2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Multiple vehicles mission coordination using Petri nets

Narcís Palomeras, Pere Ridao, Carlos Silvestre and Andres El-fakdi

Abstract— **This paper provides a methodology to model and execute coordinated missions involving multiple vehicles using Petri nets. Individual vehicle missions are defined by means of Petri nets and three constraints are added for coordination purposes: mutual exclusion, ordering and synchronisation. The proposed methodology generates a centralised net, checks if it is deadlock free and then obtains a decentralised Petri net for every vehicle minimising the communication between them. The resulting Petri nets implement the multi-vehicle mission control program that is responsible for coordinating in real-time the set of vehicles involved in the mission.**

I. Introduction

In previous articles [1] [2] mission description for autonomous vehicles using Petri nets has been studied. Petri nets have been chosen as the formalism to describe the Discrete Event System (DES) responsible for enabling and disabling the basic vehicle primitives based on the sequence of events produced by the vehicle control architecture and by the surrounding environment. Using the Petri net formalism, it is possible to analyse the generated net to check the consistency in the resulting controller avoiding to drive the vehicle into a deadlock situation and simultaneously ensuring the reachability of all the final states related to the mission.

The purpose of this paper is to further develop this framework to be able to deal with the coordination of multiple vehicles. Our approach consists in applying a set of coordination constraints to the set of vehicles involved in the mission generating a centralised mission program, check if the deadlock free property is also satisfied in the centralised net (end states reachability is satisfied by construction) and, finally, decentralise it generating an independent mission for every vehicle minimizing the communications between vehicles. It is worth noting that reducing the amount of information to be exchanged among the vehicles is of vital importance for instance in case of Autonomous Underwater Vehicle (AUV) applications due to the very low data rate achieved nowadays by acoustic modems. In this paper three coordination constraints are studied. The first one concerns with the access to shared resources (commonly known as mutual exclusion), the second consists in the ordering of two tasks between several vehicles and the last one is related with the synchronisation between several tasks (start several tasks simultaneously).

Previous work on the use of Petri nets to the coordination of multiple vehicles can be found in [3] where the implementation of a mutual exclusion among two mine robots that have to cross a tunnel through the same point is presented. There, a supervisor is generated to solve the problem using a Supervisor Based on Place Invariants (SBPI) (check [4] for an introduction on SBPIs). In [5] a framework to describe plans using Petri nets is presented. The way in which vehicles missions (plans) are defined is similar to the one introduced in our framework [1]. It relies on a set of predefined Petri net control structures that are used to join tasks. One of this control structures is responsible for a synchronisation between two vehicles. When applying this control structure, a centralised Petri net capable of synchronising the different vehicles is obtained. This net is then divided to be executed in each vehicle involved in the mission. In both cases, coordination constraints are supervised by a centralised system, however, there are some lines of research dealing with the supervision of a decentralised systems. Theoretical work on this topic appeared during the nineties. Lin and Wonham [6] studied the supervision of a decentralised DES able to recognise a language using automates. Techniques that can be used to address the supervision of a decentralised system problem using Petri nets were introduced in [7].

The main contribution of this paper is the use of the particular structure of Petri nets to model and execute the coordination of multiple vehicles missions by implementing general constraints like mutual exclusions, ordering or synchronisation as well as to address the decentralisation of the resulting discrete event system minimising the communications among the different vehicles and preserving the coordinated system free of deadlocks.

The paper is organised as follows. Section II proposes a new framework to address the multiple vehicles coordination problem. In Section III a procedure for deadlock avoidance is introduced. An overview of the decentralised supervision methodology introduced in [7] is presented in Section IV for the readers convenience. Section V presents a full example in which several missions are coordinated and distributed and finally, how the methodology have been tested is commented in Section VI before conclusions.

II. A Proposal for Multiple Vehicles Coordination

In this paper we propose a methodology for the coordinated mission control of a team of underwater vehicles based on the theoretical developments presented in [7] for decentralised supervision of a DES. The proposed method consists on the following steps i) Program the mission of each team-vehicle using an independent Petri net [1]. ii) Use mutual exclusions, task ordering and task synchronisation

This research was sponsored by FREEsubNET (MRTN-CT-2006-036186) and the Spanish government commission MCyT (DPI2008-06548-C03-03).University of Girona, Edifici Politecnica IV, Campus Montilivi 17071 Girona, Spain npalomer@eia.udg.edu.Institute for Systems and Robotics. Instituto Superior Tecnico Lisbon, Portugal.

constrains to define a set of SBPIs involving the whole team. iii) Synthesise the SBPIs and generate a centralised mission. iv) Check that the centralised mission is deadlock free. v) Partitioning the centralised Petri net into as many Petri nets as vehicles are involved minimising the communication among them.

