Validation and Verification of agent and multi-agent plans in dynamic environment

Said Brahimi
University of Guelma, Algeria
BP 24401, Guelma, Algeria
brahimi.said@yahoo.fr

Ramdane MAAMRI, Zaidi SAHNOUN
University of Constantine 2
Ali Mendjeli - BP : 67A, Constantine –Algeria
{rmaamri, sahnounz}@yahoo.fr

Abstract – In multi-agent systems evolving in complex and dynamic environments, the agents need to plan their tasks and to monitor its execution in order to deal with unpredictable situations. They must have plans that remain subject to continual updating, even during its execution. To cope with this issue, we proposed in previous works, SHPlNet, a model allowing to represent plans less sensitive to execution contexts, and to support run-time validation and verification. This paper aims to present a theoretical framework for the verification and validation of soundness and invalidity properties of partial hierarchical plans by analyzing their abstract level representation.

Keywords – multi-agent plan verification and validation**,** plans analysis, dynamic planning, planning and execution interleaving.

1. INTRODUCTION

In multi-agent systems, planning can allow agents to reason about their actions and interactions. In this context, plans can be used as procedures for resolving specific problems. Agents can be provided by plans as reusable procedural knowledges enabling them to behave in similar situations or conditions. Techniques used in this context are based on case based reasoning approach [1]. Furthermore, the plans serve as a guide that can help the agent to monitor its evolution in order to meet its goals (means-end reasoning). They also serve as a means for predicting future situations. Finally, plans serve as a tool for coordinating and monitoring activities for a set of agents [7]. By anticipating the actions of other agents, an agent can adjust and adapt its plan to avoid harmful behaviors and to benefit from the synergy of its plan with those of others.

In order to deal with the dynamics of complex environments, planning and execution must be interleaved. This is motivated by the following requirements:

- To reduce the time between the deliberation and the execution of actions to prevent these actions from becoming obsolete at the time of their execution;
- To reduce the complexity of planning and coordination by reducing the search space;
- To have information about the execution context through the execution of certain fragments plans.

To be able to succeed interleaving planning and execution, the agents must be able to reason about partially refined plans. They must be able also to take the appropriate decision regarding the initiation, suspension, repair, and the execution resuming of certain fragments of plans while continuing the execution of other's.

In previous works [10,11], we provided a formalism called SHPlNet that allows to represent hierarchical plans with multiple abstraction levels, by extending the Petri net and by taking advantage of HTN planning [4], CHiP [3], and the modular analysis of Petri nets idea [9]. SHPlNet can take into account the representation of flexible plans, and offers the necessary features allowing to monitor plans evolution, to handle plans interaction and interdependency, and to control resources evolution at run time. Furthermore, SHPlNet can allow a modular representation of plans. Firstly, there is a clear distinction between the abstraction levels of hierarchical plan. Secondly, there is clear separation between tasks and synchronization constraints. Within this aspect, the analysis of plans can be done in a modular way, and therefore, their updating may be simplified. Note that the modular representation of the plan can facilitate the revision of some decisions of planning and coordination in order to best meet the evolutionary aspect of the system. These aspects are suitable to support the interleaving of planning, execution, and coordination.

This paper aims to provide theoretical framework for analyzing and verifying agent and multi-agent plans at run-time. We explain and demonstrate how to verify the soundness property of partial plan by analyzing only its abstract level.

In fact, there are some key related works that dealt with this question, like as [8, 6, 2, 5,12,13]. Recursive Petri net proposed in [8] and extended in [6] is a more expressive formalism to represent hierarchical plans. The distinction between abstract and primitive transitions and the firing rules principles are all features enabling to verify many properties of only complete plans. While its power to express a wide range of agent plans, the recursive Petri net suffers from the inability to explicitly represent the interaction and interdependence of concurrent tasks and its inability to reason on abstract tasks.

The work proposed in [5] is based on the idea of propagating the information about related resources for each plan. This information is used to verify some properties about the validity and the quality of plan and to control its execution in some context. They used formalism based on the extension of timed automata. Similar to the formalism used here [10], the hybrid automata allows model-checking of important properties like reachability, security, liveness, deadlock absence… however, the plans have one level of abstraction and must be complete to be checked.

In [12, 2, 13] the authors provided a framework for representing the plan based on the Petri net. In these works, the plans are represented at one level of abstraction. Like the previous works, these approaches are not suitable to represent and to check partial plans.

The rest of this paper is structured as follows. Section 2 outline the key properties related to partial plans. In section 3, we present preliminary formalism, HPlNet, (Hierarchical-Plan--Net) and the underlying properties. In section 4, we illustrate the representation of hierarchical plans and its synchronization. We also explain and demonstrate how to verify these plans. Finally, we conclude our paper.

2. AGENT AND MULTI-AGENT PLAN REQUIREMENT

In multi-agent systems where the planning and execution process are interleaved, agents must be able to represent, verify (and validate), and monitor partial plans at run-time. They must be able to verify the following properties:

Soundness: soundness property denotes that the plan:

- Must not contain dead tasks. It is generally not important to incorporate unnecessary tasks in a plan;
- Should not contain tasks that can be performed more than once. Therefore, only one instance of a plan requires at most one instance for each task or decomposition method; and
- Must be completely executed. It must be correctly refined to ground and executable plans. Its execution must not lead to blocking situations.

For multi-agent plan, the soundness property denotes that there is no conflict between the tasks of one or more individual plans

Flexibility: a flexible plan is a sound one that can be refined to several (at least two) ground plans. Therefore, its execution can be flexible.

Feasibility: a plan is feasible if it can be executed correctly. A partial plan is called feasible if it can be refined to at least one executable and complete plan.

