

Cooperative Team Plan: Planning, Execution and Replanning

Olivier Bonnet-Torrès^{*,†} and Catherine Tessier[†]

^{*}ENSAE-Supaero and [†]ONERA-CERT/DCSD

2, avenue Édouard Belin 31055 Toulouse cedex 4 FRANCE

olivier.bonnet@onera.fr, catherine.tessier@onera.fr

Abstract

In the context of robot team control, this paper focuses on a framework for designing a team plan and its projection onto individual robotic agents. A mission plan, represented as a coloured Petri net, is calculated through constraint optimisation from goal and time requirements. The mission plan is then turned into a hierarchical team plan through reduction rules that also structure the dynamic hierarchical team organisation. Hence each level in the team plan corresponds to an abstract plan at the corresponding subteam level in the team hierarchy. Controlling an agent individually requires extracting individual information, such as activities involving the agent as well as interacting agents or subteams at each subteam level: the team plan is projected onto individual agents. At runtime events may disrupt plan execution. A reaction is executed while diagnosis triggers replanning which is performed as locally as possible.

Introduction

In the agent world activity planning has been widely studied. The increasing complexity of the jobs assigned to agents, especially robotic agents, has led to using groups of agents. The groups, when organised and aware of their organisation, are called teams. The problem of team planning is considered difficult (state-space size of $(2^m - 1)^k k! \prod_{j=1}^k u_j$, with m the number of agents, k the number of goals, u_j the number of recipes for the j th goal).

The general framework is a mission specified in terms of objectives: physical robotic agents are operated in order to carry out the objectives and they are hierarchically organised in a team. As outlined later on, most architectures currently proposed either do not take advantage of the agents being designed to operate as a team or restrict the agents to use reactive behaviours. Replanning itself is often considered as a separate problem. This paper aims at emphasizing the relationship between the team plan and individual agents' plans and formalising and integrating the replanning process through the use of Petri nets (PN — see Appendix) (Bonnet-Torrès & Tessier 2005a; 2005b).

In the first section the mission plan and team organisation are derived from initial problem data. Then the plan is broken down into individual executable plans. The next section

details the execution of the plan and the replanning process. Finally related works are presented.

From Mission Requirements to Mission Plan

Mission Plan Construction

The *objective* of the mission is decomposed into a hierarchy of goals to be carried out. The leaves in the hierarchy are *elementary goals*. Executing a *task* corresponds to satisfying an elementary goal. *Recipes* (Grosz & Kraus 1996) give courses of actions for performing the tasks. Several recipes requiring different sets of services may be available for the same task allowing to achieve it — and performing the associated goal — in a number of fashions.

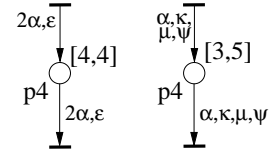


Figure 1: Two recipes for the same task p4, requiring services $\{2\alpha, \epsilon\}$ and $\{\alpha, \kappa, \mu, \psi\}$ resp. and durations 4 and 3 to 5 resp.

During mission preparation the set of recipes is defined by the (human) mission manager. A recipe is a sub-PN consisting in a single place and two transitions (see fig. 1). It specifies what services are needed, how to use them and what the predicted duration (including an allowed latency parameter) for completion is. Recipes may be connected to each other through transitions. The other assumptions of the model are: (1) the elementary goals may come along with a completion time requirement or a mutual temporal precedence specification; (2) the (heterogeneous) agents can be regarded as service aggregates managed by coloured Petri nets (CPNs — see Appendix) (Jensen 1997) (see fig. 2). In the figure each symbol is a token and its graphical type corresponds to a colour. Each colour represents a usage constraint between a number of services (the outer places) within the considered agent. Typical constraints are mutual exclusions of combinations of services and limitations in the number of services concurrently used. Black tokens model the state of each ser-

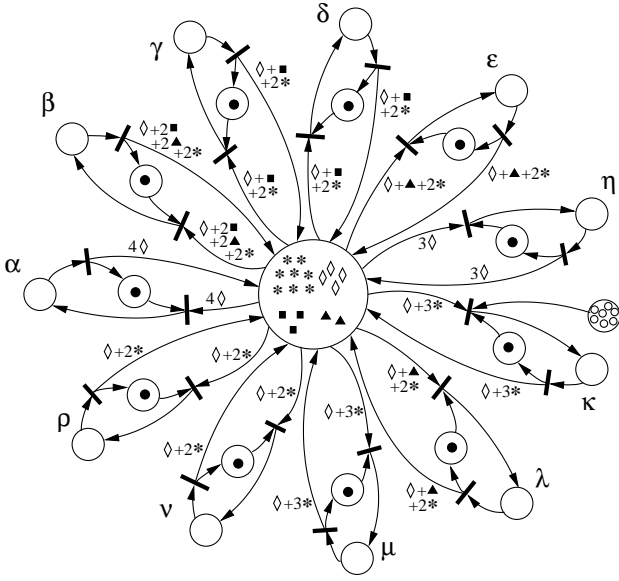


Figure 2: An agent viewed as a collection of services. (Each type of symbol is a colour; greek letters designate the services.)

vice (either free or in use). White tokens represent the remaining amount for consumable services as for κ .