As introduced in our approach [1], a mission is defined as a Petri net $N = (P, T, F)$ where P is a set of places $p_i,\ i \in \{1..n\}$, a set T of transitions $t_j,\ j \in \{1..m\}$ and a set of transition arcs $F \subseteq (P \times T) \cup (T \times P)$. A mission has a set of *starting* places called B, where $B \subset P$, and a set of *ending* places called E, where $E \subset P$, and $B \cap E = \emptyset$. In the same way, $W = \cup N_i = \{\cup P_i, \cup T_i, \cup F_i\}$, $i \in \{1...c\}$ where $\forall_{i,j \in \{1..c\}} i \neq j\ P_i \cap P_j = T_i \cap T_j = F_i \cap F_j = \emptyset$ is the union of c independent Petri nets with nc places and mc transitions.

A Petri net can be reduced to a single place p_{sub_net} iff $\bullet p_{sub\_net} = \bullet B$ and $p_{sub\_net}\bullet = E\bullet$[1]. In a mission Petri net, each place represents a task, a sub-net or a waiting place. When a place representing a task is marked (has a token), it means that the task is under execution. Moreover, when a place representing a sub-net is marked, it means that at least one of the tasks in the sub-net is under execution. All tasks or sub-net[2] places are 1-bounded.

In the following subsections, the constraints used for defining coordinated missions are detailed.

A. Mutual exclusion

Figure 1(a) shows two simple Petri nets representing two independent missions (one for each vehicle) in which three tasks are sequenced. Tasks represented by the marked places p_1 and p_4 are currently under execution.

Definition A mutual exclusion is defined as a pair (M, β), where $M \subset \cup P_i$ is the set of places involved in the mutual exclusion and $\beta \in \mathbb{N} \setminus \{0\}$ is the maximum number of places belonging to M which can be simultaneously marked. In our case N_i represents the Petri net related to the mission of vehicle i. M represents the set of tasks of possible different vehicles which must accomplish the mutual exclusion property. Finally β, commonly equal to 1, represents the maximum number of tasks in M that can be simultaneously under execution.

For instance, if the tasks represented by places p_2 and p_5 in Fig.1(a) are in mutual exclusion ($M = \{p_2, p_5\}$) and only one of the tasks can be running ($\beta = 1$), then the SBPI described in *Proposition 2.1* can be used to generate the centralised Petri net supervisor shown in Fig. 1(b) used to enforce this coordination constraint.

Proposition 2.1: Let W be a set of independent Petri nets as discussed above and (M, β) a mutual exclusion defined over W, then if the constraint in (1)[3] holds, the mutual exclusion property defined above is satisfied.

$$\mathbf{l} = [l_1 \cdots l_i \cdots l_{nc}],\ with\ l_i = \begin{cases} 1 & \text{if } p_i \in M \\ 0 & \text{otherwise} \end{cases} \quad \mathbf{l} \cdot \mu \leq \beta\ where \tag{1}$$

Proof: Because only those components $l_i \in \mathbf{l}$ related to places $p_i \in M$ are set to 1 and the rest of them are set to 0 and the task places are 1-bounded, $\mathbf{l} \cdot \mu$ gives the number of places in M simultaneously marked. Hence, using the fact that a marked place represents a running task, if the constraint $\mathbf{l} \cdot \mu \leq \beta$ holds, it follows that no more than β tasks of M can be simultaneously under execution. ■

It is worth noting that a waiting place has been added before every $p_i \in M$ in Fig. 1(b). A simple transformation called PW-Transformation has been applied for this purpose.

Definition The PW-Transformation over a place p_i in the Petri net $N = \{P, T, F\}$ is defined as $P' = P \cup \{p_i^{wait}\}$, $T' = T \cup \{t_i^{wait}\}$ where $\bullet p_i^{wait} = \bullet p_i$, $p_i^{wait}\bullet = \{t_i^{wait}\}$, $t_i^{wait}\bullet = \{p_i\}$ and $\bullet p_i = \{t_i^{wait}\}$.