Invalidity: a plan is invalid if it cannot be executed correctly. A partial plan is called invalid if it cannot be refined to any executable and complete plan. For multi-agent plan, the Invalidity property denotes that the plan contains an unsolvable conflict between tasks (of one or more individual plans).

Agents that interleave planning and execution must be able to identify and to verify these properties in order to behave in an appropriate manner and to take suitable decision about the initiation, suspension, repair, and execution resuming of certain (fragments of) plans while continuing the execution of others.

3. SIMPLE HIERARCHICAL PLANS REPRESENTATION

In this section, we present the formalism used to represent simple hierarchical plans where the tasks are hierarchically organized. For more detail about the hierarchical plan representation, the reader can refer to [11].

3.1. Hierarchical Tasks

3.1.1. Hierarchical-Task-Petri-Net (HTPN)

In this subsection, we provide a formalism, that we call Hierarchical Plan net (HTPN), to represent hierarchical plans based on extension of Petri net. The key idea consists of defining a tree-based structure. Each node in this tree is a special Petri net representing totally and partially ordered tasks networks. Formal definition of HTPN is as follow.

***Definition 1* (*HTPN*).**
A Hierarchical-Task-Petri-Net (HTPN) is defined by the tuple $\wp = (P, T, F, W, s, e, R)$ where (P, T, F, W) is a Petri net, and:

- *$T = T_{elem} \cup T_{abs}$ is a finite set of transitions, disjoint union of elementary (T_{elem}) and abstract transitions (T_{abs}). T_{abs} may be empty;*
- *$s \in P$ is a particular place representing the source place ($\bullet s = \emptyset$);*
- *$e \in T$ is a particular transition representing the end transition ($e\bullet = \emptyset$);*
- *$R \subseteq T_{abs} \times HTPN$, $R = \cup_{t \in T_{abs}} R_t$ is a finite set of refinement rules for all abstract transitions ($T_{abs} = \emptyset \Leftrightarrow R = \emptyset$); each rule $r \in R$ associates to a transition $t \in T_{abs}$ a refinement HTPN. R_t denotes the set of all rules can be used to refine the task t;*
- *(P, T, F, W) is a Petri net to have either of the following two structures:*
  - *All transitions and places belong to a single path from s to e, i.e. $P = \{s\} \cup \bigcup_{i=1}^{m}\{p_i\}$, $T = \bigcup_{i=1}^{m}\{t_i\} \cup \{e\}$, and $F = \{(s, t_1, 1)\} \cup \bigcup_{i=1}^{m}\{(t_i, p_i)\} \cup \bigcup_{i=1}^{m-1}\{(p_i, t_{i+1})\} \cup \{(p_m, e)\}$. in this case, (P, T, F, W, s, e) is called Sequential-Task-Petri-Net (Sequential-TaskPN) node*
  - *All transitions (except f and e) belong to parallel flows initiated by a fork transition f (having a single entry place s), and should be joined by the end-transition e, i.e. $P = \bigcup_{i=1}^{m}\{p_i, p'_i\} \cup \{s\}$, $T = \bigcup_{i=1}^{m}\{t_i\} \cup \{f, e\}$, $F = \{(s, f)\} \cup \bigcup_{i=1}^{m}\{(f, p_i)\} \cup \bigcup_{i=1}^{m}\{(p_i, t_i)\} \cup \bigcup_{i=1}^{m}\{(t_i, p'_i)\} \cup \bigcup_{i=1}^{m}\{(p'_i, e)\}$, the tasks are connected to source place by a fork transition $f \in T_{elem}$. in this case, (P, T, F, W, s, e) is called Parallel-Task-Petri-Net (Parallel-TaskPN) node.*

A HTPN $\wp = (P, T, F, W, s, e, R)$ may be considered as a tree of nodes. The root of this tree is (P, T, F, s, e) where $T_{abs} = \{t_0\}$ and $T = \{e, t_0\}$, t_0 is highest-level task of $\wp$. The leaves of the tree are nodes where $R = \emptyset$ ($T = T_{elem}$). The intermediate nodes are characterized by $R \neq \emptyset$ ($T_{abs} \neq \emptyset$). Abstract transitions model abstract (or compound) tasks and elementary transitions model atomic (or elementary) tasks.

As we explained above, the HTPN may be viewed as a tree of nodes having TaskPN structure. Hence, the state of HTPN must take into consideration the marked places of these nodes. The state of HTPN must also take into account the refinement state of abstract tasks. We hence extend the marking concept of ordinary Petri net to define a marking that deals with the characteristic of HTPN (definition 2).

Definition 2 (*Extended marking of HTPN*).
An extended marking of HTPN hN is defined by the tree $Tr = (N, n_0)$ such that N is the set of node; each node $n \in N$ is a tuple (Pl, Mp, Mt) such that Pl is a node in $\wp$; $Mp: P(Pl) \to \{0,1\}$, and $Mt: T_{abs}(Pl) \longrightarrow N \cup \{\varepsilon\}$ (ε denotes the absence of tokens) is a marking function of abstract transitions; $n_0 \in N$ is the root of tree; a node n' is the child of n in Tr if and only if $\exists t \in T_{abs}(Pl(n))$ such that $(t, Pl(n')) \in R(Pl(n))$, and $Mt(n)(t) = n'$

One can note that:

- The tree structure of Tr is implicitly defined, a node n' is the child of n in Tr iff $\exists t \in T_{abs}(Pl(n))$ such that $(t, Pl(n')) \in R(Pl(n))$, and $Mt(n)(t) = n'$.
- The initial extended marking is Tr_0 such that $Tr_0 = (\{n_0\}, n_0)$ where $n_0 = (Pl, Mp_0, Mt_0)$, $Mp_0 = s(Pl(n_0))$, and Mt_0 is the initial marking (where $Mt(t_0) = \varepsilon$);
- The final marking, Tr_f, is an empty tree (that has no node), noted by $Tr_f = \perp$.