Example 1 The lozenge colour (\diamond) represents the general limitation of four services used at the same time; α prevents the use of any other service and η limits the number of concurrent service use to only one. Practically lozenges represent the battery instant power consumption and service α is “emit marker signal” which consists in sending a powerful geographical reference signal from a fixed location. η is “emit video stream” which heavily draws on battery current and β is “get video stream” which also uses the camera (modelled by triangles \blacktriangle) and co-processor bandwidth (stars $*$).

The triangles (\blacktriangle) symbolise the exclusion in accessing the camera between the use of β and any of ϵ (“grab single image”) and λ (“detect object”). ϵ and λ can be performed on the same image and thus can be used at the same time.

The mission plan is a solution of a constrained optimisation problem: knowing that services may have mutual exclusion constraints within each agent and that several recipes are available to perform each elementary goal, the construction process output is a partial preorder on the goals that respects their constraints colocated with a dynamic agent service allocation. Each goal is performed according to a recipe to which some agents are attached so that both agent-internal constraints (service exclusion) and external constraints (time specifications and service requirements for each recipe) are satisfied. Technically the problem is described in three parts. The first one is the goal tree with time specifications and some global options such as the factors to apply to the different terms in the optimisation function (prevalence of time, goal achievement, or costs). The second part describes for each elementary goal (*i.e.* each task) the different recipes available and their requirements in terms of services or time.

Time specification of a task partly derives from the goals it (totally or partly) achieves and may include a duration, a start time interval, a stop time interval and a maximum latency after finish (the next task must start before the maximum latency in order for the current task to have a valid result). The last part is a list of all agents with their services and internal constraints. The agents are described as collections of services (see fig. 2). The problem is solved thanks to `choco`¹, an open-source constraint optimisation and satisfaction Java library.

Mission Plan and Agenticity Hierarchy

The output of the constraint solver is interpreted as the mission plan but is also a schedule: it corresponds to a detailed sequence of tasks with dynamic service allocation. According to the goals some recipes are chosen to performed tasks and the agents are allocated to the recipes as colour parameters. The recipe time parameters are instantiated to meet time requirements as well as minimise completion time. Then recipes are chained to each other by fusing transitions in order to form the plan, which is represented as a CPN (see fig. 4). The set of token colours is the set of elementary agents. Each reachable marking \mathcal{M} represents the allocation of the agents to the corresponding tasks: a possible organisation of the team is attached to the plan.

Since a group of closely interacting agents can be considered as an agent in itself (Shoham 1993), a subteam of agents is equivalent to a composite agent. This agent bears an *agenticity hierarchy*, whose leaves are elementary agents and whose nodes are subteams, *i.e.* composite agents. Each node has for children nodes the agents that compose the subteam it represents. There is no requirement that an individual agent be represented only once: an agent may belong to several subteams (therefore the hierarchy is not a tree). For instance agent x_1 can use its camera for taking a coordinated stereo picture of an object and at the same time send a message to another subteam reporting the local ground adherence.

More formally the team X is composed of hierarchically organised elementary agents $\{x_1, x_2, \dots, x_n\}$. Let $A = \{a_1, a_2, \dots, a_m\}$ be the set of agents in X . The agenticity of agent a_i with regards to any subteam $a_j, a_i \subset a_j$ (including X) is its depth in the hierarchy \mathcal{H}_{a_j} whose root is the considered subteam (see fig. 3).

