A Recursive Model for Distributed Planning

Amal El Fallah Seghrouchni
LIPN - CNRS URA 1507
Université Paris-Nord, France
elfallah@ura1507.univ-paris13.fr

Serge Haddad
LAMSADE - CNRS URA 825
Université Paris-Dauphine, France
haddad@lamsade.dauphine.fr

Abstract

Distributed planning is fundamental to the generation of cooperative activities in Multi-Agent Systems. It requires both an adequate plan representation and efficient interacting methods allowing agents to coordinate their plans. This paper proposes a recursive model for the representation and the handling of plans by means of Recursive Petri Nets (RPN) which support the specification of concurrent activities, reasoning about simultaneous actions and continuous processes, a theory of verification and mechanisms of transformation (e.g. abstraction, refinement, merging). The main features of the RPN formalism are domain independence, broad coverage of interacting situations and operational coordination. This paper also provides an approach to the interleaving of execution and planning which is based on the RPN semantics and gives some significant methods allowing plan management in distributed planning. It goes on to show how this approach can be used to coordinate agents' plans in a shared and dynamic environment.

Distributed Planning Requirements

Multi-agent planning is fundamental to the generation of cooperative activities. The planning process is distributed when it implies autonomous agents who generate and execute their own plans. Two cases can be distinguished: task-driven planning which lies within the framework of distributed problem solving (Durfee & Lesser 1987) (Decker 1992) and agent-driven planning which concerns the coordination problem (Martial 1990). This latter (i.e. distributed planning) is characterized by two strongly correlated aspects: there is no global plan (i.e. several goals have to be reached simultaneously) and the planning process must be dynamic (i.e. interleaving of planning and execution)(Ephrati & Rosenschein 1995) .
One of the major problems in distributed planning is to define an efficient model of actions and plans which can easily and naturally take into account the potential concurrence between the agents' activities or between agents and any dynamic processes resulting from the environment. Many up till now well-established concepts in single-agent planning (e.g. the world state, considering an action as a relationship between two states, the action history, atomic events, etc.) (Georgeff 1990) require a new meaning. The execution of concurrent actions in a shared and dynamic environment introduces new problems such as ordering actions which are linked to the notion of conflictual resources (i.e. consumable or not), non-instantaneous actions, non-atomic events, etc.
Different research has been done in distributed planning, in particular incremental planning. The approach suggested in (Martial 1990) offers a number of advantages in that the model is a theoretical one which handles both positive and negative interactions. It is however limited by the fact that only two agents (point to point) can be coordinated at a time, whereas it would be more useful to be able to coordinate n agents, thus providing a really distributed coordination. In addition, this approach doesn't provide for the interleaving of planning and execution. The approach put forward by (Alami et al. 1994) is based on the paradigm of plan merging. Although it is robust and handles n agents at a time, it is centralized by the new agent. Moreover, the existing agents are suspended during the process and the new plan is generated in function of existing ones. This is incompatible with the idea of an autonomous agent who can produce his own plan to be coordinated later. Other research focuses on distributed planning but is based rather on organizational structures (Werner 1990) (Osawa 1992). The approach we have described in (El Fallah & Haddad 1996) proposes a distributed algorithm which coordinates n agents' plans at once but the formalism used may be improved by introducing the notions of abstraction and dynamic refinement in the plan.
Here, we are concerned with the paradigm of agent-driven planning where it is assumed that no global plan or global view are necessary, and an agent plan can change for several reasons (e.g. environment changes, agents' requests, etc.). This paper provides a recursive model for plan management and an approach to the interleaving of execution and planning for dynamic tasks performed by a group of agents. It tries to answer the question of how to get a group of self-interested agents

to carry out coordinated activities in a dynamic environment.
In the following, the case study based on a transportation domain scenario is briefly introduced. An efficient framework for plan management is then developed. The ordinary Petri Net formalism is extended to a Recursive Petri Net (RPN) in order to fulfill the distributed planning requirements. An overview of the main advantages expected from introducing RPN formalism is given and the RPN formalism itself is described. The plan management through RPN is illustrated through some modeling aspects which allow the handling of both positive and negative interactions.

Transportation Domain Scenario

This scenario is inspired by the case study described in (Martial 1990). It will be used to illustrate each stage of our approach. The system is made up of two types of agents: Conveyors and Clients.

- **The Conveyor:** A Conveyor has a maritime traffic net modeled as a graph where the nodes describe the ports on the net and the arcs represent the channels between them. He also has a boat with a given volume. From each port, the directed neighbors are known using a routing table. Each Conveyor has 2 hangars per port: In_Hangar used for loading incoming goods, and Out_Hangar used for unloading outgoing goods. Each hangar can stock only one container at a time. The Conveyor's motivation (goal) is to transport goods through the net using his boat between a source port and a destination port.