The extended marking of HTPN is considered as a state indicating the activated nodes and the state of each place and each abstract transition in these nodes. A step sp between two marking states Tr and Tr', denoted by $Tr \xrightarrow{sp} Tr'$, concerns the firing of elementary transition, end-transition, or (selected) refinement-rule. The firing will be possible only if the transition or the refinement rule is enabled (Definition 3).

Definition 3 (*Firing conditions in HTPN*).
Given an extended marking Tr, a node n in Tr, a step sp is enabled in Tr, denoted by $Tr \overset{sp}{\rightarrow}$, iff:
- $sp \in T(Pl(n)) \Longrightarrow \forall p \in \bullet sp, Mp(n)(p) = 1;$
- $sp \in R_t \Rightarrow Tr \overset{t}{\rightarrow}$.

In the definition 4, we formalize firing rules.

Definition 4 (*Firing rules of HTPN*).
Let Tr be an extended marking and n be a node in Tr, the firing of a step sp leads to the extended marking Tr', denoted by $Tr \overset{sp}{\rightarrow} Tr'$, such that:
***Case 1**: $sp \in T_{elem}(Pl(n))$:*
- $\forall p \in P(Pl(n)),\ Mp'(n)(p) = Mp(n)(p) + W(Pl(n))(t,\mathrm{p}) - W(Pl(n))(\mathrm{p}, sp)$ and
- $N(Tr) = N(Tr')$

***Case 2**: $sp \in R_t(Pl(n))$:*
- $\forall p \in P(Pl(n)), Mp'(n) = Mp(n) - \mathrm{W}(Pl(n))(\mathrm{p},\mathrm{t})$
- $N(Tr') = N(Tr) \cup \{n' = (Pl, Mp_0(Pl), Mt_0(Pl)) / r = (t, Pl)\}$
- $Mt'(n)(t) = n'$

***Case 3**: $sp = e(Pl(n))$ and n is child of n' by $r = (t', Pl(n))$:*
- $Mt'(n')(t') = \varepsilon,$
- $N(Tr') = N(Tr) - \{n\}$
- $Mp'(n') = Mp(n') + \mathrm{W}(Pl(n'))(t',.)$

***Particular case**: $t = e(Pl(n_0))$: $Tr' = \perp$*

The concept of extended marking, enabled transition (or refinement rule), and firing rules in HTPN allow to have explicit representation of hierarchical task state and its evolution.

3.1.2. Properties of HTPN

We present some properties of HTPN, especially the soundness property.

Definition 5 (*soundness of an HTPN*).
An HTPN is sound iff:

- *There are no dead transitions: all transitions must be quasi-lives;*
- *Each step must not be fired more than once;*
- *Proper termination: for each state Tr reachable from the initial state Tr_0, it is always possible to reach the unique final state $\perp$.*

***Theorem 1**. Each HTPN is sound.*

***Proof**. Pursuant to finite (the finite number of nodes component) character of the tree representing the HTPN, the absence of recursion, and soundness of nodes (because they have a structure TaskPN), for demonstrating the three conditions cited in the definition 5 (about the soundness of a HTPN) it suffices to prove that: a) the firing of each transition in each node is always possible; and b) each transition in each node must not be fired more than once. By its simplified structure, it is very easy to prove that TaskPN is sound. So is the case for nodes of a HTPN, because each node has a control structure of a TaskPN. On the other hand, the choice of the refinement-rule to use did not depend on the marking; it just depends on whether the transition to refine is enabled. The firing condition of this transition depends only on the marking of active node marking where this transition is located.* ■

The soundness property implies that the number of accessible states of a HTPN is bounded. Therefore, the reachable extended markings graph is also bounded. It indicates also that all paths in the reachable graph lead to a single final state. Each path contains the refinement rules and elementary transitions (including end and fork transitions) to select in order to perform the task t_0, the highest level of abstraction. Among these transitions or refinement rules appearing in the reachable graph, we want to distinguish between two types of transitions:

Definition 6 (*Necessary and Eventual transition*).

Let $(\wp, Tr_0)$ be a market HTPN and t be a transition in $\wp$. t is:

- *Necessary Transition iff t must be fired to reach some final state, whatever path to take. Formally: $[Tr_0 \overset{\sigma}{\rightarrow} Tr_f] \Longrightarrow [\sigma = \sigma', t, \sigma'' \vee (\sigma = \sigma', r, \sigma'' / r \in R_t)]$;*
- *Eventual transition, iff t can be fired to reach a final state. Formally: $\exists\, \sigma / Tr_0 \overset{\sigma}{\rightarrow} Tr_f$ and $(\sigma = \sigma', t, \sigma'' \vee (\sigma = \sigma', r, \sigma'' / r \in R_t))$.*

Necessary transitions correspond to the tasks that *must* be performed to accomplish the task t_0 of a plan. However, Eventual transitions correspond to the tasks that *may* be performed to accomplish the task t_0 of a plan.

3.2. Hierarchical Plan

3.2.1. Hierarchical-Plan-Net (HPlNet)

We provide a formalism, that we call Hierarchical-Plan-Net (HPlNet), making an extension of HTPN by adding information about the resources to consume and to produce. The formal definition is given below (definition 7).