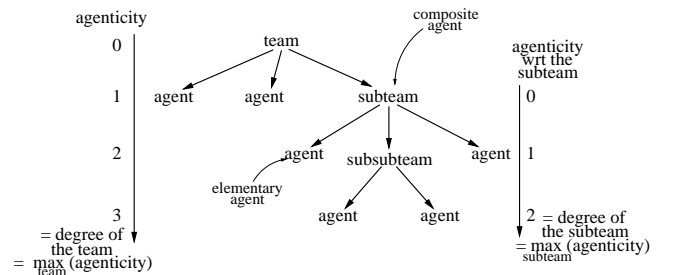


Figure 3: Hierarchy of agenticity

¹<http://choco.sourceforge.net>

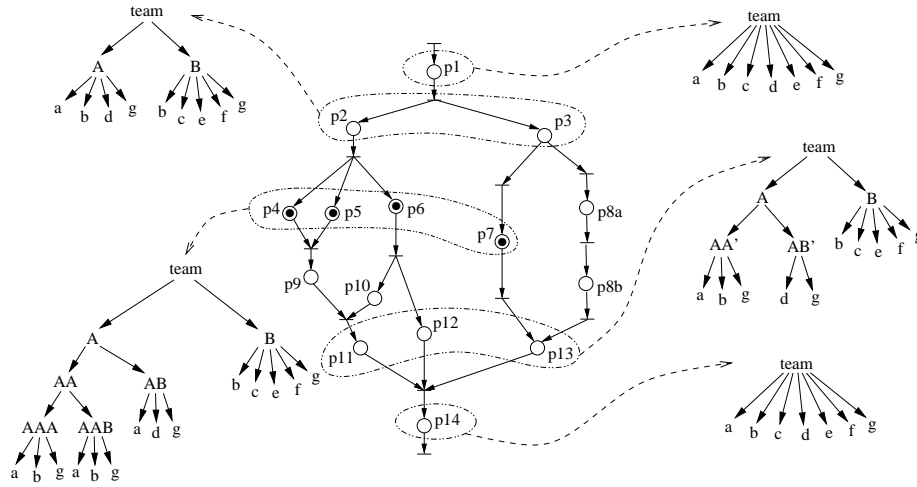


Figure 4: Mission plan with some agency hierarchies derived from the initial allocation

As the marking evolves during plan execution, the organisation of the team changes letting the agency hierarchy be dynamic. Since it reflects the interactions between agents and subteams, the agency hierarchy provides a means to monitor and diagnose team activity and henceforth a hint for determining the parts of the plan to be repaired.

Petri net analysis can be performed through the use of the incidence matrix \mathcal{A} (Murata 1989). \mathcal{A} represents the relations between places and transitions, namely the arcs. The arcs provide a visual verification that no agent can leave a subteam if it has not been planned.

animation and analysis tool². An example of mission plan is provided in figure 5.

From Mission Plan to Individual Plans

Planning is not an end in itself. The ultimate goal is to execute a mission on physical robotic entities. Hence each of these robots needs to have a valid executable plan. Therefore the mission plan has to be adapted to the individuals. The idea is then to project the team plan onto individual agents. Indeed the mission plan bears some typical structures that can be identified as modifications of the team organisation, which give hints for turning the plan into a hierarchy of tasks. The hierarchical plan will help monitoring team activity at any level of details and will show the activity and team breakdowns: indeed diagnosis and repair will be confined in and only in the failing parts of the plan, which is worth considering when physical robots are spread out on the field.

Reduction

Representing a team plan as a hierarchical Petri net (HPN) allows for more flexibility than CPN and helps incorporating team-related hierarchical information. The net is therefore adapted so as to represent the activities at each level of agency. To represent the hierarchical information we extend the ordinary Petri net reduction rules according to the semantics of basic team management structures (Bonnet-Torrès & Tessier 2005b), namely *arrival* or *withdrawal* of an agent, *create* or *merge* two subteams, *transfer* agents between subteams and *choose* a possible course of actions. Within the plan these structures combine so as to form some activity patterns as shown on figure 6. For instance splitting a team and merging its subteams again corresponds to the *parallel activities* pattern (fig. 6.c). Each pattern is associated a specific reduction rule that isolates it in a separate

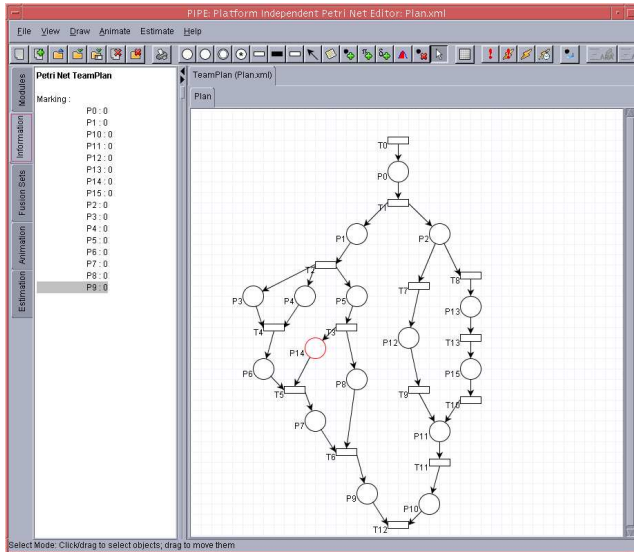


Figure 5: The mission plan in Pipe. The lighter place (p14) corresponds to a transfer

Due to the nature of planning with scheduling the plan is actually represented as a coloured Petri net including time (see Appendix) and implemented with a modified version of Pipe (Bloom *et al.* 2003), an open-source Petri net design,

²<http://pipe.sourceforge.net>, an offspring of which stands at <http://pipe2.sourceforge.net>