The PW-Transformation does not modify the Petri net behaviour, however, the additional waiting place allows a previous task to finalise even if the next task in the sequence can not be executed due to a mutual exclusion. Applying the PW-Transformation to p_2 and p_5 and the *Proposition 2.1* to the Petri net in Fig.1(a) to impose the mutual exclusion ($M = \{p_2, p_5\}, \beta = 1$), the following constraint is obtained.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mu(p_1) \\ \mu(p_2) \\ \mu(p_3) \\ \mu(p_4) \\ \mu(p_5) \\ \mu(p_6) \end{bmatrix} \leq 1 \ = \ \mu(p_2) + \mu(p_5) \leq 1 \tag{2}$$

To synthesise the SBPI derived from *Proposition 2.1* it is necessary to obtain the incidence matrix of the centralised Petri net (D_p). This incidence matrix is a block diagonal matrix composed by joining the incidence matrix of each team-vehicle Petri nets (D_{pi}).

$$D_p = \begin{bmatrix} D_{p1} & 0 & \cdots & 0 \\ 0 & D_{pi} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_{pk} \end{bmatrix} \tag{3}$$

In order to apply the restriction $\mathbf{l} \cdot \mu \leq b$ to the Petri net represented by D_p equations (4) and (5) are used as reported in [4]:

$$D_c = -\mathbf{l} \cdot D_p \tag{4}$$

$$\mu_{c0} = b - \mathbf{l} \cdot \mu_{p0} \tag{5}$$

where D_c describes the Petri net controller and μ_{c0} is its initial marking. Then, the incidence matrix of the resulting centralised Petri net (D) and its initial state (μ_0) can be obtained as:

$$\mathrm{D} = \begin{bmatrix} D_p \\ D_c \end{bmatrix} \qquad \mu_0 = \begin{bmatrix} \mu_{p0} \\ \mu_{c0} \end{bmatrix} \tag{6}$$

[1]Where $\bullet p$ is the set of transitions with output arcs to place p, $p\bullet$ is the set of transitions receiving input arcs from place p, $\bullet t$ is the set of places with output arcs to transition t and $t\bullet$ is the set of places receiving input arcs from transition t.

[2]From this moment both tasks and sub-nets will be referenced as tasks.

[3]Where $\mathbf{l}$ is a vector of integers with as many elements as places in the centralised Petri net and μ is the marking vector of the centralised Petri net.

B. Ordering

The notion of order appears quite naturally when describing distributed systems. Ordering between two tasks happens when one task can be only executed after the termination of another one. In this framework, launching (enabling) a task represented by the 1-bounded place p with $\mu(p) = 0$ consists in marking p with one token ($\mu(p) = 1$). On the other hand, terminating (disabling) a task p currently under execution ($\mu(p) = 1$) consist in removing the token from p ($\mu(p) = 0$.)

Definition An ordering pair is defined as a set of two places $O = \{\mathrm{p}_s,\, p_w\} \subset \cup P_i$ where p_w can not be marked before the unmarking of p_s.

Again, a PW-Transformation must be applied before adding the ordering constraint. This transformation must be applied to the place $p_w \in O$.

To ensure the ordering constraint, an extended form of the $\mathbf{l} \cdot \mu \leq b$ constraint is used. This form, $\mathbf{c} \cdot \upsilon \leq b$, described in [8] uses the Parikh vector (υ) to control the Petri net behaviour. The Parikh vector contains a counter for every transition in the system. Every counter is initialised to 0 and when a transition t_j fires, the corresponding element υ_j of the Parikh vector is incremented. For ordering the two places contained in an ordering pair, transitions $p_s\bullet$ and $\bullet p_w$ are used.

Proposition 2.2: Let W be a set of independent Petri nets and $O = \{p_s, p_w\}$ an ordering pair defined over W, then it can be shown that if the constraint (7) holds, the properties of the ordering pair are satisfied.

$$\mathbf{c} \cdot \upsilon \leq 0 \; where$$
$$\mathbf{c} = [c_1 \cdots c_j \cdots c_{mc}],\; with\; c_j = \begin{cases} -1 & \text{if } t_j \in p_s\bullet \\ 1 & \text{if } t_j \in \bullet p_w \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Proof: With the Parikh vector it is possible to register the number of firings of each transition. To ensure the order between both tasks, the number of firings of the transition immediately after the place p_s must be always equal or greater than the number of firings of the transition before the place p_w. If the transition before place p_w fires before the transition after the place p_s, the constrain synthesised in *Proposition 2.2* will be bigger than 0 making itself false. ■

Figure 1(c) shows an example of an ordering constraint $O = \{p_5, p_2\}$ applied over the nets presented in Fig. 1(a) where the PW-Transformation has been applied to p_2. Following *Proposition 2.2* the constraint (8) has been defined and used to generate the SBPI.

$$\begin{bmatrix} 0 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} \upsilon(t_1) \\ \upsilon(t_2) \\ \upsilon(t_3) \\ \upsilon(t_4) \end{bmatrix} \leq 0 \;=\; \upsilon(t_2) - \upsilon(t_4) \leq 0 \tag{8}$$

C. Synchronisation

Synchronisation constraints are used to fix rendevouz among tasks allowing them to be launched simultaneously.