Definition 7 (*Hierarchical-Plan-Net*).
A Hierarchical-Plan-Net (HPlNet) is defined *by the tuple* $\wp = (P, T, F, W, s, e, Res, R)$ *where:*

- (P, T, F, W, s, e, R) *is HTPN where* $T = \{t_0, e\}$ *and* t_0 *is the single abstract transition of the highest level;*
- $Res: T \to \mathcal{RQ} \times \mathcal{RQ}$ *is a function defining the sets of the consumption and the production associated to each transition.* $\mathcal{RQ} = \{(rs_1, q_1), \ldots, (rs_n, q_n)\}$ *where* $rs_i \in \mathcal{R}$ *is resource name and* $q_i = (x, y)$ *represents the lower* (x *or* q_i^-) *and the upper* (y *or* q_i^+) *quantity (number) of* rs_i. $\mathcal{R}$ *is the set of all resources.*

We denote by $Res(t).cons$ (resp. $Res(t).prod$) the set of the consumption and the production of the task represented by t, we can also write $Res(t) = (cons, prod)$. If (rs_i, q_i) is associated to an elementary transition then $q_i^- = q_i^+$. In this case, we can represent q_i by a simple value. The end-transition and fork-transition are not related to any resource. Hence, if t is one of these two kinds of transitions then $Res(t) = (\emptyset, \emptyset)$. Graphically we omit the representation of the empty sets related to the fork and end-transitions.

The state of HPlNet is represented by an extended marking that inherits all features of state representation of HTPN, and takes into account the state of the available resources (that we call execution context). Its formal definition is as follows (definition 8).

Definition 8 (*Extended marking of HPlNet*).
An extended marking of HPlNet $\wp = (P, T, F, W, s, e, Res, R)$ *is defined by the tuple* (Tr, Cxt) *where:*

- Tr *is the extended marking of* (P, T, F, W, s, e, R);
- $Cxt = \{(rs_1, qt_1), \ldots, (rs_n, qt_n)\}$ *is a finite set of available resources, where* qt_i *is the amount of resource* rs_i. *We write* $qt_{Cxt}(rs)$ *to denote the amount of resource* rs *in* Cxt.

The initial extended marking is defined by (Tr_0, Cxt_0), such that Tr_0 is the initial extended marking of (P, T, F, W, s, e, R) and Cxt_0 is the initial state of available resources. The final extended marking is (Tr_f, Cxt) in which Tr_f is an empty tree (that has no node). The tuple $(\wp, (Tr_0, Cxt_0))$ represents the marked HPlNet.

As the case of HTPN, a step in HPlNet concerns an elementary-transition, end-transition, or a refinement-rule. However, the steps firing in HPlNet must take into account the summary information about the resources associated to the transitions. The formalization of steps firing in HPlNet is summarized by definition 9 and 10.

Definition 9 (*Firing conditions in HPlNet*).
Given an extended marking (Tr, Cxt), *a node* n *in* Tr, *a step* sp *is enabled in* (Tr, Cxt), *denoted by* $(Tr, Cxt) \overset{sp}{\to}$, *iff:*

- $Tr \overset{sp}{\to}$ *and*
- $sp \in T_{elem}(Pl(n)) \Rightarrow \forall (rs, q) \in Res(sp).cons, qt_{Cxt}(rs) \geq q^-$ (Note that $q^- = q^+$).

Definition 10 (*Firing rules in HPlNet*).
Let (Tr, Cxt) *be an extended marking,* n *be a node in* Tr, *the firing of a step* sp *leads to the extended marking* (Tr', Cxt'), *denoted by* $(Tr, Cxt) \overset{sp}{\to} (Tr', Cxt')$, *such that:*

Case 1*:* $sp \in T_{elem}(Pl(n))$

- $Tr \overset{sp}{\to} Tr'$;
- $Cxt' = \{(rs_i, q_i - q_i' + q_i'')\}$ such that $(rs_i, q_i) \in Cxt$, $(rs_i, q_i') \in Res(sp).cons$ and $(rs_i, q_i'') \in Res(sp).prod$.

Case 2*:* $sp \in R_t(Pl(n))$ *or* $sp \in e(Pl(n))$

- $Tr \overset{sp}{\to} Tr'$;
- $Cxt' = Cxt$.

We note that the only difference between the firing rules in HTPN and HPlNet is the firing of elementary transitions whose execution context must be updated according to the amount of resources to consume and produce.

3.2.2. Properties of HTPN

If the soundness property of HTPN is ensured, the soundness property of HPlNet is not guaranteed. The soundness of a plan represented by HPlNet depends exactly on the availability of resources in the initial state (initial context Cxt_0). Therefore, the soundness checking of an HPlNet is only limited to the verification of quasi liveness property of all transitions and proper termination criterion, because the property on the absence of a multiplicity of firing step is inherited from HTPN. We note that the soundness of an HPlNet relaxes the criterion of the uniqueness of the sinks state in terms of the context, Cxt. This is justified by the fact that the diversity of firing sequence leads to the consumption and production of different amounts of resources.

***Lemma* 1**. *In marked HPINet $(\wp, (Tr_0, Cxt_0))$ there is no step that can be fired more than once.*

Proof. *By contradiction, we assume that there is (at least) a step that can be fired more than once. Let sp a step such that: $(Tr_i, Cxt_i) \overset{sp}{\rightarrow} (Tr_j, Cxt_j)$ and $(Tr_i, Cxt_i) \in A(\wp, (Tr_0, Cxt_0))$. If sp can be fired again then there can be a state $(Tr_k, Cxt_k) \in A(\wp, (Tr_j, Cxt_j))$ such that $(Tr_k, Cxt_k) \overset{sp}{\rightarrow}$. By definition, $(Tr_i, Cxt_i) \overset{sp}{\rightarrow} (Tr_j, Cxt_j)$ implies $Tr_i \overset{sp}{\rightarrow} Tr_j$, and if $Tr_j, Cxt_j) \overset{*}{\rightarrow} (Tr_k, Cxt_k)$ that implies $Tr_j \overset{*}{\rightarrow} Tr_k$, then $Tr_k \nrightarrow^{sp}$, and consequently $(Tr_k, Cxt_k) \nrightarrow^{sp}$. Therefore, sp cannot be fired more than once.* ■

Theorem 2. *A marked HPINet $\wp, (Tr_0, Cxt_0))$ is bounded.*

Proof. *Direct consequence of the Lemma* 1. ■

The boundedness of HPINet means that the number of nodes in the marking tree is limited, the places and abstract transitions in each node are bounded, and the amount of each resource in the context is limited. Boundedness of HPINet can help to analyze the plans represented by HPINet by exploiting their Reachability Graph.