- **The Client:** A Client produces goods near the departure port and consumes them at the arrival port. The Client puts the goods he produced in the Out_Hangar of the departure port. He has to find a Conveyor who will transport the goods to the arrival port. Then the Client gets these goods at the In_Hangar of the arrival port in order to consume them. The Client's motivation (goal) is to get his goods transported between the production and consumer ports.

The problem now is to answer the two questions:

- When must agents' coordination happen? When positive interactions are detected, coordination is desirable and even necessary and may be considered as an optimization of the plans' execution (e.g. Cooperation between Conveyors and Clients and optimization of the boat filling). On the other hand, when negative interactions are detected, coordination becomes indispensable to the plans' execution (e.g. a hangar is a critical resource since it can stock one good at once (Boolean value) and the boat is limited by the quantity of goods it can contain (Real value)).

- How may agents' coordination be ensured? To answer this question the coordination mechanism will be illustrated in the following.

RPN Formalism for Distributed Planning

Motivation

The synergy between agents helps the emergence of coherent plans, i.e. it cancels negative effects and favors cooperation between agents. Consequently, it requires sharing information and synchronization between the parallel activities of the agents. The Petri Nets are suitable for modeling, analyzing and prototyping dynamic systems with parallel activities, so distributed planning lends itself very well to this approach. The main contribution we expect from Petri Nets is their ability to improve the representation of complex plans and to allow their dynamic coordination and execution. Applied to distributed planning, the Petri Net model mainly offers the following advantages:

- natural and graphical expression of the synchronization of parallel activities that are the performance of the agents' tasks,

- clear decomposition of processing (transitions) and sharing data (places),

- scheduling of the actions (causal and temporal relationships) of plans,

- dynamic allocation of tasks,

- qualitative and quantitative analysis of Petri Net modeling of a plan.

The Recursive Petri Net formalism we have introduced overcomes some limitations of usual categories of Petri Nets (Jensen 1991) (e.g. ordinary Petri Nets, High-Level Petri Nets (HLPN) and even Hierarchical HLPN (HHLPN)) that are apparent if one considers a Petri Net as a plan of actions:

- transition firings are instantaneous whilst an action lasts some time,

- (HLPN only) transitions are elementary actions as one needs to see an action as an abstraction of a plan,

- (HHLPN) when a transition is an abstraction, there is no clear end to its firing,

- some dynamicity is required in the structure of the net but in a controlled way.

The processing described in the next section is based on dynamic planning which is supported by RPN through the interleaving of execution and planning. In addition, the hierarchical aspect of RPN supports the dynamic refinement of transitions and allows a plan to be considered at multiple levels of abstraction.

A Recursive Model of Plans and Actions

Plans: A plan organizes a collection of actions which can be performed sequentially or concurrently in some specific order. Furthermore these actions demand both local and shared resources. A correct plan execution requires that whatever the total order that extends the partial order obtained from the plan, it must remain feasible.

Actions: A plan involves both elementary actions associated with irreducible tasks (but not instantaneous ones) and complex actions (i.e. abstract views of the task). Semantically, there are three types of actions:

- an *elementary action* represents an irreducible task which can be performed without any decomposition,
- an *abstract action*, the execution of which requires its substitution (i.e. refinement) by a new sub-plan. There are two types of abstract actions:
  - a *parallel action*, which can be performed while another action is being executed,
  - an *exclusive action*, the execution of which requires a coordination with other current executions,
- an *end action*, which is necessarily elementary, gives the final results of the performed plan. The plan goals are implicitly given through its end actions.

Methods: Intuitively, a method may be viewed as some way to perform an action. Several methods can be associated with an action. A method requires: a label, a list of formal parameters to be linked when the method is executed, a set of Pre-Conditions (i.e. the conditions that must be satisfied before the method is executed), and a set of Post-Conditions (i.e. the conditions that will be satisfied after the method has been executed). Depending on the action definition, a method may be elementary or abstract. An elementary method calls for a sub-routine in order to execute the associated action immediately but not instantaneously. An abstract method calls for a sub-plan corresponding to the chosen refinement of the associated abstract action. The refinement occurs so as to detail abstractions and display relevant information.
Example: Let us assume that the Initial Conveyor Plan (see Fig.1) is reduced to the abstract action Self.Work where Self represents the agent identity and Work is the method label. The associated method Ag.Work encapsulates two methods Ag.GoTo(Dest) where Dest represents the source location of the good (see the following table) and Ag.Transport(Good) corresponds to the abstract action in the Self.Work refinement (see Fig.1). No method is associated with the end action Self.End_Conv since it is just a synchronization action.