Definition A set of places $S \subset \cup P_i$ is said to be synchronised if and only if $\forall_{p_i, p_j \in S}$ / $p_i \neq p_j$ and $\sharp\bullet p_i = \sharp\bullet p_j = 1$, $\bullet p_i$ is enabled if and only if $\bullet p_j$ is enabled.

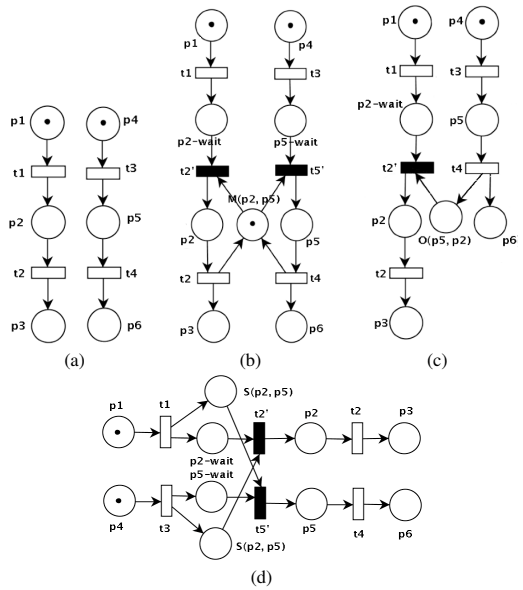

Fig. 1. (a) Example of two simple Petri net missions. (b) Mutual exclusion between tasks p_2 and p_5. (c) Ordering: p_2 has to be executed after the termination of p_5. (d) Synchronisation of tasks p_2 and p_5.

To synthesise a set of SBPIs to enforce this constraint a new set of places $S' = \{p_i^{wait}\}$ has to be defined where all the p_i^{wait} comes from the PW-Transformation of p_i $\forall_{p_i \in S}$.

Proposition 2.3: Let W be a set of independent Petri nets and S a synchronisation set defined over W , then it can be shown that if the constraint (9) holds, the synchronisation property is satisfied.

$$\forall_{p_k^{wait}, p_w^{wait} \in S'} \; where\; p_k^{wait} \neq p_w^{wait}$$
$$obtain\; a\; \mathbf{c} \cdot \upsilon \leq 0 \; where$$
$$\mathbf{c} = [c_1 \cdots c_j \cdots c_{mc}],\; with\; c_j = \begin{cases} -1 & \text{if } t_j \in \bullet p_k^{wait} \\ 1 & \text{if } t_j \in p_w^{wait}\bullet \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Proof: A synchronisation between places $S = \{p_i, p_j\}$ is equivalent to the orderings $O_1 = \{\bullet \bullet p_j, p_i\}$ and $O_2 = \{\bullet\bullet p_i, p_j\}$. Therefore the proof follows the lines of the proof of *Proposition 2.2*. ■

If the tasks represented by the places p_2 and p_5 in Fig.1(a) have to be executed synchronously, a synchronisation set $S = \{p_2, p_5\}$ must be defined. After apply the PW-Transformation to p_2 and p_5 and building the set $S' = \{p_2^{wait}, p_5^{wait}\}$ the constraints $-\upsilon(t_1) + \upsilon(t_5') \leq 0$ and $\upsilon(t_2') - \upsilon(t_3) \leq 0$ are synthesised following *Proposition 2.3* to get the supervised system presented in Fig. 1(d).

III. Deadlock Avoidance

After apply a set of coordination constraints among several single vehicle missions it is necessary to check that the resulting centralised mission is deadlock free.

Figure 2 shows two basic cases in which deadlocks can appear by combining the above mentioned constraints. The deadlock in Fig.2(a) arise because $O(p_2, p_5)$ has to be marked before firing t_5' to mark then p_5, fire t_4 and finally mark $O(p_5, p_2)$. Hence, $O(p_2, p_5)$ is a predecessor