Pursuant to the lemma 1, we may decide that HPINet is sound if it is quasi-live and proper termination criterion is checked.

In fact, there is dependence between these two criteria. Each termination of HPINet, that is not proper, is termination when there are steps (exactly transitions) which cannot be fired, i.e. blocking.

Lemma 2. *The proper termination criterion of marked HPINet $(\wp, (Tr_0, Cxt_0))$ holds if all its transitions are quasi-lives.*

Proof. To demonstrate that the proper termination criterion of marked HPINet $\wp, (Tr_0, Cxt_0))$ holds if it is quasi-live, we proceed to assume that the proper termination criterion did not hold and demonstrate that HPINet is not quasi-live. We assume that the termination is not proper, then, there exists a sink state (Tr_p, Cxt_p) reachable from the initial state such that $Tr_p \neq Tr_f$. By projecting HPINet on HTPN and according to the theorem 1, if $Tr_p \neq Tr_f$ then there must be at least one firing sequence σ (containing steps that have not been fired) such that $Tr_p \overset{\sigma}{\rightarrow} Tr_f$. Therefore, if the state $Tr_p, Cxt_p)$ is sink then the sequence σ must begin with an elementary transition t that is enabled vis-a-vis *Tr*, but not enabled Cxt_p ($Cxt_p \nrightarrow^{t}$), so $(Tr_p, Cxt_p) \nrightarrow^{t}$ and then t is a dead transition, so HPINet is not quasi-live. ■

***Theorem* 3.** A marked HPINet $(\wp, (Tr_0, Cxt_0))$ is sound if *all its transitions are quasi-lives*.

Proof. Pursuant to the lemma 1, in each marked $\wp, (Tr_0, Cxt_0))$ steps cannot be fired more than once. On the other hand, according to the lemma 2, the proper termination holds if the marked HPINet is quasi-live. So marked HPINet $\wp, (Tr_0, Cxt_0))$ is sound if it is quasi-live. ■

The most simple and intuitive method to verify that a marked HPINet $(\wp, (Tr_0, Cxt_0))$ is sound, i.e. is quasi-live, is to analyze the reachability graph. This can be done by checking that each transition or refinement rule belongs to a path from the initial state (Tr_0, Cxt_0), and leads to a terminal and sink state (Tr_f, Cxt).

In this regard, we define (Definition 11) the concepts of run, feasibility, flexibility, safe state, and invalidity.

Definition 11 (*run,* feasibility, flexibility and safe state, *invalidity*).

Let $\wp$ be a plan represented by an HPINet and Cxt_0 be an initial execution context:

- *A run for $\wp$ is an enabled steps sequence (decisions) σ in Cxt_0, such that $(Tr_0, Cxt_0) \overset{\sigma}{\rightarrow} Tr_f, Cxt)$;*
- *$\wp$ is executable in the context Cxt_0, or its execution is feasible, iff there is at least a run for it ;*
- *$\wp$ is flexible in the context Cxt_0 iff it is sound and have several runs (at least two) ;*
- *A state (Tr_i, Cxt_i) reachable from (Tr_0, Cxt_0) is safe-state iff there is steps sequence σ such that $(Tr_i, Cxt_i) \overset{\sigma}{\rightarrow} (Tr_f, Cxt)$;*
- *$\wp$ is invalide in the the context Cxt_0* if it has no run.

A run is a safe execution of an HPINet. We note that HPINet is associated with a set of possible runs. This is justified by the presence of several refinement rules (for an abstract transition) and/or by the presence of concurrent tasks that can be triggered in a different order. The set of possible runs correspond to all possible paths between the initial extended marking and final extended markings.

The analysis of the plans by exploiting their reachability graph is inappropriate because it can cost the complexity of calculation. In addition, it does not properly exploit the multiple levels of abstraction characterizing HPlNet. Plans analysis can be improved by using summary information associated with the abstract transitions. With this information, the verification of certain key properties is possible by analyzing highest level of abstraction of plans only.

By projecting on the analysis of executing plan, we propose some properties on the proper termination of its evolution. This concerns the safe, possible, and impossible termination. These properties are defined as follows.

Definition 12 ***(safe, possible, and impossible proper Termination).***

Let (Tr, Cxt) be the current execution state of a plan $\wp$ represented by an HPlNet. The proper termination of the execution of $\wp$ in (Tr, Cxt) is:

- *safe, iff the state (Tr, Cxt) is safe;*
- *Impossible, iff the available resources in Cxt are not sufficient to complete all remaining subtasks. The execution of t_0 never reaches an final state:* $\exists (rs, q) \in Res(t_0).cons, qt_{Cxt}(rs) < q^-$;
- *Possible, iff the reachability of a final state is uncertain. According to a particular order of steps firing, a final state may be reached :* $\forall (rs, q) \in Res(t_0).cons, qt_{Cxt}(rs) > q^-$ *and* $\exists (rs, q) \in Res(t_0).cons, qt_{Cxt}(rs) \in [q^-, q^+[$.

4. HIERARCHICAL PLANS WITH SYNCHRONIZATION

4.1. HPlNet with synchronization (SHPlNet)

In this section, we present SHPlNet (Hierarchical Plan Net with Synchronization), an extension of HPlNet [10] that deals with the interaction between tasks and plans.

