \mathrm{card}(O_h \cap N_i) \leq \Psi({\mathcal{Q}_i}^a \cap {\mathcal{Q}_h}^u)\,, \tag{3}$$

where $\Psi : \mathcal{Q} \to \mathbb{Z}$ is an application returning the maximum number of agents that can physically lay in the given configuration space according to geometric constraints induced from $\mathcal{R}$. The proposed approach for estimating n_i consists of trying to use, at any time, the minimum value satisfying the inequalities in 3, and allowing to explain the observed sequence of behaviors q_i. Whenever the process fails, the number of presumed interacting agents is increased by a unit — and also $\hat{n}_i$ — thus considering a richer cooperation model. Along the same line of [14], each cooperation models is able to conclude that the agent is correct or "faulty if no–one says non–faulty". When all possible models fail, then a failure is detected, and the agent is deemed to be an intruder performing non cooperative actions.

V. The Observer's Architecture

In this section we specifically deal with the construction of the observer provided that $\hat{n}_i$ is properly initialized and changed, as described above, whenever a fault is detected.

Owing to the decomposition in Eq. 2, every sub–event $e_{i,l}$ of Eq. 1 can be seen as the conjunction of a decidable part ${e_{i,l}}^o$, depending only on ${\mathbf{q}_i}^o$, with an undecidable part ${e_i}^u$, depending also on ${\mathbf{q}_i}^u$. Therefore we have:

$${e_{i,l}}(\mathbf{q}_i) = {e_{i,l}}^o({\mathbf{q}_i}^o) \wedge {e_{i,l}}^u({\mathbf{q}_i}^o, {\mathbf{q}_i}^u)\,.$$

Furthermore, let ${E_i}^o = \{\epsilon, {e_{i,1}}^o, {e_{i,2}}^o, \cdots\}$ be the set of observable parts, and let ${E_i}^u = \{\oslash, {e_{i,1}}^u, {e_{i,2}}^u, \cdots\}$ be the set of all unobservable parts. Symbols $\oslash$ and ϵ are necessary to represent completely observable and unobservable events, respectively. Then, the set E_i of all input events e_i acting on automaton $\mathcal{A}_i$ can be represented as a subset of the Cartesian product between ${E_i}^o$ and ${E_i}^u$. In formula we have:

$$E_i \subseteq {E_i}^o \times {E_i}^u\,.$$

Along the same line of [22], it is straightforward to formally define an event observation map $\Lambda_e : E_i \to {E_i}^o$ as the natural projection of any event of the original alphabet E_i into the observable alphabet ${E_i}^o$:

$$\Lambda_e(e_{i,l}) = \Lambda_e({e_{i,l}}^o \wedge {e_{i,l}}^u) = {e_{i,l}}^o\,.$$

Accordingly, application ${\Lambda_e}^{-1} : {E_i}^o \to 2^{E_i}$ is the map returning the set of events $\hat{e}$ obtained as the inverse projection of any observable part ${e_{i,l}}^o$. Hence we have:

$$\hat{e} = \{e_i \in E_i\ s.t.\ \Lambda_e(e) = {e_{i,l}}^o\} = {\Lambda_e}^{-1}(e_{i,l})\,.$$

Another useful definition is the following. Given a deterministic automaton $\mathcal{A} : E \to \Sigma$, we define the inverse automaton $\mathcal{A}^{-1} : \Sigma \to 2^E$ as the (non–deterministic) system receiving a sequence of discrete state σ, and generating, at any instant, the set $\hat{e}$ of all events that can explain such sequence according to the following relation:

$$\hat{e}(t_k) = \{e \in \Gamma(\sigma(t_{k-1})) \,|\, \delta(\sigma(t_{k-1}), e) = \sigma(t_k)\}\ .$$

To predict the set $\hat{\sigma}_i$ of all possible future discrete states of the observed agent, we introduce a further component $\mathcal{P}_i$, referred to as the predictor automaton, encoding all the observer's undecidability due to its partial knowledge of $\mathcal{N}_i$. This automaton is in general non–deterministic, and can be systematically obtained from the original automaton $\mathcal{A}_i$ as follows (see Fig. 2-b). First, the automaton $\mathcal{P}_i$ contains the same set of nodes of $\mathcal{A}_i$. Then, for every edge $e_{i,l}^{h \to k}$