Method		**Ag.GoTo(Dest)**	
Type		Abstract	
Variables		Conditions	
Name	Class	Pre	Post
Ag	CONVEYOR	None	Ag.Cur_Loc = Dest
Dest	LOCATION	None	None

Syntactic Definitions of RPN

Definition 1 Method
A Method is defined through three components:

- *An identifier or label*
- *An abstract attribute which represents the type of the associated method (e.g. abstract, elementary)*
- *A set of initialized RPN generated by an initialization mechanism in the case of abstract method.*

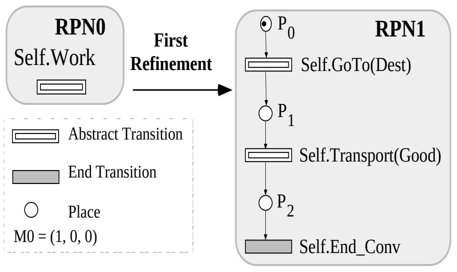

Figure 1: First Refinement of Initial Conveyor Plan

Definition 2 Recursive Petri Net
An RPN is defined as a tuple
$< P, T, Pre, Post, Var, Call >$ *where:*

- P *is the set of places*
- T *is the set of transitions such that:*
  $T = T_{elem} \uplus T_{abs} \uplus T_{end}$ *where:*
  $T_{abs} = T_{par} \uplus T_{exc}$ *and* $\uplus$ *represents the disjoint union*
- Pre *is the precondition matrix and is a mapping from* $P \times T$ *to* N
- $Post$ *is the postcondition matrix and is a mapping from* $P \times T$ *to* N
- Var *is a set of variables*
- $Call(t)$ *is a method call associated with* t *and defined through the following components:*
  - *the label of the method,*
  - *an expression (built on* Var *variables) of the agent who calls the method,*
  - *an expression of the call parameters (built on* Var *variables) which represents the Pre- and Post-Conditions associated with the method.*

Definition 3 An initialized RPN
An initialized RPN is defined as a tuple
$< R, M_0, Bind >$ *where:*

- R *is the skeleton of* RPN,
- M_0 *is the initial marking of* RPN *(mapping from* P *to* N*),*
- *Bind is the function which links all (a total link) or some (a partial link) variables of* Var *to the domain objects.*

Let us note that the objects represent the domain data and allow to instantiate the RPN variables and the parameters of the methods.
An RPN model represents a plan according to the previous definitions:

- The initial marking M_0 allows the plan execution to start.

- $Pre(p,t)$ (respectively $Post(p,t)$) equals $n > 0$ if the place p is an input (respectively output) place of a transition t and the valuation of the arc linking p to t (respectively t to p) is n. It equals 0 otherwise.
- The default value of a non-valuate arc equals to 1.

The RPN Semantics

A plan is executed by executing its actions. In RPN formalism, a transition models an action and its firing corresponds to executing an action. The dynamic refinement of an abstract transition when it has to be fired is an elegant way of the handling a conditional plan without developing all the situations at plan generation. In addition, it allows the interleaving of planning and execution. In our model, the planner chooses dynamically the best refinement according to the execution context depending on the Pre- and Post-Conditions (see cases (a), (b) and (c) in Fig.2). The choice heuristics are not detailed in this paper in order to focus on the plan management.

In the following the RPN semantics is given in order

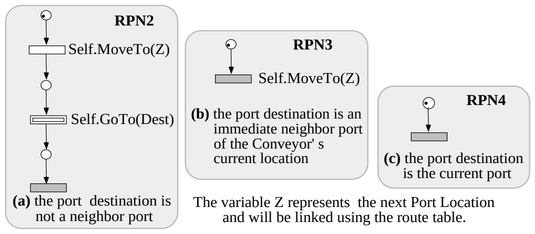
RPN2
Self.MoveTo(Z)
Self.GoTo(Dest)
(a) the port destination is not a neighbor port
RPN3
Self.MoveTo(Z)
(b) the port destination is an immediate neighbor port of the Conveyor' s current location
RPN4
(c) the port destination is the current port
The variable Z represents the next Port Location and will be linked using the route table.

Figure 2: Self.GoTo(Dest) Refinements

to illustrate the dynamic execution. An RPN models a plan. The successive states of a plan are represented in a tree (which may be considered as an execution tree). A node represents an initialized RPN and an arc represents an abstract transition firing.