One can note that the execution of plans may lead to critical situations due to the potential conflict between the tasks sharing critical resources. The conflict may occur between tasks in an individual agent plan or between tasks belonging to different agents' plans. To address these conflicting situations, the tasks in the plans must be synchronized and some decomposition methods must be avoided. To be able to represent plans taking into account the synchronization between tasks, we extend HPlNet by adding features allowing to impose an execution order between parallel tasks, and to avoid the activation of some refinement rules. The idea is to add a separate module grouping synchronization and inhibition constraints. We use an ordinary Petri net, that we call synchronization net, to represent this module. We call the new formalism *Hierarchical-Plan-Net with Synchronization* (SHPlNet), its formal definition is as follows (definition 13).

Definition 13 (*SHPlNet*).

Hierarchical-Plan-Net with Synchronization (SHPlNet) is defined by the tuple: $(P, T, F, s, e, Res, R, S)$ *where* (P, T, F, s, e, Res, R) *is HPlNet;* $S = (\cup_i P_i^s, \cup_i T_i^s, \cup_i W_i^s, \cup_i F_i^s)$ *is a Petri net such that* $(P_i^s, T_i^s, F_i^s, W_i^s)$ *can have either of the following structure:*

- $(\{p, rs\}, \{t_1, t_2\}, \{(t_1, p), (t_1, rs), (p, t_2), (rs, t_2)\}$ $\{((t_1, p), 1), ((t_1, rs), x), ((p, t_2), 1), ((rs, t_2), x)\})$ *such that* $x \leq q^-$ *where* $(rs, q) \in Res(t_1).prod$, *to represent a production-consumption relationship to exchange* x *unites of the resource* $rs \in \mathcal{R}$ *between two necessary and concurrent tasks (a producer and a consumer);*
- $(\{p\}, \{t_1, t_2\}, \{(t_1, p), (p, t_2)\}, \{((t_1, p), 1), ((p, t_2), 1)\})$ *to represent a temporal order between two necessary and concurrent tasks,* t_1 *and* t_2*;*
- $(\{p\}, \{r\}, \{(p, r)\}, \{((p, r), 1)\})$ *to represent an inhibition of a refinement rule* $r \in R_t$ *for an abstract and necessary task t.* $R_t \cap T(S) \neq R_t$
- *Each transition defined in* $T(S)$ *is a transition or a refinement rule defined in HPlNet.*

The Petri net S defined in the definition 13 constitutes a coordination module including synchronization constraints that enforce an execution order and ensure the exchange of some quantities of resources between concurrent tasks (explicit positive interaction). It includes also constraints that enforce the selection of one refinement rule.

One can note that several causal relationships can be related to the same resource that is represented by a unique place. In order to protect the amounts of resources to be exchanged between different peers of transitions (producers and consumers), we propose to add, for each causal relationship, a second temporal constraint between the same transitions (using another place). This additional constraint allows the producer task to notify the availability of expected amount of resource to consumer task.

The state of plan $\wp$ represented by SHPlNet is modeled by the extended marking of SHPlNet (definition 14) that takes into account marking state of synchronization net.

Definition 14 (*extended marking of SHPlNet*).

An extended marking of SHPlNet $\wp = (P, T, F, W, s, e, Res, R, S)$ is defined by the tuple (Tr, Cxt, Ms) such that:

- *(Tr, Cxt) is the extended marking of HPlNet $(P, T, F, W, s, e, Res, R)$; and*
- *Ms is a marking of S; the initial and the final marking of S is $Ms_0 = 0$.*

The steps of SHPlNet are firing of an elementary-transition, end-transition, or refinement-rule. The definition 15 and 16 below state respectively the firing condition and firing rules.

Definition 15 (*firing condition in SHPlNet*). *Given an extended marking (Tr, Cxt, Ms), a node n in Tr, a step sp is enabled in (Tr, Cxt, Ms), denoted by $(Tr, Cxt, Ms) \overset{sp}{\rightarrow}$, iff:*

(i) $(Tr, Cxt) \overset{sp}{\rightarrow}$;

(ii) $sp \in T_s \Longrightarrow Ms \overset{sp}{\rightarrow}$;

(iii) $sp \in R_t(n) \wedge t \in T_s \Longrightarrow Ms \overset{t}{\rightarrow}$.

In the definition 15, the condition (*ii*) states that all possible steps appearing in S cannot be enabled if they are not enabled in M_s. The condition (*iii*) states that an abstract transition appearing in S cannot be refined if it is not enabled in M_s.

Definition 16 (*firing rules in SHPlNet*). Let (Tr, Cxt, M_s) *be an extended marking,* n *be a node in* Tr*, the firing of a step* sp *leads to the extended marking* (Tr', Cxt', Ms')*, denoted by* $(Tr, Cxt, Ms) \overset{sp}{\rightarrow} (Tr', Cxt', Ms')$*, if and only iff:*

***Case 1**: $sp \in T_{elem}(Pl(n) \cup \{e(Pl(n_0))\}$:*
- $(Tr, Cxt) \overset{sp}{\rightarrow} (Tr', Cxt')$;
- $sp \in T_s \Longrightarrow Ms \overset{sp}{\rightarrow} Ms'$; and
- $sp \notin T_s \Longrightarrow Ms' = Ms$

***Case 2**: $sp \in R_t(Pl(n))$:*
- $(Tr, Cxt) \overset{sp}{\rightarrow} (Tr', Cxt')$;
- $t \in T_s \Longrightarrow [\forall p \in P_s, Ms'(p) = M_s(p) - W_s((p, t))]$, *and*
- $t \notin T_s \Longrightarrow Ms' = Ms$

***Case 3**: $sp = e(Pl(n))$ and n is child of n' by $r = (t', Pl(n))$:*
- $(Tr, Cxt) \overset{sp}{\rightarrow} (Tr', Cxt')$;
- $sp \in T_s \Longrightarrow Ms \overset{sp}{\rightarrow} Ms''$;
- $sp \notin T_s \Longrightarrow Ms'' = Ms$;
- $t' \in T_s \Longrightarrow [\forall p \in P_s,\ Ms'(p) = Ms''(p) + W_s((t', p))]$; *and*
- $t' \notin T_s \Longrightarrow [Ms' = Ms'']$.