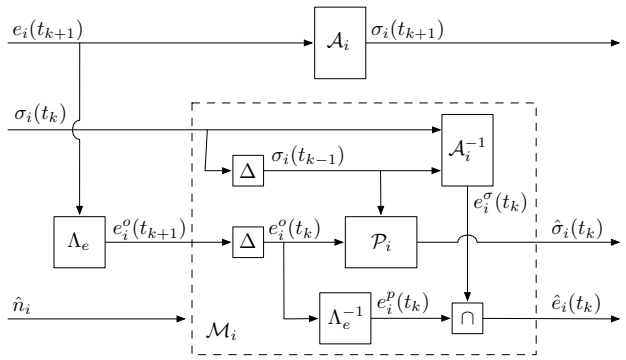

Fig. 3. Block $\mathcal{M}_i$ generates the set of all possible discrete state through the predictor automaton $\mathcal{P}_i$, and estimates the unknown input $\hat{e}_i$.

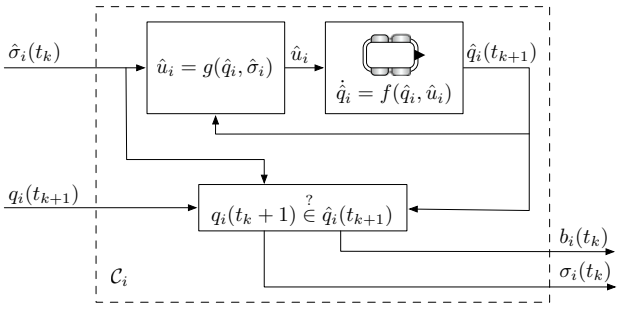

Fig. 4. The classifier $\mathcal{C}_i$ is a block detecting the actual agent's discrete state, through the calculation of the physical behavior, and checking the agent's cooperativeness.

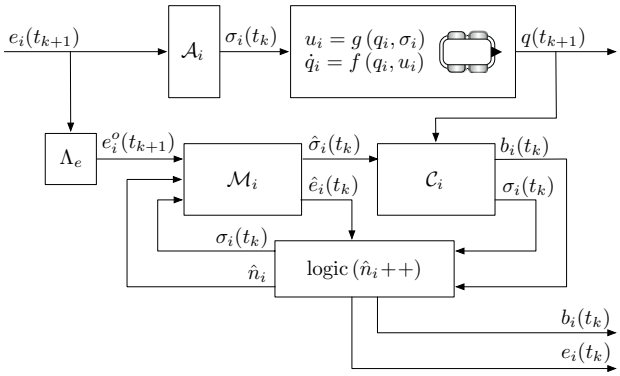

Fig. 5. The decentralized detection scheme exploiting blocks $\mathcal{M}_i$ and $\mathcal{C}_i$ in conjunction with the block logic.

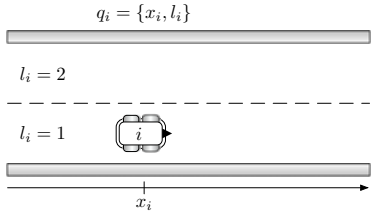

Fig. 6. A 2–lane automated highway with vehicles that are supposed to move according to the common driving rules.

connecting node σ^h with node σ^k in $\mathcal{A}_i$, insert in $\mathcal{P}_i$ an edge assigned with label $\Lambda_e(e_{i,l}^{h\to k})$, and connecting the same two nodes. Moreover, to take into account for the undecidability of the value of $e_{i,l}^{h\to k}$, insert also a self–loop with the same label around node σ^h. Hence, the predictor automaton is defined as the following application:

$$\mathcal{P}_i : {E_i}^o \to \Sigma_i \, .$$

It is worth noting that $\mathcal{P}_i$ reduces to the deterministic automaton $\mathcal{A}_i$ if the observer $\mathcal{O}_h$ has complete knowledge of $\mathcal{N}_i$.