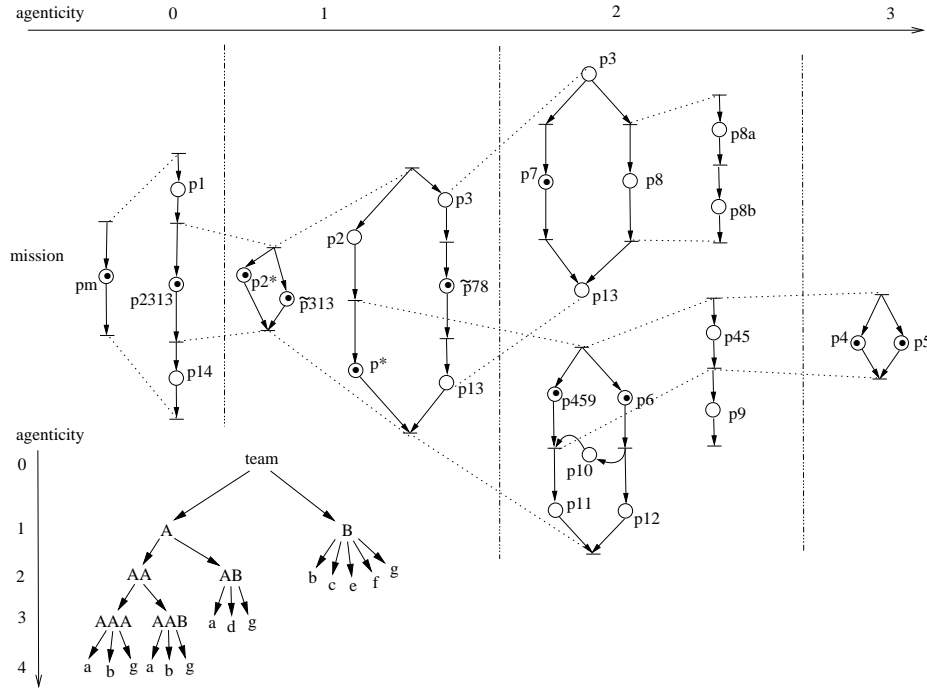


Figure 7: A hierarchical team plan with agenticity hierarchy tokens

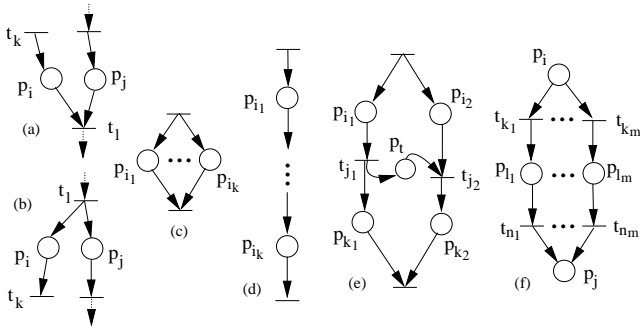


Figure 6: Some activity patterns: (a) arrival, (b) withdrawal, (c) parallel activities, (d) sequential activities, (e) agent transfer, (f) choice

Petri net and abbreviates it as a single place using modular Petri net fusion sets. The reduction rules are iteratively applied until the net is reduced to a single place (Bonnet-Torrès & Tessier 2005a). The algorithm is then traced back so that each reduction place is hierarchically unfolded. The resulting plan then consists in a HPN whose levels correspond to the levels of agenticity in the agent team (see fig. 7). Each place develops into a subnet of higher agenticity whose places hold the agent(s) performing the activity corresponding to the marking. Hence each reachable marking \mathcal{M} corresponds to an agenticity hierarchy $\mathcal{H}_X(\mathcal{M})$ of the whole team X . The Petri net in figure 4 in fact corresponds to the *detailed global plan* built from the leaf-places of the hierarchical team plan of figure 7. The leaf-places are the places

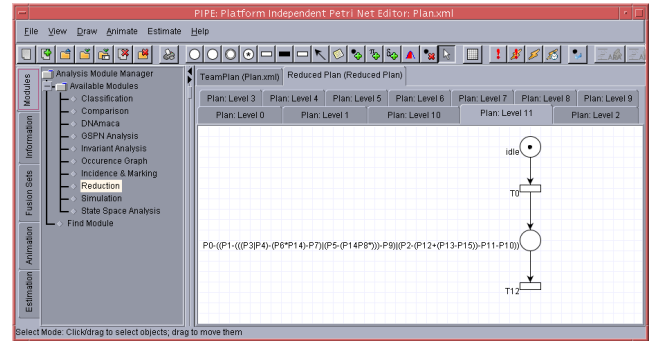


Figure 8: The hierarchical team plan in Pipe

that are not expanded in the hierarchical plan and thus correspond the different tasks.

The hierarchical team plan in figure 7 is calculated from the net in figure 5 by applying the reduction module that we have implemented in Pipe. It results in a set of subnets linked by fusion sets that represent the hierarchical team plan (see fig. 8). The name of reduced places follows an unambiguous algebraic semantics that reflects the sequence of reductions applied.