In the definition 16 the case 1 states that the firing on an elementary transition appearing in S leads to its firing in S. The case 2 states that the firing of refinement rule of a transition that appear in S leads to the consumption of tokens. The production of the tokens will be after the firing of end transition of the node refining the abstract transition (case 3).

4.2. Properties of a SHPlNet

In the same way as a plan represented by HPlNet, a plan represented by SHPlNet can be analyzed from an abstract level. Based on the summary information about the resources associated to transitions, we can decide that the proper termination of plan execution is sure, possible, or impossible, provided that this plan is cycles free. The cycles lead always to deadlock state that prevents the execution of any task of $\wp$. We formally define the concept of cycle and acyclic plan as follows.

***Definition** 17 (**Cycle in SHPlNet and Acyclic Plan**).*
Let $\wp = (P, T, F, W, s, e, Res, R, S)$ be a SHPlNet and n_1, n_2 be two nodes in $\wp$. $\wp$ includes a cycle represented by $t_0 \prec t_1 \prec \ldots \prec t_m \prec t_0$, iff:

(i) for each $t_i \prec t_j$ such that $i \in \{0,1,\ldots,m\}$ and $j = (i+1)mod(m)$,
- *$\{(t_i, p_i), (p_i, t_j)\} \in F(S)$, or*
- *$\{(t_i, p_i), (p_i, t_j)\} \in F(n_1)$, or*
- *$t_i \in T_{abs}(n_1)$ and $\bullet t_j = \{s(n_2)\}$ such that $(t_j, n_2) \in R_{t_i}(n_1)$, or*
- *$t_i = e(n_1)$ and $t_j \in T_{abs}(n_2)$ such that $(t_j, n_1) \in R_{t_j}(n_2)$.*

(ii) for each $t_i \prec t_j \prec t_k$ such that $i \in \{0,1,\ldots,m\}$, $j = (i+1)mod(m)$, and $k = (j+1)mod(m)$: $[t_i = e(n_1)$ and $t_j \in T_{abs}(n_2)$ and $\bullet t_k = \{s(n_1)\}] \Longrightarrow (t_j, n_1) \notin R_{t_j}(n_2)$.

$\wp$ is called acyclic iff it is cycles free.

In the definition 18 we formulate the conditions in which the proper termination of plan execution will be *sure*, *possible*, or *impossible*

Definition 18 (*Sure, Possible and Impossible for Proper termination*).
Let (Tr, Cxt, Ms) be an execution state of a plan $\wp$. The proper termination of the execution (or simply the execution termination) of $\wp$ is:
- *Sure, iff $\wp$ is acyclic and $\forall(rs, q) \in Res(t_0).cons, qt_{Cxt}(rs) \geq q^+$,*
- *Impossible, iff is $\wp$ is cyclic or $\exists(rs, q) \in Res(t_0).cons, qt_{Cxt}(rs) < q^-$,*
- *Possible, iff $\wp$ is acyclic and $\forall(rs, q) \in Res(t_0).cons, qt_{Cxt}(rs) > q^-$ and $\exists(rs, q) \in Res(t_0).cons, qt_{Cxt}(rs) \in [q^-, q^+[$.*

A plan whose execution is impossible is a plan that has no way to ensure the success of plan execution. A plan whose the safe execution is sure is a flexible plan. It can be executed correctly regardless of the choice to be taken to refine abstract transitions and the execution order of concurrent tasks. Between these two cases, the success of execution may be possible in an uncertainty case. To address this incertitude, the plan must be reorganized by updating the synchronization net (block some refinement rules and/or add some constraints on the execution of tasks).

Soundness of SHPlNet

The consideration of concurrent tasks synchronization leads to the redefinition of the conditions under which a hierarchical plan is sound. Establishing the temporal order relationships and exchange of resources between parallel tasks can lead to reduced consumption of resources, which can lead therefore to obtain a sound plan. However, it leads to deadlock in cyclic plans.

The verification of soundness property of a SHPlNet is only limited to the verification of quasi-liveness property of all transitions and proper termination criterion, because the property relative to the absence of multiple firing of steps is inherited directly from HPlNet model.

Lemma 4. A marked SHPlNet $(\wp, (Tr_0, Cxt_0, Ms_0))$ does not contain steps that can be fired more than once.

Proof. By contradiction, we assume that there is (at least) a step that can be fired more than once. Let sp be step such that $(Tr_i, Cxt_i, Ms_i) \xrightarrow{sp} (Tr_j, Cxt_j, Ms_j)$ and $(Tr_i, Cxt_i, Ms_i) \in A(\wp, (Tr_0, Cxt_0, Ms_0))$. If sp can be fired again then there is $(Tr_k, Cxt_k, Ms_k) \in A(\wp, (Tr_j, Cxt_j, Ms_j))$ such that $(Tr_k, Cxt_k, Ms_k) \xrightarrow{sp}$. By definition $(Tr_i, Cxt_i, Ms_i) \xrightarrow{sp} (Tr_j, Cxt_j, Ms_j)$ implies $Tr_i \xrightarrow{sp} Tr_j$, and if $(Tr_j, Cxt_j, Ms_j) \xrightarrow{*} (Tr_k, Cxt_k, Ms_k)$ which implies $Tr_j \xrightarrow{*} Tr_k$ than $Tr_k \nrightarrow^{sp}$, and therefore $(Tr_k, Cxt_k, Ms_k) \nrightarrow^{sp}$. Thus, sp cannot be fired more than once. ■

Pursuant to the lemma 4, we can decide that SHPlNet is sound if it is quasi-live and proper termination criterion is verified. In fact, there is a dependency between these two criteria. Each termination of SHPlNet, that is not proper, is termination when there are steps (exactly transitions) which cannot be fired, i.e. blocking.