Misbehaviors of agent i can then be detected by using the two components in Fig. 3 and 4, respectively. In particular, system $\mathcal{M}_i$ is a filter receiving as inputs the observable part ${e_{i,l}}^o$ of any event and the actual discrete state σ_i of the observed agent, and generating a prediction $\hat{\sigma}_i$ of possible future discrete states and an estimate of the complete $\hat{e}_i$. Afterward, system $\mathcal{C}_i$ first runs a multiple execution of the physical layer, in order to compute the set of all possible behaviors $\hat{q}_i(t)$, and then checks whether the actual behavior $q_i(t)$ of the agent was predicted or not. If so, it returns the actual previous discrete state $\sigma_i(t_k)$, and sets b_i to $\mathrm{correct}$, otherwise to faulty and raises an alarm. The two components are combined as in Fig. 5 where $''\mathrm{logic}''$ is a block representing the strategy according to which $\hat{n}_i$ is updated.

VI. Case Study – An Automated Highway

In this section we illustrate the use of the proposed decentralized intrusion detection scheme by the application to an example of multi–agent system consisting of n vehicles that are travelling along a 2–lane automated highway. Each vehicle may enter the highway in different positions, have different maximum velocity, and be assigned with a different destination. Referring to Fig. 6, the physical state of vehicle i is represented by $q_i = (x_i, l_i)$ where x_i represents its position along the current lane l_i. In order to avoid collisions, each vehicle is supposed to plan its motion in accordance with the common driving rules (the set $\mathcal{R}$ of Sec. III). Within such a scenario, we want each vehicle to be able to establish a subjective reputation b_i of its neighbors based only on its partial knowledge of the system.

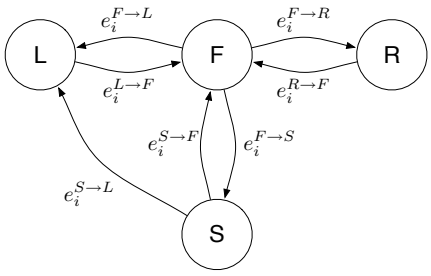

Fig. 7. The deterministic automaton $\mathcal{A}_i$ of the supervisor for any neighborhood N_i.

TABLE I

INPUT EVENTS TO THE AUTOMATON $\mathcal{A}_i$ FOR ANY NEIGHBORHOOD N_i.

$$
\begin{aligned}
e_i^{F\to L} &= (\exists j \in N_i \; s.t. \; x_j - x_i \le d, x_j \ge x_i, l_j = l_i) \wedge \\
&\wedge \; (\nexists k \in N_i, k \ne j \; s.t. \; |x_k - x_i| \le d, l_k > l_i) \wedge \\
&\wedge \; (l_i \ne 2) \; , \\
e_i^{F\to S} &= (\exists j \in N_i \; s.t. \; x_j - x_i \le d, x_j \ge x_i, l_j = l_i) \wedge \\
&\wedge \; (\exists k \in N_i, k \ne j \; s.t. \; |x_k - x_i| \le d, l_k > l_i) \vee \\
&\vee \; (\exists j \in N_i \; s.t. \; x_j - x_i \le d, x_j \ge x_i, l_j = l_i) \wedge (l_i = 2) \; , \\
e_i^{F\to R} &= (\nexists j \in N_i \; s.t. \; |x_j - x_i| \le d, l_i > l_j) \wedge \\
&\wedge \; (l_i \ne 1) \; , \\
e_i^{L\to F} &= (l_i = 2) \; , \\
e_i^{R\to F} &= (l_i = 1) \; , \\
e_i^{S\to L} &= e_i^{F\to L} \; , \\
e_i^{S\to F} &= (\nexists j \in N_i \; s.t. \; x_j - x_i \le d, x_j \ge x_i, l_j = l_i) \; .
\end{aligned}
$$

A. Modeling of supervisor $\mathcal{S}_i$

The model of the i–th agent's supervisory system $\mathcal{S}_i$ can be obtained as in Sec. III, and is given by the deterministic automaton of Fig. 7. More precisely, having defined with σ_i its discrete state, the supervisor $\mathcal{S}_i$ is supposed to choose at any instant among one of the following maneuvers: fast (F), left (L), right (R), and slow (S). The choice of the current maneuver σ_i depends on the logical conditions (guards) of Table I, where we have used x_j and l_j as short–hands for x_{i_j} and l_{i_j}, being the state coordinates of the neighboring vehicle of index i_j.