Definition 4 A Plan State *A plan state is defined as a tree $Tr =< S, A >$ where:*

- *S is the set of nodes*
- *A is the set of arcs $a \in A$ such that:*
  - $a =< s, s' >$ if and only if s' is the child of s in Tr
  - An arc a is labeled by a transition which is called $Trans(a)$ where $Trans$ is a function from A to T with $trans(a) = t$
- *The initial state Tr_0 is a tree which is reduced to only one node s such that: $M(s) = M_0$ where M_0 is the initial marking.*

Example: Starting from the refinement RPN1 in Fig.1, the successive states of the plan are given in Fig.3 where case (P) gives the chosen initialized RPN associated with the call of the method Self.GoTo(Good.Src_Loc) and using the RPN2 refinement (see Fig.2); case (Q) illustrates the firing of the abstract action Self.GoTo(Dest); and case (R) illustrates the firing of the elementary action Self.MoveTo(Dest) according to the following definitions.

Notation: The index Tr of S (respectively of A and $Trans$) means that we consider the set of nodes S (respectively the set of arcs A and the transition $Trans$) relative to the tree Tr.

$Pre(.,t)$ (respectively $Post(.,t)$) is a mapping from P to N induced from Pre (respectively $Post$) by fixing to t as a second argument.

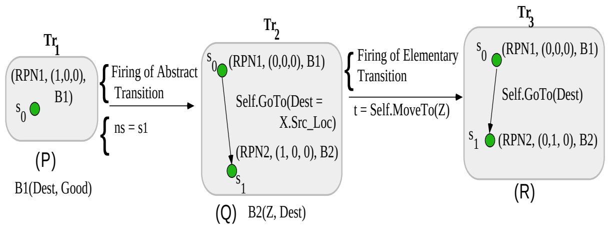

Figure 3: States of the Conveyor Plan

Definition 5 A Transition Firing Rule *A transition t is said to be fireable from a node $s \in S$ if and only if:*

- *$Pre(s,t) \leq M(s)$*
- *$R(s)$ is totally initialized*
- *The Pre-Conditions of the method associated with t are instantiated with $Call(t)$ and are valid.*

Definition 6 *A situation is said to be* **a failure situation** *if all the transition firing conditions are satisfied except for the Pre-Conditions of the associated method.*

Let us now examine the transition firing rule .

Firing Rules in RPN formalism

Definition 7 Elementary transition: *Let $t \in T_{elem}$. The firing of t from $s \in S$ produces the new tree Tr' with the new marking $M_{Tr'}$ of s such that:*

- *$Tr' = (S_{Tr'}, A_{Tr'})$ where:*
  - *$S_{Tr'} = S_{Tr}$ such that: $\forall s' \neq s, M_{Tr'}(s') = M_{Tr}(s')$ (i.e. the marking is unchanged $\forall s' \neq s$)*
  - *$A_{Tr'} = A_{Tr}$ such that: $\forall a \in A_{Tr}, Trans_{Tr'}(a) = Trans_{Tr}(a)$*
- *$M_{Tr'}(s) = M_{Tr}(s) + Post(.,t) - Pre(.,t)$*

This definition means that the only change that occurs concerns the node $s \in S_{Tr}$ by firing t. The effects of this change are to add $Post(.,t)$ to and subtracts $Pre(.,t)$ from the previous marking of s (see the node s_1 in case(R) in Fig. 3). In the plan, the change results corresponds to the applying of the Post-Conditions associated with the fired transition.

Definition 8 Abstract transition: *Let $t \in T_{abs}$. The firing of t from $s \in S$ produces the new tree Tr' with the new marking $M_{Tr'}$ of s such that:*

- $S_{Tr'} = S_{Tr} \bigcup \{ns\}$ *where ns represents an initialized RPN produced by the initialization mechanism of the method associated with t.*
- $A_{Tr'} = A_{Tr} \bigcup \{a\}$ *where* $a = < s, ns >$ *and* $Trans_{Tr'}(a) = t$
- $M_{Tr'}(s) = M_{Tr}(s) - Pre(., t)$

This definition means that the firing of t from the node s subtracts $Pre(., t)$ from the previous marking of s, creates a leaf ns in Tr' (see the node s_0 in case (Q) in Fig. 3) as a child of s. ns is labeled by one of the initialized RPN associated with the method call of t (see s_1 in case(Q) in Fig. 3). The new arc $< s, ns >$ is labeled by the abstract transition t.