Lemma 5. The proper termination criterion of marked SHPlNet $(\wp, (Tr_0, Cxt_0, Ms_0))$ is verified if it is quasi-live.

Proof. To demonstrate that the proper termination criterion of a SHPlNet is verified if it is quasi-live, we assume that the proper termination criterion of a SHPlNet is not verified and demonstrate that it is not quasi-live. We assume that the termination is not proper. Then, there exists a sink state (Tr_p, Cxt_p, Ms_p) reachable from the initial state such that $Tr_p \neq \perp$. By projecting SHPlNet on HTPN and pursuant to the theorem 1, if $Tr_p \neq \perp$ then it must be at least one firable sequence σ (including steps that are not yet fired) such that $Tr_p \xrightarrow{\sigma} Tr_f$. Thus, if (Tr_p, Cxt_p, Ms_p) is sink state then the first step in σ must be a transition t which is enabled vis-a-vis Tr, but it is not vis-a-vis (Cxt_p, Ms_p) (or $((Cxt_p, Ms_p) \nrightarrow^{t})$; thus $(Tr_p, Cxt_p, Ms_p) \nrightarrow^{t}$ and then t is blocking transition. In conclusion, SHPlNet is not quasi-Live. ■

Theorem 4. A marked SHPlNet $(\wp, (Tr_0, Cxt_0, Ms_0))$ is sound if it is quasi-live.

Proof. Pursuant to the lemma 4, there is no marked SHPlNet $(\wp, (Tr_0, Cxt_0, Ms_0))$ that can contain steps to be fired more than once. On the other hand, pursuant to the Lemma 5, proper termination criterion is verified if a marked SHPlNet is quasi-live. Therefore, the marked SHPlNet $(\wp, (Tr_0, Cxt_0, Ms_0))$ is sound if it is quasi-live. ■

In the previous section, we showed how to verify the soundness of a plan represented by HPlNet by analyzing only the summary information associated with the task of highest abstraction level. The condition used for this is not sufficient for the case of SHPlNet due to the possible occurrence of the cycles causing deadlock situations. Therefore, the absence of cycles in a plan represented by SHPlNet is a necessary condition for it to be sound.

5. CONCLUSION

The work presented in this paper complement our previous works about the representation of hierarchical plans with synchronization. We presented here a theoretical framework for the verification and validation of soundness and invalidity properties of partial hierarchical plans represented by SHPlNet. We are focused on the demonstration of some key properties that allow to verify the soundness and invalidity of plans by only analyzing the summary information associated to the task of highest abstraction level.

Future work will focus on the analysis of the computational complexity. We will show how the summary information based analysis can reduce the computational complexity compared to the reachability graph analysis.

6. REFERENCES

[1] Aamodt, Agnar, and Enric Plaza. "Case-based reasoning: Foundational issues, methodological variations, and system approaches". AI communications 7.1 (1994): 39-59.

[2] C. Linqin, M. Tao , S.Yining, S. Lei, M. Zuchang. "Modeling and analyzing multi-agent task plans for intelligent virtual training system using Petri nets". Sixth World Congress on Intelligent Control and Automation, 2006. The. Vol. 1. IEEE, 2006.

[3] B. J. Clement, E. H. Durfee, and A. C. Barrett. "Abstract reasoning for planning and coordination". JOURNAL OF AI RESEARCH, 28 :453–515, 2007.

[4] K. Erol. Hierarchical Task Network Planning. Formalization, Analysis, and Implementation. PhD thesis, College Park, MD, USA, UMI Order No. GAX96-22054, 1996.

[5] Fallah-Seghrouchni, A. E., Irene Degirmenciyan-Cartault, and Frédéric Marc. "Modelling, control and validation of multi-agent plans in dynamic context."Autonomous Agents and Multiagent Systems, 2004. Proceedings of the Third International Joint Conference on. IEEE, 2004.

[6] Haddad, Serge, and Denis Poitrenaud. "Recursive petri nets." Acta Informatica44.7-8: 463-508, 2007.

[7] M. E. Pollack. "The uses of plans". Artificial Intelligence, 57(1) :43–68, 1992.

[8] Seghrouchni, A. El Fallah, and Serge Haddad. "A recursive model for distributed planning." Proceedings of the 2nd International Conference on Multi-Agent Systems, 1996.

[9] Christensen, Søren, and Laure Petrucci. "Modular analysis of Petri nets." The computer journal 43.3: 224-242. 2000.

[10] S.Brahimi, R. Maamri, & Z. Sahnoun. "Partially Centralized Hierarchical Plans Merging". In Recent Developments in Computational Collective Intelligence (pp. 59-68). Springer International Publishing, 2014.

[11] S.Brahimi, R. Maamri, & Z. Sahnoun. "Hierarchical Multi-Agent Plans Using Model-Based Petri Net". International Journal of Agent Technologies and Systems. (IJATS), 5(2): 1-30. 2013.

[12] Ziparo, Vittorio Amos, and Luca Iocchi. "Petri net plans." Proceedings of Fourth International Workshop on Modelling of Objects, Components, and Agents. 2006.

[13] Shaw, P. H., & Bordini, R. H. "Towards alternative approaches to reasoning about goals". In Declarative Agent Languages and Technologies V(pp. 104-121). Springer Berlin Heidelberg. 2008.