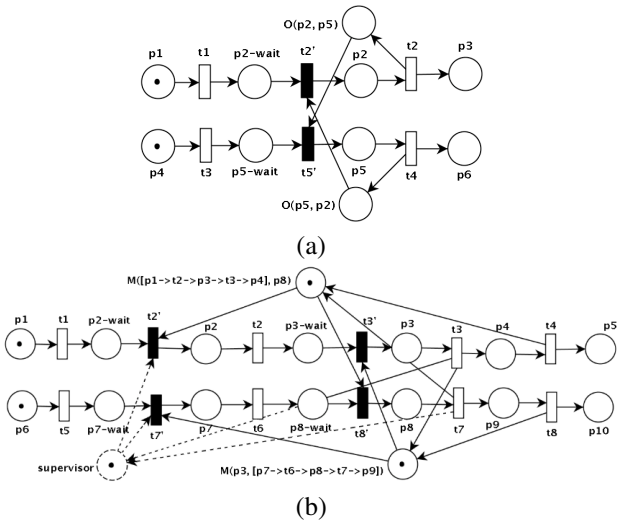

Fig. 2. (a) A deadlock appears when two ordering constraint $O(p_2, p_5)$ and $O(p_5, p_2)$ are combined. (b) A deadlock appears when the two mutual exclusions $M([p_2 \rightarrow t_2 \rightarrow p_3 \rightarrow t_3 \rightarrow p_4]_{sn}, p_8)$ and $M(p_3, [p_7 \rightarrow t_6 \rightarrow p_8 \rightarrow t_7 \rightarrow p_9]_{sn})$ are combined.

of $O(p_5, p_2)$. But at the same time $O(p_2, p_5)$ can only be marked if t'_2 fires marking p_2 and also firing t_2. Since $O(p_5, p_2)$ must be marked in order to enable t'_2, $O(p_5, p_2)$ must also be a predecessor of $O(p_2, p_5)$ and hence the deadlock appears. This deadlock can be detected but not avoided. However, the deadlock shown in Fig.2(b) can appear if t'_7 fires after t'_2 and before t_3 or if t'_2 fires after t'_7 but before t_7. This deadlock can not be only detected but avoided too applying an extra supervisor. A procedure to check if a Petri net is deadlock free as well as able to add the needed supervisors to avoid them is described next.

A. Deadlock Avoidance Procedure

Several techniques to avoid deadlocks in Petri nets have been proposed in the literature [9]. Most of them are based on the well known necessary condition for deadlock, namely that a deadlocked ordinary Petri net contains at least one empty siphon. Hence, to avoid a deadlock, it is necessary to avoid that the siphons in the net become empty. To achieve this, two conditions have to be accomplished: (i) all the siphons must be initially marked, and (ii) all the siphons must be controlled[4]. A siphon can be controlled by a trap or by a place invariant. If a siphon is not controlled by the own Petri net it is possible to add a constraint like $\mathbf{l} \cdot \mu \leq 1$ to control it. Algorithm 1 describes a simple deadlock avoidance procedure applicable to our framework.

Applying Algorithm 1 to the Petri net in Fig. 2(a) the invariants obtained are: $p_1 + p_2^{wait} + p_2 + p_3 = 1$, $p_4 + p_5^{wait} + p_5 + p_6 = 1$ and $p_2 + p_5 + O(p_2, p_5) + O(p_5, p_2) = 0$. To apply the Petri net modification described in step ii, a transition (t') must be added between the end place p_3 and the begin place p_1 as well as between p_6 and p_4. Five minimal siphons are obtained in the transformed Petri net. All of them are

[4]A siphon is said to be controlled if it will never lost all its tokens.

Algorithm 1 Deadlock avoidance procedure

i) Calculate the invariants of the centralised net.
ii) For every vehicle mission, join the final places with the initial place through a single transition.
iii) Calculate the minimal siphons and traps in the transformed Petri net resulting from step (ii).
iv) For every siphon Sp that is not controlled by a place invariant or a trap or that is not initially marked, generate a constraint $\mathbf{l} \cdot \mu \leq 1$ using equation (10). If there are no uncontrolled or initially unmarked siphons, the Petri net is deadlock free and the algorithm finalises.
v) If the constraint $\mathbf{l} \cdot \mu \leq 1$ generated in step (iv) produces a $D_c = [0, 0, ...0]$ applying equation (4) means that it is impossible to add a supervisor to avoid the deadlock. The algorithm finalises with a deadlocked Petri net.
vi) If the D_c generated in step (iv) is valid ($d_c \neq [0, 0, ...0]$), add the supervisor to the original Petri net, add the place invariant to the list created in step (i) and repeat from step (ii).

$$\mathbf{l} = [l_1 \cdots l_i \cdots l_{nc}],\ where\ l_i = \begin{cases} 1 & \text{if } p_i \in Sp \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

trap or invariant controlled but $\{p_2, p_5, O(p_2, p_5), O(p_5, p_2)\}$ is not initially marked. When we try to synthesise a supervisor the resulting supervisor it happens to be not valid ($D_c = [0, 0, ..., 0]$) and the algorithm terminates at step (v). However, applying the same algorithm to the Petri net of Fig. 2(b) 4 invariants and 5 siphons are obtained. The only uncontrolled siphon is $\{p_4, p_3, p_8, p_9, M([p_2 \rightarrow t_2 \rightarrow p_3 \rightarrow t_3 \rightarrow p_4], p_8), M(p_3, [p_7 \rightarrow t_6 \rightarrow p_8 \rightarrow t_7 \rightarrow p_9])\}$. A supervisor is generated in step (iv) and applied to the Petri net as shows the $supervisor$ place in Fig.2(b). As no more siphons appear, the algorithm terminates with a deadlock free Petri net.