Definition 9 *Let $Tr(s)$ be a tree with the node s as its root. The function $PRUNE(Tr, s)$ allows the tree Tr to be cut from the node s as follows:*
$Prune(Tr, s) \mapsto Tr'$ *such that* $Tr' = Tr \setminus Tr(s)$

Remark: $Tr' = \emptyset$ if the node s is the root of Tr.

Definition 10 End transition: *Let $t \in T_{end}$.The firing of t from $s \in S$ produces the new tree Tr' with the new marking $M_{Tr'}$ of s such that:*

- $Tr' = PRUNE(Tr, s)$
  Let s' be the immediate predecessor of s in the tree Tr and $a =< s', s >$ the arc labeled by t, then:
  - $\forall a \in A_{Tr'}, Trans_{Tr'}(a) = Trans_{Tr}(a)$
  - $\forall s'' \in (S_{Tr'} \setminus \{s'\}), M_{Tr'}(s'') = M_{Tr}(s)$
  - $M_{Tr'}(s') = M_{Tr}(s') + Post(., Trans_{Tr}(a))$

This definition means that the firing of an end action t_{end} belonging to an initialized RPN which has been generated by the firing of t_{abs} corresponds to the applying of the Post-Conditions of the call of the abstract method associated with t_{abs}. It closes the sub-net and adds the $Post(., t_{abs})$.

Concurrent Plan Management

Interacting situations are generally expressed in terms of binary relationships between actions and are often detected statically. The interleaving of planning and execution requires both static and dynamic detection of such situations which is ensured through the RPN semantics. Moreover, these situations are usually represented semantically which makes their handling difficult if not impossible. The syntactic aspects of such situations are often required to allow their operational management.

Interacting Situations through RPN

Here, planning and coordination aspects are merged, thus offering a number of advantages. When an agent cannot execute the refined plan he communicates it to another agent. Communication triggers a coordination mechanism which is based on the plan merging paradigm. Coordination is globally initiated by the incoming new plan (i.e. a new plan is submitted to an agent). The most important interacting situations handled by our approach include both positive and negative interactions.

Positive Interactions

- *Redundant Actions:* Actions are redundant if they appear in at least two plans belonging to two different agents, have identical Pre- and Post-Conditions, and the associated methods are instantiated by the same parameters except for the agent parameter (who has to perform the action). Hence, coordination assigns action execution to one of the agents. The agent who will perform the action has to provide his results. The others have to modify their plans by including a synchronizing transition.
- *Helpful Actions:* Actions are said to be helpful if the execution of one satisfies all or some of the Pre-Conditions of the others. Their execution will be respectively possible or favored. There are two ways of detecting such a situation:
  - dynamic detection: during the execution of an abstract action, the refinement of which encapsulates an elementary action which validates another action' s Pre-Conditions,
  - static detection: when the execution of one action precedes the execution of another and validates its Pre-Conditions.

Negative Interactions

- *Harmful Actions:* Actions are said to be harmful if the execution of one invalidates all or some of the Pre-Conditions of the others. Consequently, the execution of the latter will be respectively impossible or at an unfair disadvantage. Such a situation must be detected before the new plan execution starts in order to predict failure or deadlock. Our coordination mechanism introduces an ordering between the harmful actions as in (El Fallah & Haddad 1996) which provides a coordination algorithm (COA) for handling such interactions between n agents at once.
- *Exclusive Actions:* Actions are said to be exclusive if the execution of one momentarily prevents the execution of the others (e.g. their execution requires the same non-consumable resource). Detected dynamically, this situation occurs when an exclusive action has been started (i.e. an exclusive transition firing). In this case, execution remains possible but is deferred since it requires coordination with other executions.
- *Incompatible Actions:* Actions are said to be incompatible if the execution of one prevents the execution of the others (e.g. their execution requires the same consumable resource). In our model, such a situation models an alternative (e.g. two transitions share the same input place with one token). In this case, execution remains possible only if the critical resource can be replaced. In our approach, the planner uses heuristics based on two alternatives which

may be combined in order to avoid conflicts: if the conflict concerns an abstract action, the planner tries to substitute the current refinement. Otherwise, the method used will be replaced.

Plan Merging Paradigm

Hypotheses

- The RPN is acyclic and the initial net is reduced to an abstract transition before starting the execution.
- After plan generation, the plan actions have no negative interactions.
- The value of each unspecified condition is assumed to be unchanged as in STRIPS formalism.
- If the Pre-Conditions of an action method are valid, then the Post-Conditions are necessarily satisfied after its execution. Otherwise, no assumption is made about the validity of the Post-Conditions.

In the following, the handling of positive and negative interactions will be described through our case study. Let Clients $\{Cl_1, Cl_2\}$ and Conveyor Cv be three agents who are working out their respective plans.