Projection

The hierarchical structure of the team plan allows the agents' individual plans to be derived. In the mission plan (see fig. 4), the plan of agent a_i consists of the sequence of places where a_i appears and all levels above. The corresponding activities all involve a_i or its ancestors in the different agen-

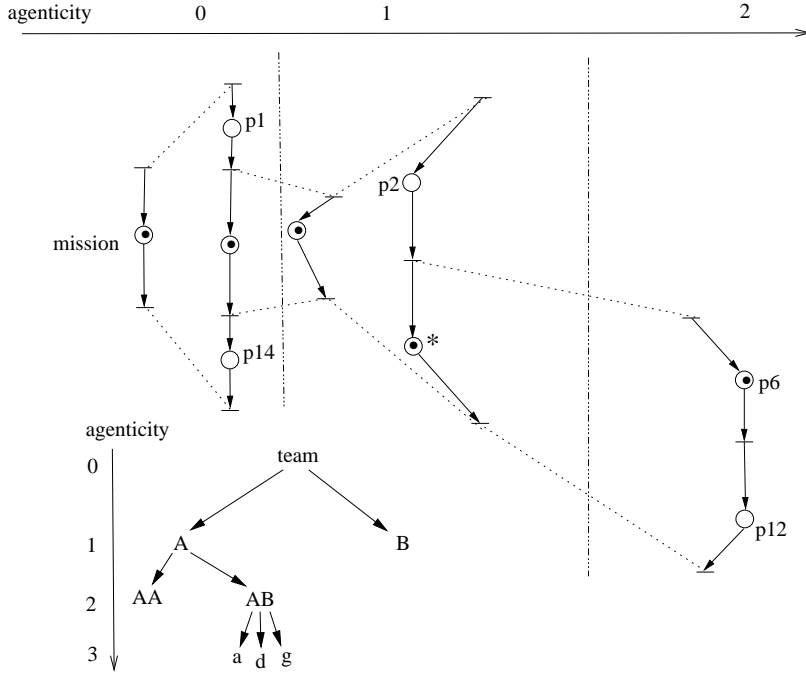


Figure 9: Projection of the team plan on agent d

ticity hierarchies. The projection of the team plan on agent a_i consists in isolating the places of the corresponding level of agenticity in which a_i is involved and extracting the hierarchies of places and of agenticity.

Example 2 Figure 9 shows the agent plan for the elementary agent d. At each level the hierarchical team plan Petri net has been pruned so that the remaining places involve d or its ancestors. One can notice that the same operation can be performed locally for agent AA. Locality is a consequence of the fugacity of AA due to its being a composite agent. Modifying the team organisation according to the activity creates local cooperation groups. For instance in marking $\{p_4, p_5, p_6\}$, a, b and g are collaborating for p_4 and p_5 in resp. AAA and AAB, whereas b is not individually involved with AB: AAA and AAB interface with AB is AA and AB does not need to know the specifics of the activity of each individual agent.

Plan Execution

A General Overview

Another Petri net models execution control (see fig. 10). The controller is the same for all agents at any level of agenticity including the team itself: an abstract instance of the controller is considered during mission design and monitoring and is distributed on the team at mission start-up. The controller is designed and executed with ProCoSA³. ProCoSA is

³Programmation et Contrôle des Systèmes Autonomes (Autonomous System Programming and Command), developed at Onera-DCSD.

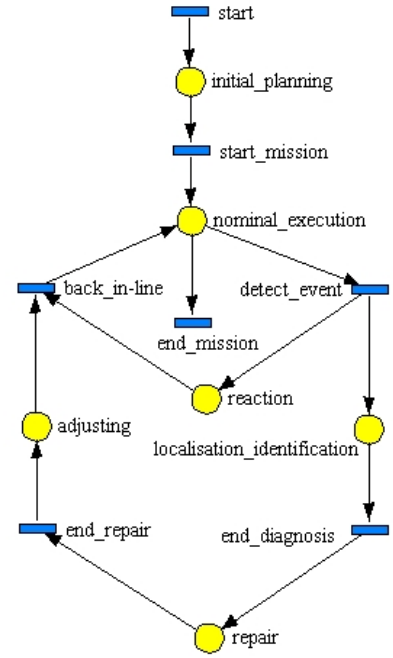


Figure 10: Execution control PN displayed in ProCoSA

a Petri net design, analysis and real-time execution environment running a simplified LISP interpreter. It deals with safe Petri nets (PNs with at most one token per place) or coloured Petri nets. It takes into account external events as transition triggering pre-requisites and can launch the execution of any compiled computational module.

Before the mission begins an initial planning phase (first place in the PN on figure 10) is performed: the plan is prepared out of the set of recipes, then reduced and projected onto individual agents, as mentioned in the previous sections. Then the plan is executed. At the occurrence of a disruptive event a replanning step is triggered. At that time a reaction is performed (e.g. an immediate stop for a ground robot) while the system goes under diagnosis. When the failure is located the plan is repaired as locally as possible. Once the new plan is elaborated an adjusting step is required to ensure a smooth switch between the reaction and normal execution of the new plan. Details are given in the next two sections.

Event Detection, Reaction and Diagnosis

During plan execution a disruptive event may be detected. Events can be categorised as follows:

Predicted events: they are usually predicted from mission preparation as highly probable events in specific situations. They may model the indeterminate outcome of an uncertain action. They are taken care of thanks to alternate branches in the team plan — cf. choice structure.

Identified events: they are identified during mission preparation but are likely to happen at any time. They may

model either mission-critical or highly probable unwanted events. At mission preparation a ready-to-instantiate alternative plan is computed for each of these events.

Unidentified events: they are not distinguishable and are usually diagnosed through time-out. They trigger a generic reaction defined at mission preparation.