Furthermore, the active configuration space ${\mathcal{Q}_i}^a$ of agent i (see Sec. II) can be geometrically characterized as a function of q_i and the parameter d, and it is highligthed in Fig. 8. Then, the number n_i of vehicles that can at any time affect the behavior of agent i satisties the following relation:

$$0 \le n_i \le \Psi({\mathcal{Q}_i}^a) \, ,$$

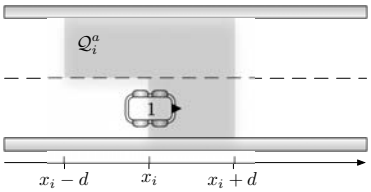

Fig. 8. The active configuration space ${\mathcal{Q}_i}^a$ of agent i depends on its state q_i, and on the parameters of the cooperation rules.

TABLE II

SIGNALS IN BLOCK $\mathcal{M}_i$ WHEN THE OBSERVABLE REGION IS THAT OF FIG. 10 (FROM A TO D).

	$\hat{n}_i = 0$	$\hat{n}_i = 1$
$\mathcal{A}_i^{-1}(F, L)$	$\emptyset$	$\{a_1\}$
$\hat{\boldsymbol{\sigma}}_i(e_1^o, F)$	$\{F\}$	$\{F, L\}$
$\Lambda_e^{-1}(e_1^o)$	$\emptyset$	$\{a_1, c_1, f_1\}$

where Ψ has been defined in Sec. V.

B. Modeling of observer $\mathcal{O}_h$

Let $\mathcal{O}_h$ be the observer assigned with agent h aiming at deciding whether agent i is cooperative or not. Its model can be obtained as in Sec. V. Furthermore, from agent h's point of view, the configuration space is partitioned in an observable region ${\mathcal{Q}_h}^o$ and an unobservable region ${\mathcal{Q}_h}^u$. Hence, as described in Sec. IV, observer h will predict all the maneuvers of agent i that are consistent with the partially observed environment. As a matter of fact, the known part of agent i's neighborhood N_i is $O_h \cap N_i$.

VII. SIMULATION

We now show how the proposed scheme works through the following example where $k \in \mathbb{N}$ is an index representing at any time the number of observed events.

Simulation starts with observer h approaching to vehicle 1 which is currently performing a fast maneuver ($\sigma_i = F$). From agent h's point of view, the configuration space is partitioned as depicted in Fig. 10-a, and for the number n_i of interacting agents, satisfying inequality 3, must hold: $n_i \in [0, 1]$. Therefore the estimate number $\hat{n}_i$ is initialized with 0 since observer h sees no other vehicle than 1. Under such a hypothesis, the predictor automaton $\mathcal{P}_i$ of Sec. V only admits the maneuver set $\hat{\sigma}_i = \{F\}$ (see first automaton in Fig. 9, and first column in Table II) reported in Fig. 10-b. Hence the behavior of agent i can be explained, and we have: $b_i = \text{correct}$.

Assume now that agent i changes to a left maneuver ($\sigma_i = L$) as in Fig. 10-c. Since $\hat{n}_i = 0$ does not provide

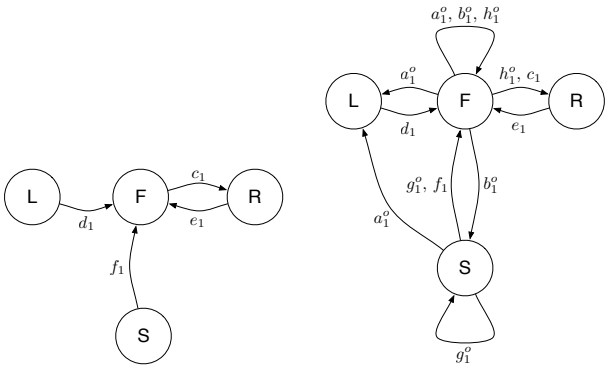

Fig. 9. Explicit construction of the predictor automaton $\mathcal{P}_i$ for the cases $n_i = 0$ (left) and $n_i = 1$ (right).

TABLE III

SIGNALS IN BLOCK $\mathcal{M}_i$ WHEN THE OBSERVABLE REGION IS THAT OF FIG. 10 (E AND F).