Handling Positive Interactions: The main phases of the coordination algorithm are:

1. **Starting coordination:** An agent (e.g. Cl_1) has to perform a plan Π_1 corresponding to a leaf in his execution tree Tr but Π_1 is partially instantiated i.e. some plan methods associated with plan transitions have non-instantiated call parameters (e.g. $Y \in \Pi_1$ the TYPE of which is CONVEYOR). The agent (e.g. Cl_1) must find an agent (e.g. Conveyor) who will execute these methods. He starts a selective communication, based on his acquaintances, until he receives a positive answer. Let us assume that Cl_1 chooses Cv and then sends him Π_1.
2. **Recognition and unification:** The agent who receives Π_1 (e.g. Cv) detects the methods that are partially instantiated. Then, he examines his execution tree in search of the same methods. This phase is achieved with success if he finds an RPN (a node in his Tr) where appear all the methods to be instantiated (see Y.Transport(X) which is non-assigned in Π_1 and assigned in Π_2 in Fig. 4). Then, Cv triggers unification of the methods through their call parameters and instantiation of the variables w.r.t. the two plans. If both unification and instantiation are possible, Cv tries to merge the two plans Π_1 and Π_2.
3. **Structural merging through the transitions:** Cv produces a first merging plan Π_m (see Fig. 4) through the transitions associated with the previous methods and instantiates the call parameters. Then he checks that all the variables have been instantiated and satisfy both the Pre- and Post-Conditions.
4. **Consistency checking:** This phase is the keystone of the coordination mechanism since it checks the feasibility of the new plan which results from the structural merging. It is based on the algorithm using the Pre- and Post Conditions Calculus (PPCC) described in the following.

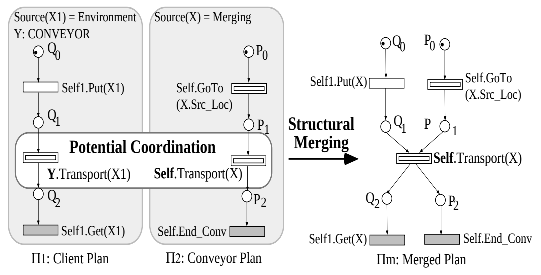

Figure 4: Structural Merging

Pre- and Post-Conditions Calculus Algorithm (PPCC) The PPCC algorithm is based on two phases:

i. **Reachability Tree Construction (RTC):** To begin with, we have to build an RT using a classical recursive right depth first algorithm which returns the tree root. Fig.5 shows the RT of the Structural Merging Plan Π_m given in Fig.4.

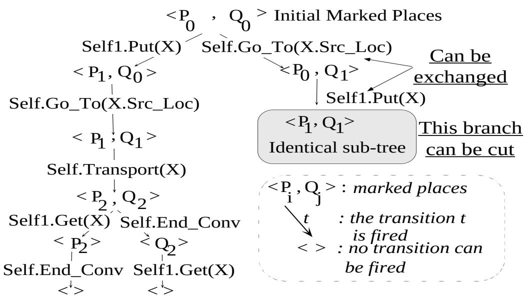

Figure 5: A Reachability Tree

ii. **Pre- and Post-Conditions Calculus:** This algorithm is called with the root of RT and with the parameters of the current execution context.

```
Function Is_Feasible(in s: node; in c: context): Boolean;
{context represents the value of methods' parameters and Tree is returned by RTC Algorithm}
begin
    for all arcs (s, s') in Tree do
     if Evaluate((s, s').Trans.Method.Pre, c) then
     {the Pre-Conditions in the given Context are valid}
      if not (Is_Feasible(s, Apply((s, s').Trans.Method.
                Post, c))) then return(false)
                   {there exists a non feasible sub-tree
                   i.e. the Post-Conditions are not valid}
      endif
     else return(false)
```

```
        endif
      endfor
      return(true)
    end {Is_Feasible }.
```

Property: For all plan Π modeled as an RPN, if the PPCC algorithm applied to the reachabilty tree (RT) of Π returns true (i.e. Π is consistent) and the environment is stable, then no failure situation can occur.
Discussion: In order to avoid a combinatorial explosion, there exist algorithms which allow to construct a reduced RT. In our context, the RT can be optimized as the following: let n_i and n_j be two nodes of RT. The RTC can be optimized through analysis of the method calls as follows:

- Independent Nodes: there is no interference between the Pre- and Post-Conditions of n_i on the one hand and the Pre- and Post-Conditions of n_j on the other hand, i.e. the associated transitions can be fired simultaneously whatever their ordering (i.e. the global execution is unaffected). Here, an arbitrary ordering is decided (e.g. Self.Put and Self.Go_To can be exchanged) which allows many sub-trees to be cut(the right sub-tree in Fig. 5).
- Semi-Independent Nodes: there is no interference between the Post-Conditions of n_i and n_j, i.e. the associated transitions don't affect the same attributes. If the exchanged sequences ($\{n_i, n_j\}$ or $\{n_j, n_i\}$) of firing transitions which lead to the same marking can be detected then the sub-tree starting from this marking can be cut. The obtained graph is then acyclic and merges the redundant sub-trees.