For the last two categories a diagnosis step is triggered at the same time as the reaction plan is instantiated and executed. Diagnosis determines which agent(s) is(are) in trouble and which task(s) is(are) impacted, what the current locations of the agents are, *etc.*, according to the features of the situation.

Example 3 *Let us consider a team of robots on an exploration mission. In the plan on figure 4, (case 1) agent d diagnoses a payload failure affecting the use of the IR camera it is using for completing task p6. The reaction of the robot is to stop while replanning. At another time, let p7 be a precision line-in where e is to check the alignment. Suppose (case 2) agent e is late and misses the time window for transition p7→p13 which is externally detected by agent g. The reaction plan consists in all robots in subteam B halting the activity corresponding to p7 but carrying on their other tasks. At that point there is no guarantee e performed task p7 at all.*

Replanning

Replanning is triggered for unidentified events. The event results in some new constraints on the remaining recipes. The plan is repaired by applying the planning process locally. Indeed a subset of the initial problem is input to the *choco* constraint solver. Locality is ensured in trying to solve the problem at the lowest possible level in the agentcity hierarchy. The plan repair consists in replacing the failing recipe by another recipe or subset of recipes that realise the same goal. The subsequent activities may be modified so as to respect the constraints. If this fails it is necessary to involve other parts of the initial plan in ascending the agentcity hierarchy. One can notice that, as far as some subteams are concerned, there exists a discontinuity between the already-executed part of the plan and the new plan at the repaired level, but not at the above levels in the hierarchy.

Example 4 *(case 1) agent d tries to use its EO camera instead of IR vision, since it is another recipe for the same task of taking a picture. However this other recipe accounts for a lesser reward than the original one because of lower image contrast. (case 2) agent B broadcasts to all agents a request for precision theodolite. Answers are treated on a first-come first-serve basis. If no agent can take e's place agent B replaces p7 with p47 that has a higher cost in energy and lasts longer. The subsequent tasks are not impacted because of high latencies.*

When the repaired plan is constituted the current state of the team may not correspond to the initial state of the repaired plan. An adjustment is then necessary. Two possibilities have been identified: either the beginning of the new plan is appended a dispatch phase that will be suited to the expected end point of the reaction, or the reaction is interrupted and the dispatch is organised from that point. In both cases the transition to standard mission execution is smooth.

Example 5 *(case 1) agent d resumes its task and moves to the next task with a reduced latency. (case 2) all robots in B, except for e who may not be operational any more, have to get back to their working locations and start the new recipe from where the failing one stopped.*

Related Work

Replanning is a particular type of planning. In our model objective decomposition is close to that in HTN (Erol, Hendler, & Nau 1994). In the wake of HTN, Grosz *et al.* (Grosz & Kraus 1996) base the *SharedPlan* approach on the hierarchical decomposition of shared plans into a sequence of recipes to be applied by a team of agents. Their work also inherits from the logics of beliefs and intentions (Cohen & Levesque 1990). Tambe *et al.* (Tambe 1997) have focused in STEAM on reactive team behaviour based on rules. Kaminka has pursued the previous work and proposes BITE (Kaminka & Frenkel 2005), a behaviour-based architecture extending Tambe's TEAMCORE. It consists of three components: a team (or organisation) hierarchy, a hierarchically and sequentially decomposed behaviour graph and a set of interaction protocols (described with Petri nets) for behaviour transitions. One of the characteristics of the approach compared to ours is that an agent may only have one behaviour and thus only appears in one branch of the organisation.

A sustained interest in handling durations and time windows also exists in planning (Fox & Long 2005; Gerevini, Saetti, & Serina 2005). Our approach is to use constraint programming to treat both time and service allocation. The current setup makes use of a centralised solver that will allow easier assessment of replanning quality. Yet this choice, which is linked to our goal to assess the relevance of local vs. from-scratch replanning, may be changed for a more elaborate algorithm, such as OptAPO (Mailler & Lesser 2004) or Adopt (Modi *et al.* 2005). Their semi-centralised distributed nature is actually well adapted to our problem (sparse constraints, distribution, fast approximate). However such open issues as *n*-ary constraints also prevent a fast use of both OptAPO and Adopt. Cox and Durfee, while using Adopt as a distributed solver (Cox, Durfee, & Bartold 2005), focus on coordination (Cox & Durfee 2005), with limited common goals and individual planning *a posteriori* adapted to the multiagent setting. In that sense coordination is different from team work where all agents have knowledge of a common objective.

Van der Krogt (van der Krogt & de Weerd 2004) identifies two effects in the repair process: removing actions from the plan and adding actions. However the approach also considers multiagent (re)planning from a coordination point of view (van der Krogt & de Weerd 2005). This characteristic stems from the privacy requirement. The diagnosis in (Witteveen *et al.* 2005) is performed on the values of specific variables in the plan. Though the idea allows prediction of failures and propagation of effects, its limit stands in the number of monitored variables that might not capture all failure conditions. On the contrary, our approach does not depend on variable monitoring – thus cannot provide prediction — and deals with events as they occur by cancelling the affected task(s).