	$\hat{n}_i = 0$	$\hat{n}_i = 1$
$\mathcal{A}_i^{-1}(F, F)$	$\emptyset$	$\emptyset$
$\hat{\boldsymbol{\sigma}}_i(e_1^o, F)$	$\{R\}$	$\{R\}$
$\Lambda_e^{-1}(e_1^o)$	$\emptyset$	$\{c_1\}$

for this behavior, the block logic in Fig. 5 updates the estimated number of interacting agents to $\hat{n}_i = 1$. By doing so, the predictor automaton $\mathcal{P}_i$ becomes the second of Fig. 9 which admits the maneuver set $\hat{\sigma}_i = \{F, L\}$ (see also second column in Table II) reported in Fig. 10-d. Again, the behavior of agent i can be explained, and as a result we have $b_i = \text{correct}$.

When agent i reaches the second lane ($l_1 = 2$), it switches to maneuver fast ($\sigma_i = F$). The configuration space is partitioned w.r.t observer h as in Fig. 10-e. The number n_i has then to stay within range $[0, 1]$ according to inequality 3, and for the estimated number of interacting agents we have $\hat{n}_i = 0$. The predictor automaton is the first in Fig. 9. At the same time, event c_1 is detected, meaning that the right lane of vehicle 1 is free. Under this circumstance and with $\hat{n}_i = 0$, the predicted maneuver set is $\hat{\sigma}_i = \{R\}$, whereas the observed behavior is $\sigma_i = F$ (see first column in Table III). Hence b_i becomes uncertain, and for the estimated number of interacting agents we infer $\hat{n}_i = 1$. Yet, the predicted maneuver set is $\hat{\sigma}_i = \{R\}$, as depicted in Fig. 10-f (see second column in Table III), and the behavior of agent i can not be explained.

At this stage, block logic detects that there exists no other valid value for $\hat{n}_i$, and hence the agent i's behavior is finally classified as not–cooperative by setting $b_i = \text{faulty}$. All

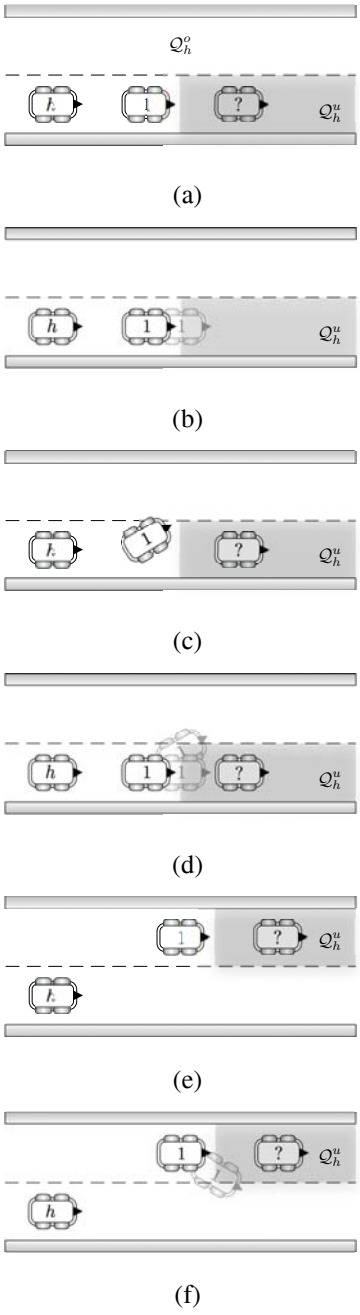

Fig. 10. Simulated system evolution along with estimated maneuvers represented by the faded agents.

signal summarizing the simulation are reported in Fig. 11.

VIII. CONCLUSION

In this paper we addressed the problem of detecting faulty behaviors in cooperative multi–agent systems. A novel decentralized and scalable architecture that can be

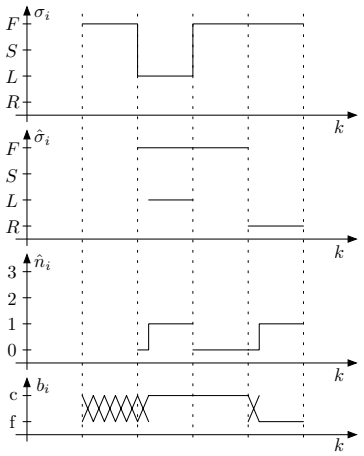

Fig. 11. Plot of measured σ_i and predicted $\hat{\sigma}_i$ maneuvers, estimated number $\hat{n}_i$ of agents interacting with i, and the cooperativeness b_i during the simulation.

adopted to realize an observer for monitoring the agents' behavior was proposed. The work aimed at defining a basic framework by which automatically realize decentralized intrusion detectors for (hybrid) multi–agent systems.