Handling Negative Interactions: Now an other agent (e.g. Cl_2) sends his plan to Cv who processes Π_1 (Cl_1's plan).
GCOA as Generalization of the COA Algorithm (El Fallah & Haddad 1996) The Conveyor starts a new coordination. The first and second phases are the same as in the case of positive interactions.He chooses the plan Π'_2. Negative interactions arise when the two refinements (Π_2 and Π'_2) have shared attributes (e.g. the boat volume constraint). Now, Cv has to solve internal negative interactions before proposing a merging plan to Cl_2. Again the GCOA is divided into two steps.

i. Internal Structural Merging by Sequencing: Cv connects Π_2 and Π'_2 by creating a place p_i for each pair of transitions (t_e, t_i) in $End(\Pi_2) \times Init(\Pi'_2)$ and two arcs in order to generate a merged plan Π_m:
Function Sequencing(in Π_1, Π_2: Plan): Plan;
{*this function merges* Π_1 *and* Π_2*; produces a merged plan* Π_m *and the synchronization places* }

```
    begin Let $T_E = \{t_e \in \Pi_1 / t_e$ is an end transition$\}$
      and $T_I = \{t_i \in \Pi_2 / t_i$ is an initial transition $\}$
         (i.e. $t_i$ has no predecessor in $Pi_2$)
      for all $(t_e, t_i) \in T_E \times T_I$ do
        Create a place $p_{e,i}$
        Create an input arc $IA_{e,i}$ from $t_e$ to $p_{e,i}$
        Create an output arc $OA_{e,i}$ from $p_{e,i}$ to $t_i$
          (i.e. $Post(p_{e,i}, t_e) = 1$ and $Pre(p_{e,i}, t_i) = 1$)
      endfor
    $\Pi_m$ := Merged_Plan ($\Pi_1, \Pi_2, \{p_{e,i}\}, \{IA_{e,i}\}, \{OA_{e,i}\}$)
    return ($\Pi_m, \{p_{e,i}\}$)
    end {Sequencing}
```

ii. Parallelization by Moving up arcs: Cv applies the PPCC algorithm to the merged net Π_m obtained by sequencing. If the calculus returns true then the planner proceeds to the parallelization phase by moving up the arcs recursively in order to introduce a maximum parallelization in the plan.

Procedure Parallelization(PL: Plan; in SP: Places): Plan;
{*both the plan PL and the set of synchronization places SP are obtained by sequencing; this algorithm returns a new plan PL modified by parallelization*}

```
begin
while ($\exists p \in SP / p$ is not stamped) do
begin
  Let $t\_in \in T(PL) / Post(p, t\_in)$
  and $t\_out \in T(PL) / Pre(p, t\_out)$
   ($t\_in$ is an input ($t\_out$ is an output) transition of $p$)
  $T\_Pred = \{t \in T(PL) / \exists q$ such that
     $Post(q, t\_Pred)$ and $Pre(q, t\_in)\}$
     (i.e. the set of the predecessor transitions of $t\_in$)
  $T\_Succ = \{t \in T(PL) / \exists q'$ such that
     $Pre(q', t\_Succ)$ and $Post(q', t\_out)\}$
     (i.e. the set of the successor transitions of $t\_out$)
  $P\_Pred = \{p \in P(PL) / Pre(p, t\_in)\}$ ($Pre(., t\_in)$)
   if ($\forall p_i \in P\_Pred$ ; $p$ is not stamped ) and
     ($\forall t_j \in T\_Pred$ ; the firing of $t_j$ is not started)
   then $\Pi_2$ = Copy(PL)
     forall $t \in T\_Pred$ do
      Create in $\Pi_2$ a place $q/$
        $Post(q, t) = 1$ and $Pre(q, t\_out) = 1$
     endfor
     for all $t \in T\_Succ$ do
      Create in $\Pi_2$ a place $q'/$
        $Post(q', t\_in) = 1$ and $Pre(q', t) = 1$
     endfor
     if PPCC (RT($\Pi_2$), Current_Context)
     then PL := $\Pi_2$
     else stamp(p)
     endif
   else stamp(p)
   endif
end
end {Parallelization}
```

This algorithm tries to move (or eliminate) the synchronization places. The predecessor transition of each synchronization place will be replaced by its own predecessor transition in two cases: the transition which precedes the predecessor transition is not fired or is not in firing. If both the Pre- and Post-Conditions remain valid, then a new arc replaces the old.
The result of this parallelization is to satisfy both Cl_1 and Cl_2 by executing the merged net Π_m.