IV. Decentralised Supervision

In [7], Iordache and Antsaklis presented a set of algorithms to check if a centralised system can be distributed among several subsystems, how to add minimal communication in case that the system is not directly distributable and how to implement the distributed subsystems if it is distributable. This section reproduces some of the algorithms described in [7] and that will be used to decentralise the multiple vehicle mission control.

A. Checking the d-admissibility of a constraint

A system is admissible if all its supervisors control only controllable transitions and detect only observable transitions. To check if a system that is admissible in a centralised way (c-admissible) is also admissible once distributed (d-admissible) all its constraints have to accomplish Algorithm 2. T_o^M and T_c^M are the observed and controlled transitions by the constraint $\mathbf{l} \cdot \mu + \mathbf{h} \cdot q + \mathbf{c} \cdot \upsilon \leq b$ to be checked. ζ is the set of subsystems in which the centralised system will be split and $T_{o,i}$ and $T_{c,i}$ are the observable and controllable transitions for each subsystem. In our framework, the subsystems ζ as well as the $T_{o,i}$ and $T_{c,i}$ sets are defined for the initial uncoordinated vehicle missions. There are as many sub-missions as vehicles and

it is assumed that each vehicle is only capable of observing and controlling their own transitions.

Algorithm 2 Check the d-admissibility of a constraint

1) Find T_o^M and T_c^M.
2) Let ζ be the set of indices i satisfying $T_{o,i} \supseteq T_o^M$.
3) If $\zeta = \emptyset$, declare the constraint not d-admissible and exit.
4) Define $T_c = \cup_{i \in \zeta} T_{c,i}$.
5) If T_c satisfy $T_c \supseteq T_c^M$ then constraint d-admissible else constraint not d-admissible.

In general, it can be difficult to compute T_o^M and T_c^M. Alternatively, estimates of $T_c^e \supseteq T_c^M$ and $T_o^e \supseteq T_o^M$ can be used. However, in this case the algorithm only checks a sufficient condition for d-admissibility. T_c^e and T_o^e can be calculated using the control place C generated for the SBPI synthesised by the constraint to check in the centralised system as $T_c^e = C\bullet$ and $T_o^e = \bullet C\bullet$

B. Design minimising communication

If the Algorithm 2 returns that a constraint is not d-admissible it is possible to introduce communication in order to make the constraint d-admissible. To characterise communication three binary variables are introduced: $\alpha_{i,j} = 1$ iff the transition t_j is communicated to subsystem s_i, $\epsilon_{i,j} = 1$ iff the transition t_j is remotely controlled by subsystem s_i and $\delta_{i,k} = 1$ iff the subsystem s_i intervenes with the constraint k.

Algorithm 3 applies an Integer Linear Program (ILP) to solve (14) minimising the communication cost of the distribution.

Algorithm 3 Design minimising communication

1) Solve (14) subjected to (11), (12) and (13).

$$\alpha_{i,j} \geq \delta_{i,j} \ \forall j \in \{f : t_j \in T_o^k \setminus T_{o,i}\} \tag{11}$$

$$\forall k = 1...n_c : \ \sum_{i=1}^{n} \delta_{i,k} \geq 1 \tag{12}$$

$$\forall k = 1...n_c, \ \forall x = 1...n, \ \forall j \in \{y : t_y \in T_c^k\} :$$
$$\delta_{x,k} \leq \epsilon_{x,j} + \sum_{i \in I_j} \delta_{i,k} \tag{13}$$

$$min \sum_{i,j} \alpha_{i,j} c_{i,j} + \sum_{i,j} \epsilon_{i,j} f_{i,j} + \sum_{i,k} \delta_{i,k} h_{i,k} \tag{14}$$

Where n_c is the number of constraints, n the number of subsystems, $c_{i,j}$ is the cost of communicate the t_j firing to subsystem s_i, $f_{i,j}$ is the cost of control the t_j firing from subsystem s_i and $h_{i,k}$ is the cost that subsystem s_i intervenes in constraint k.

C. Supervisor Design for a d-admissible constraint

If the Algorithm 2 shows that all the constraints are d-admissible or using Algorithm 3 a communication policy is found to made all the constraints d-admissible, it is possible to design a decentralised supervisor applying Algorithm 4 to every constraint.