Task allocation often uses constraint solving (Baptiste, Le Pape, & Nuijten 2001). Scerri's approach (Scerri *et al.* 2005) presupposes that all agents can perform all tasks and that an agent is assigned to a unique task thus forbidding collaborative work. The idea of information tokens being passed around is also developed in (Xu *et al.* 2005). In this work the agents are organised in an acquaintance network that in the end does not take advantage of tighter or looser coupling due to joint task performance. In our approach the agentcity hierarchy provides an acquaintance network with degrees of acquaintance: agents may know each other well (meaning they directly interact) or more vaguely (meaning they only interact through the hierarchy). The hierarchy of tasks takes advantage of tight coupling between tasks within a single structure but also describes looser coupling between structures in two different branches of the hierarchy. Plan representation itself tends to make use of the automata theory (El Fallah-Seghrouchni, Degirmenciyan-Cartault, & Marc 2004) and the Petri net formalism (Chanthery, Barbier, & Farges 2004).

Experiments & Conclusion

In the context of teams of robots, the approach described in this paper aims at dynamically responding to disruptive events, such as a failure or an external action, at a relevant level. The extensive use of Petri nets is due to the completeness of the formalism (with clear semantics and extensions for time or net coupling) as well as the possibility to both visually and computationally verify the soundness of the plans. Current works concern the reallocation problem in the repair. An identified difficulty is to avoid global repair that involves the whole team: the repair must be attempted at the lowest agentcity level (recipe level) so as to ensure its locality; if unsuccessful the next level is considered. An assessment of the pertinence of local replanning is under investigation by comparing runtime and quality of local replanning and replanning from scratch. The midterm objective is the completion of ELAIA, a Petri net-based decision architecture for local replanning within the team.

The evaluation will consist in measuring time used for replanning when an obstacle or failure event prevents the mission to be carried out. The comparison is made on local replanning against replanning from scratch. Experiments are currently in progress in order to run the architecture and validate the principles of local replanning with a team of *PeKee* robots at Supaero. The mission consists in two UAVs and two UGVs getting at some points through different ways to take a stereo picture and then reaching together a final way-point. Obstacles may block some routes. The envisioned applications concern the implementation of cooperative robots for missions ranging from search and rescue operations to military UAV/robot team operation.

Acknowledgements

The authors wish to thank W. Knottenbelt (Imperial College, London, UK) and J. Bloom (Sapient, London, UK) for permission to use the name and part of their tool.

References

- Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-based scheduling: applying constraint programming to scheduling problems*. Boston, MA: Kluwer Academic.
- Bloom, J.; Clark, C.; Clifford, C.; Duncan, A.; Khan, H.; and Papantoniou, M. 2003. Platform Independent Petri-net Editor – final report. Technical report, Imperial College, London, UK. *Under the supervision of W. Knottenbelt*.
- Bonnet-Torrès, O., and Tessier, C. 2005a. From multiagent plan to individual agent plans. In *ICAPS'05 Workshop on Multiagent Planning and Scheduling*.
- Bonnet-Torrès, O., and Tessier, C. 2005b. From team plan to individual plans: a Petri net-based approach. In *AAMAS'05*, 797–804.
- Chanthery, E.; Barbier, M.; and Farges, J.-L. 2004. Integration of mission planning and flight scheduling for unmanned aerial vehicles. In *ECAI'04 - Workshop on Planning and Scheduling: Bridging Theory to Practice*.
- Christensen, S., and Petrucci, L. 1992. Towards a modular analysis of coloured Petri nets. In *ATPN'92*, 113–133.
- Cohen, P., and Levesque, H. 1990. Intention is choice with commitment. *Artificial Intelligence* 42:213–261.
- Cox, J., and Durfee, E. 2005. An efficient algorithm for multiagent plan coordination. In *AAMAS'05*, 828–835.
- Cox, J.; Durfee, E.; and Bartold, T. 2005. A distributed framework for solving the multiagent plan coordination problem. In *AAMAS'05*, 821–827.
- David, R., and Alla, H. 2005. *Discrete, continuous and hybrid Petri nets*. Springer-Verlag.
- El Fallah-Seghrouchni, A.; Degirmenciyan-Cartault, I.; and Marc, F. 2004. Modelling, control and validation of multi-agent plans in dynamic context. In *AAMAS'04*, 44–51.
- Erol, K.; Hendler, J.; and Nau, D. 1994. HTN planning: complexity and expressivity. In *AAAI'94*, 1123–1128.
- Fox, M., and Long, D. 2005. Time in planning. In Fisher, M.; Gabbay, D.; and Vila, L., eds., *Handbook of temporal reasoning in Artificial Intelligence*. Amsterdam, The Netherlands: Elsevier. 497–536.
- Gerevini, A.; Saetti, A.; and Serina, I. 2005. Integrating planning and temporal reasoning for domains with durations and time windows. In *IJCAI'05*, 1226–1231.
- Grosz, B., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.
- Jensen, K. 1997. *Coloured Petri nets. Basic concepts, analysis methods and practical use*. MTCS. Springer.
- Kaminka, G., and Frenkel, I. 2005. Flexible teamwork in behavior-based robots. In *AAAI'05 (NCAI'05)*, 108–113.
- Lakos, C. 1995. From coloured Petri nets to object Petri nets. In *ATPN'95*, 278–297.
- Mailler, R., and Lesser, V. 2004. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS'04*, 438–445.