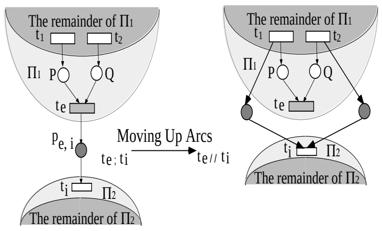

Figure 6: Sequencing and Parallelization

Remark: At each moving up the arcs, the PPCC algorithm is applied to the new net. The exchanged plans are the old ones augmented by synchronization places upstream and downstream. This algorithm can be optimized at the consistency control level. In fact, the coherence checking can be applied in incremental way to each previous plan Π_2.

Conclusion

This paper provides a formal model for distributed planning. Both plan representation and plan management are presented through the RPN formalism which offers the main advantages:

- representation and reasoning about simultaneous actions and continuous processes (e.g. concurrent actions, alternatives, synchronization, etc.),
- the formalism is domain-independent and supports complex plans with different levels of abstraction (i.e. only the relevant information is represented at the earlier phases),
- the formalism allows dynamic modifications with the associated verification (e.g. no structural inconsistency) and valuation methods (e.g. robustness),
- recursivity and dynamicity ensure the interleaving of execution and planning w.r.t. environment changes,
- plan reuse allowing agents to bypass the planning process in the case of similar situations according to the execution context (library of abstract plans which are the basic building blocks of the new plans),
- agents can skip some of the planning actions, detect conflicts early and reduce communication costs,
- execution control is dynamic in accordance with the associated refinement and therefore minimizes the set of revocable choices (Barrett & Weld 1993) because the instantiation of actions can be deferred,
- plan size remains controllable for the specification and plan complexity is tractable for validation.

Acknowledgments: This research was partly supported by France Telecom, CNET, under contract *CTI 904-15-4371-123*. The authors are grateful to Jean-Philippe Coumes (CNET Paris) and Gilles Deflandre (CNET Lannion) for their interesting discussions.

References

Books

Jensen, K. 1991. *High-level Petri Nets, Theory and Application*. Springer-Verlag.

Book Articles

Georgeff, M.P. 1990. Planning. *In Readings in Planning*. Morgan Kaufmann Publishers, Inc. San Mateo, California.

Werner, E. 1989. Cooperating agents: a unified theory of communication and social structure. *In L. Gasser and M.N. Huhns (eds.), Distributed Artificial Intelligence*. Vol II, pp 3-36. Pitman.

Osawa, E., and Tokoro, M. 1992. Collaborative Plan Construction for Multi-Agent Mutual Planning. *In E. Werner and Y. Demazeau, DECENTRALIZED A.I.3*. Elsevier/North Holland.

Journal Article

Decker, K.S., and Lesser, V.R. 1992. Generalizing the Partial Global Planning Algorithm. *In International Journal on Intelligent Cooperative Information Systems*.

Proceedings Papers

Alami, R., Robert, F., Ingrand, F., and Suzuku, S. 1994. A Paradigm for Plan-Merging and its use for Multi-Robot Cooperation. *In Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, Texas - USA.

Barrett, A., and Weld, D.S. 1993. Characterizing Subgoal Interactions for Planning. *In Proceedings of IJCAI-93*, pp 1388-1393.

Durfee, E.H., and Lesser, V.R. 1987. Using partial Global Plans to Coordinate distributed Problem Solvers. *In Proceedings of IJCAI-87*, Milan, 1987.

El Fallah Seghrouchni, A., and Haddad, S. 1996. A Coordination Algorithm for Multi-Agent Planning. *In Proceedings of MAAMAW'96*, LNAI:1038. Ed. Springer Verlag. Eindhoven, Netherlands.

Ephrati, E., and Rosenschein, J.S. 1995. A framework for the interleaving of Execution and Planning for Dynamic Tasks by Multiple Agents. *In Proceedings of ATAL'95*.

Martial, V. 1990. Coordination of Plans in a Multi-Agent World by Taking Advantage of the Favor Relation. *In Proceedings of the Tenth International Workshop on Distributed Artificial Intelligence*.