Algorithm 4 Supervisor Design for a d-admissible constraint

1) Let μ_0 be the initial marking of N, C the control place of the centralised SBPI enforcing $\mathbf{l}\mu + \mathbf{h}q + \mathbf{c}v \leq b$ and ζ the set of subsystems.
2) $\forall_{i \in \zeta}$, let $x_i \in \mathbb{N}$ be a state variable of s_i.
3) Define S_i, for $i \in \zeta$, by the following rules:
3.1 - Initialise $x_i = \mu_{s0}$
3.2 - If $t \in T_{c,i}$, $t \in C\bullet$ and $x_i < W_s(C,t)$, then S_i disables t.
3.3 - If t fires, $t \in T_{o,i}$ and $t \in \bullet C$, then $x_i = x_i + W_s(t,C)$.
3.4 - If t fires, $t \in T_{o,i}$ and $t \in C\bullet$, then $x_i = x_i - W_s(C,t)$.

V. Multiple vehicles coordination example

A typical underwater vehicles mission involving several coordinated vehicles is a Mine Countermeasures Mission (MCM). In the proposed example two vehicles are used to survey a zone and inspect all the Mine Like Objects (MLO) found. The first vehicle has a torpedo shape and it is in charge to survey the zone and inform the other about the position of all the MLO found. The second one is an open frame shaped vehicle which goal is to inspect the already detected MLO. Both vehicles share the same acoustic localisation system installed on the main surface vessel. To avoid acoustic interference's only one vehicle can access the acoustic localisation device at the same time. Hence, both vehicles must navigate using a dead reckoning system and take a position fix only when their navigation uncertainty is above a threshold.

To define a Petri net mission for every vehicle the Mission Control Language (MCL) introduced in [2] is used. This high level language is able to automatically compile a mission program into the corresponding Petri net. Fig. 3 show how these independent missions are written using MCL.

To describe the necessary constraints to coordinate the vehicles missions, three new keywords have been added to the MCL in order to define mutual exclusions, orderings and synchronisation constraints. In the presented mission three constraints have been proposed: (i) A synchronisation between both vehicles in order to go to the initial position simultaneously, (ii) an ordering constraint to allow only the inspection vehicle check for a MLO location when the survey vehicle has found a new MLO and, (iii) a mutual exclusion to

```
mission: surveyVehicle {
 CheckSystems();
 Goto(initial_position): startMission;
 parallel {
    parallel {
       Trajectory(way_points_list)
    } or {
       while( True() ) {
          SearchMineLikeObject();
          SendInfo(MLO_position)
       }
    };
    SendInfo(home_position);
    Goto(home)
 } or {
    while( True() ) {
       if ( CheckDrift() )
          { WaitPositionFix() }
       else { Wait(30) }
    }
 }
}
```

(a)

```
mission: inspectionVehicle {
 CheckSystems();
 Goto(initial_position): startMission;
 parallel {
    while( True() ) {
       CheckInfo();
       Goto(received_location);
       InspectMineLikeObject();
    }
 } or {
    while( True() ) {
       if ( CheckDrift() )
          { WaitPositionFix() }
       else { Wait(30) }
    }
 }
}
```

(b)

Fig. 3. (a) Survey Vehicle Mission. (b) Inspection Vehicle Mission.

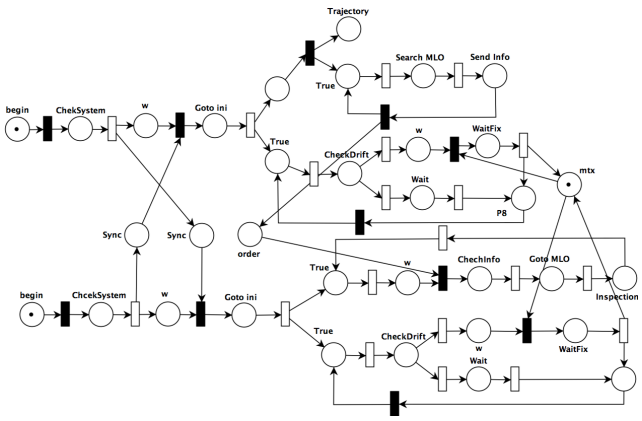

Fig. 4. Centralised Petri net mission with coordination constraints.

prevent both vehicles to use its acoustic localisation system simultaneously. These constraints are specified using new MCL keywords as follows:

- **sync**: {surveyVehicle: startMission, inspectionVehicle: startMission}
- **order**: {surveyVehicle: SendInfo, inspectionVehicle: CheckInfo}
- **mutex**: {surveyVehicle: WaitPositionFix, inspectionVehicle: WaitPositionFix} = 1

Once the imposed constraints are applied the centralised Petri net showed in Fig. 4 is automatically obtained. To enhance the paper readability the Petri net has been simplified removing the fail and abort states and its corresponding subnets. The *w* places were generated by the PW-Transformation and four supervisors places are also added according to *Proposition 2.1*, *2.2* and *2.3*. The synchronisation between the tasks *Goto(initial_position)* in both vehicles is supervised by the two *sync* places. The ordering between *SendInfo* and *CheckInfo* is supervised by the *order* place that will receive as many tokens as MLO are found[5]. Finally, the mutual exclusion between the vehicles to take a location fix is supervised by the *mtx* place initialised with one token.