In this work we considered in particular a method by which an agent is able to establish whether its neighbors are cooperative or not, only using the onboard sensing, and hence having partial knwoledge of the system.

Future work will concern the cooperation of many local observers that can exchange information in order to improve their detection capability.

REFERENCES

[1] C. Tomlin, G. J. Pappas, and S. Sastry, "Conflict resolution for air traffic management: A case study in multi-agent hybrid systems," vol. 43, pp. 509–521, 1998.

[2] R. Ghosh and C. J. Tomlin, "Maneuver design for multiple aircraft conflict resolution," Chicago, IL, 2000.

[3] L. Pallottino, V. Scordio, and A. Bicchi, "Decentralized cooperative conflict resolution among multiple autonomous mobile agents," in *Proceedings of the Conference on Decision and Control*, vol. 5, Dec. 2004, pp. 4758–4763.

[4] L. Pallottino, V. Scordio, E. Frazzoli, and A. Bicchi, "Probabilistic verification of a decentralized policy for conflict resolution in multi-agent systems," *IEEE International Conference on Robotics and Automation*, pp. 2448–2453, 2006.

[5] J. Blum and A. Eskandarian, "The threat of intelligent collisions," *IT Professional*, vol. 6, no. 1, pp. 24–29, Jan.-Feb. 2004.

[6] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[7] T. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially observed discrete-event systems," *Automatic Control, IEEE Transactions on*, vol. 47, no. 9, pp. 1491–1495, 2002.

[8] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Failure diagnosis using discrete-event models," *Control Systems Technology, IEEE Transactions on*, vol. 4, no. 2, pp. 105–124, 1996.

[9] R. Boel and J. van Schuppen, "Decentralized failure diagnosis for discrete-event systems with costly communication between diagnosers," *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pp. 175–181, 2002.

[10] P. Ramadge and W. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[11] ——, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[12] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *Automatic Control, IEEE Transactions on*, vol. 40, no. 9, pp. 1555–1575, 1995.

[13] C. Özveren and A. Willsky, "Invertibility of Discrete-Event Dynamic Systems," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 5, no. 4, pp. 365–390, 1992.

[14] Y. Wang, T.-S. Yoo, and S. Lafortune, "Decentralized diagnosis of discrete event systems using unconditional and conditional decisions," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 12-15 Dec. 2005, pp. 6298–6304.

[15] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partialobservations," *Automatic Control, IEEE Transactions on*, vol. 33, no. 3, pp. 249–260, 1988.

[16] R. Debouk, S. Lafortune, and D. Teneketzis, "Coordinated Decentralized Protocols for Failure Diagnosis of Discrete Event Systems," *Discrete Event Dynamic Systems*, vol. 10, no. 1, pp. 33–86, 2000.

[17] G. Fourlas, K. Kyriakopoulos, and N. Krikelis, "Diagnosability of Hybrid Systems," *Proceedings of the 10th IEEE Mediterranean Conference on Control and Automation*, 2002.

[18] A. Balluchi, L. Benvenuti, M. Di Benedetto, and A. Sangiovanni-Vincentelli, "Design of observers for hybrid systems," *Hybrid Systems: Computation and Control*, vol. 2289, pp. 76–89, 2002.

[19] S. Narasimhan, F. Zhao, G. Biswas, and E. Hung, "Fault isolation in hybrid systems combining model based diagnosis and signal processing," *Proc. of IFAC 4th Symposium on Fault Detection, Supervision, and Safety for Technical Processes*, 2000.

[20] G. Fourlas, K. Kyriakopoulos, and N. Krikelis, "A Framework for Fault Detection of Hybrid Systems," *Proceedings of the 9th IEEE Mediterranean Conference on Control and Automation*, 2001.

[21] A. Balluchi, L. Benvenuti, M. Di Benedetto, and A. Sangiovanni-Vincentelli, "Observability for hybrid systems," *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 2, 2003.

[22] E. K. Yongseok Park, "On d-Inversion in Timed Discrete Event Systems with Interruption," *Decision and Control, 1995. Proceedings. 34nd IEEE Conference on*, vol. 2, 1995.