Merlin, P., and Farber, D. 1976. Recoverability of communication protocols: implications of a theoretical study. In *IEEE trans. on Communications*, volume 24-9, 1036–1043.

Modi, P.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161(1-2):149–180.

Murata, T. 1989. Petri nets: properties, analysis and applications. In *Proc. of the IEEE*, volume 77-4, 541–580.

Ramchandani, C. 1973. *Analysis of asynchronous concurrent systems by timed Petri nets*. Ph.D. Dissertation, MIT, Cambridge, MA.

Scerri, P.; Farinelli, A.; Okamoto, S.; and Tambe, M. 2005. Allocating tasks in extreme teams. In *AAMAS'05*, 727–734.

Shoham, Y. 1993. Agent-oriented programming. *Artificial Intelligence* 60:51–92.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.

van der Krogt, R., and de Weerd, M. 2004. The two faces of plan repair. In *Proc. of the 16th Belg.-Neth. Conf. on Art. Int.*, 147–154.

van der Krogt, R., and de Weerd, M. 2005. Self-interested planning agents using plan repair. In *ICAPS'05 workshop on Multiagent Planning & scheduling*, 36–44.

Witteveen, C.; Roos, N.; van der Krogt, R.; and de Weerd, M. 2005. Diagnosis of single and multi-agent plans. In *AAMAS'05*, 805–812.

Xu, Y.; Scerri, P.; Yu, B.; Okamoto, S.; Lewis, M.; and Sycara, K. 2005. An integrated token-based algorithm for scalable coordination. In *AAMAS'05*, 407–414.

Appendix

A Petri Net Reminder

A *Petri net* $\langle P, T, F, B \rangle$ is a bipartite graph with two types of nodes: $P = \{p_1, \dots, p_i, \dots, p_m\}$ is a finite set of places; $T = \{t_1, \dots, t_j, \dots, t_n\}$ is a finite set of transitions (fig. 11(a)) (Murata 1989; David & Alla 2005). Arcs are directed and represent the forward incidence function $F : P \times T \rightarrow \mathbb{N}$ and the backward incidence function $B : P \times T \rightarrow \mathbb{N}$ respectively. An *interpreted Petri net* is such that conditions and events are associated with places and transitions respectively (fig. 11(b)). When the conditions corresponding to some places are satisfied, tokens are assigned to those places and the net is said to be marked. The evolution of tokens within the net follows transition firing rules. Petri nets allow sequencing, parallelism and synchronization to be easily represented.

Several extensions have been proposed to model more complex systems. A *coloured Petri net* (Jensen 1997) is a Petri net whose tokens are assigned simple or complex (vector) colours. Conditions on colours, called guards, may be imposed for firing a transition. A place may accept tokens if their colours are included in the local set of allowed colours. Colour functions may additionally be allocated to the arcs.

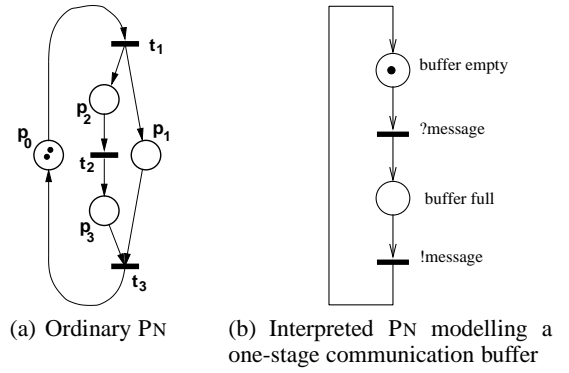


Figure 11: Petri net examples

They modify the colour of the token while it is “transferred” through the arc. The use of colours simplifies otherwise large and sometimes redundant Petri nets. Coloured Petri nets are also used for describing type-dependent behaviours. For instance a flexible manufacturing system may be modelled with a coloured Petri net where places represent the shops and coloured tokens the different types of parts.

Time in Petri nets has been dealt with in two different ways. One — timed Petri nets — has considered that time parameters are limited to specific dates (Ramchandani 1973). The other — time Petri nets — promotes the use of intervals (Merlin & Farber 1976). For both models two (dual) versions exist, time parameters being placed either on transitions (t -time(d)) or on places (p -time(d)). Conditions and external events may also respectively guard or trigger transitions.

Another extension is the possibility to link several places or transitions in different Petri nets. For instance the execution of a plan is conditioned by the availability of the services in the agents. Therefore the agent service Petri nets are linked to the plan by using place or transition fusion sets in modular Petri nets (MPN) (Christensen & Petrucci 1992).

For more details on these extensions, refer to (David & Alla 2005; Lakos 1995).