After checking that this centralised Petri net is deadlock free (Algorithm 1), the minimal communication policy between vehicles must be found. If the cost to remotely control a transition uncontrollable without communication is supposed to be 100 times larger than the cost to remotely observe a transition unobservable without communication, the transitions that must be communicated from one vehicle to another are found applying the ILP method described in Algorithm 3. With the list of transitions to communicate and following Algorithm 4 an independent Petri net for each vehicle is generated again. These Petri nets can be executed in real time on board each vehicle ensuring no deadlock and minimal communication between vehicles.

VI. Tests

In order to test the whole system, the synthesis of SBPI for the coordination constraints as well as the PW-Transformation have been implemented together with the ILP presented in Algorithm 3 and the supervisor design for d-admissible constraints detailed in Algorithm 4. All these procedures have been added to the existing version of Mission Control Language - Compiler (MCL-C) [2] to allow for the multiple vehicles mission specification. The MCL-C generates an individual mission for every vehicle that can be executed in real time through a Petri net player against any autonomous vehicle implementing an Architecture Abstraction Layer (AAL) [1]. Several multiple vehicle missions have been tested using the Webots[6] simulator and the e-puck robots. After programming a control architecture with some available primitives and an AAL module to enable the communication between the e-pucks control architecture and the Petri net player, different missions have been written in MCL and tested in simulation. The MCL-C, the software necessary to build and distribute the Petri nets and the code to run the e-pucks under the Webots simulator can be found in *http://eia.udg.edu/∼npalomer*.

VII. Conclusions

The aim of this paper was to provide a methodology to model and execute multiple vehicles coordinated missions using Petri nets. Individual vehicle missions were described by means of Petri nets and using three coordination constraints (mutual exclusions, ordering and synchronisations) the proposed methodology generates a centralised net, checks if it is deadlock free, calculates the minimal communication between vehicles, and obtain again an individual Petri net for every vehicle. The proposed methodology presents a reliable and completely automatic way to add coordination constraints between several vehicle missions modelled and executed using the Petri net formalism. An example was detailed in Section V to illustrate how the methodology can be applied. Moreover, some tests have been make in simulation using the Webots software and the e-puck robots. Experiments involving real vehicles are expected in a near future.

References

[1] N. Palomeras, P. Ridao, M. Carreras, and C. Silvestre, "Towards a mission control language for auvs," *17th IFAC World Congress*, 2008.

[2] ——, "Using petri nets to specify and execute missions for autonomous underwater vehicles," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4439–4444, Oct 2009.

[3] J. King, R. Pretty, and R. Gosine, "Coordinated execution of tasks in a multiagent environment," *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, vol. 33, no. 5, pp. 615–619, 2003.

[4] J. Moody and P. Antsaklis, "Petri net supervisors for discrete event systems," *Thesis*, p. 282, Sep 1998.

[5] V. Ziparo and L. Iocchi, "Petri net plans," *Proc. of ATPN/ACSD Fourth International Workshop on Modelling of Objects, Components, and Agents*, 2006.

[6] F. Lin and W. Wonham, "Decentralized control and coordination of discrete-event systems with partial observation," *Automatic Control, IEEE Transactions on*, vol. 35, no. 12, pp. 1330 – 1337, Dec 1990.

[7] M. Iordache and P. Antsaklis, "Decentralized supervision of petri nets," *Automatic Control, IEEE Transactions on*, vol. 51, no. 2, pp. 376 – 381, Feb 2006.

[8] ——, "Synthesis of supervisors enforcing general linear vector constraints in petri nets," *American Control Conference, 2002. Proceedings of the 2002*, vol. 1, pp. 154 – 159 vol.1, Apr 2002.

[9] M. V. Iordache, J. O. Moody, and P. J. Antsaklis, "Automated synthesis of deadlock prevention supervisors using petri nets," *Technical Report of the ISIS Group at the University of Notre Dame*, p. 56, Jul 2002.

[5] See that this is not a task place, hence, it is not 1-bounded.

[6] http://www.cyberbotics.com/