A Framework for Multi-Agent Planning

André Bos [*] Hans Tonino Mathijs de Weerdt[†] Cees Witteveen

Delft University of Technology
{email: A.Bos, J.F.M.Tonino, M.M.deWeerdt, C.Witteveen}@its.tudelft.nl

Abstract

We introduce a computational framework, consisting of *resources*, *skills*, *goals* and *services* to represent the plans of individual agents and to develop models and algorithms for cooperation processes between a collection of agents.

Keywords: Teamwork and cooperation, multiagent planning, distributed resource allocation.

1 Introduction

Cooperative planning is an important topic in multi-agent systems. Usually, the starting point for research on this subject is the observation that there exist problems that cannot be solved by a single agent in isolation, but require several agents to work together, coordinating their plans and sharing their resources and goals.
Although this observation points out a necessary reason to cooperate, another, almost equally important reason should not be overlooked: Even if agents may be able to solve problems on their own, they might prefer to cooperate if sharing their resources and coordinating their activities just may save costs. This observation refers to applications such as, e.g. *supply chain management* in which individual planning and cooperation phases can be clearly distinguished. Here, a chain manager first assigns each agent to a part of a complex task via a *task allocation process*; next, the agents have to create their plans to complete their part of the task; finally, the chain manager, responsible for the quality of the overall planning, analyses these plans and may insist on cooperation between some agents in order to satisfy (additional) constraints on costs or performance. In such cases, cooperation can be accomplished by plan revision: to reduce the total cost of his plan an agent tries to revise part of his plan by exchanging resources and goals with other agents.
In the literature, some attention already has been given to the development and analysis of efficient (approximately optimal) task allocation methods. We did not encounter, however, computationally feasible approaches for solving the plan cooperation problem. Instead, often a coalition of agents is saddled with the computationally very hard problem of constructing a joint plan to perform a complex task from scratch.

[*]Supported by research program Freight Transport Automation and Multimodality (FTAM).

[†]Supported by research program Seamless Multimodal Mobility (SMM). Both programs are carried out within the TRAIL research school for Transport, Infrastructure and Logistics.

In this paper, we take a special approach to multi-agent cooperative planning. First, we will introduce a new framework for representing plans and resources. The main features of this framework are *(i)* the ability to treat plans as first-class citizens, *(ii)* the distinction between resources, goals and side-products of resource production processes and *(iii)* the idea to view plan cooperation as an iterated plan revision process.

2 The planning framework

Three primitive notions play a central role in our framework: *(i)* elementary production processes, in our framework called *skills* of an agent, *(ii)* the *resources* needed to put a skill into action, *(iii)* and the *goals* to be realized.
On top of these notions, we will introduce *services* as partially ordered sets of skills to model more complex (generic) production processes. Next, we will introduce *plans* as instances of services, that can be used to transform a given set of resources into another set of resources satisfying a given set of goals. The connection between plans and services is important: whenever we try to revise a part of the plan, we will have to check whether the revised plan still satisfies the constraints that are imposed by the service.
To discuss our framework into some detail, first we will introduce a many-sorted (resource) language to describe primitive notions like resources, resource schemes, goals and goal schemes. Then we describe skills and services. Much of our concepts and notational conventions have been taken from logic programming and are easy to grasp.

Language We use a many-sorted first-order language $\mathcal{L}$ containing

- Sorts: $S = \{s_1, s_2, \ldots s_d\}$;
- Variables: $V_s = \{x_s^1, \ldots, x_s^k, \ldots,\}$ for each $s \in S$; we will use $Var = \bigcup_{s\in S} V_s$ to denote the set of all variables.
- Predicate symbols: $Pred = \{p, r, \ldots,\}$ with rank function $r : Pred \to S^*$;
- Function symbols: $Fun_s = \{f_0, f_1, \ldots,\}$ and rank function $r : Fun_s \to S^+ \times \{s\}$ for each $s \in S$;
- Constants: $Con_s = \{c_s^1, \ldots c_s^n, \ldots,\}$ for each $s \in S$.

Equality is dealt with as usual; the set $Term = \bigcup_{s\in S} Term(s)$ of many-sorted *terms* and *Form* of formulae are defined in the usual inductive way, where we distinguish *ground terms* and *ground formulae* from general terms and formulae that may contain variables. If x is a variable of sort s, we will use $x : s$ to indicate its sort.

Resources and resource schemes In our language $\mathcal{L}$, a *resource fact*, or simply *resource*, is an *$\mathcal{L}$-atomic* ground formula $p(t_1 : s_1, t_2 : s_2, \ldots, t_n : s_n)$. As an example, take a resource fact

$$taxi(1234 : id, ams : loc, 100 : time, 3 : cap)$$

referring to the taxi with identification 1234 at location Amsterdam at time 100 with a capacity of 3 free seats.

Remark. It is essential to postulate that every resource (fact) is *unique* and can only be used once. Uniqueness can be guaranteed by several means. For example we could treat a resource $p(t_1 : s_1, t_2 : s_2, \ldots, t_n : s_n)$ as a database tuple with a uniquely identifying key, occurring in some permanent store. Although in our planning world, once a resources is consumed it disappears, every resource is preserved in the store and once a resource is produced it is added to the store. The uniqueness postulate then states that only resources can be created that not already do occur in the store. ∎

Often, we don't need to point out a specific individual resource: it is sufficient if the resource we want, does have certain specific properties. That is, often, we are looking for an arbitrary resource belonging to some *class* of resource facts instead of a specific unique resource. To denote such a class, we use a *resource scheme*; a resource scheme simply is an $\mathcal{L}$-atomic formula (not necessarily ground). Resource schemes are useful to denote arbitrary resources that are ground instances of a resource scheme.
For example,

$$taxi(x : id, ams : loc, t : time, 3 : cap)$$

is a resource scheme denoting a taxi at Amsterdam with three free seats, where we don't care about the time t it is available.
Given some resource r and a resource scheme rs of course we would like to know whether r indeed is a ground instance of rs. Hence, we introduce *substitutions*. A substitution θ is a finite set of pairs of the form $x_i = t_i$ with $x_i \in Var$ and $t_i \in Term$ satisfying the standard conditions[1]. We say that a resource r is an *instance* of the scheme rs if there exists a substitution θ such that $rs\theta = r$. If we have a set RS of resource schemes, we would also like to know if a set R of resources *satisfies* RS, i.e., whether R is a typical set of resources denoted by RS. Note that for each $rs \in RS$ we have to find a unique resource $r \in R$ as an instance of rs. So an individual resource cannot be used to satisfy more than one resource scheme. Therefore, we do not allow a substitution θ such that for two different schemes $rs, rs' \in RS$ we have $rs\theta = rs'\theta$. Hence, we need to introduce *resource-identity preserving* substitutions.

Definition 2.1 *Let RS be a set of resource schemes. A substitution* θ *is said to be* resource-identity preserving *w.r.t. RS, if for every* $r_1, r_2 \in RS$, $r_1 \neq r_2$ *implies* $r_1\theta \neq r_2\theta$.

Since for every substitution θ we have $|RS\theta| \leq |RS|$, a substitution θ is resource-identity preserving w.r.t. RS iff $|RS| = |RS\theta|$.

Definition 2.2 *We say that a set of resources R satisfies a set RS of resource schemes, denoted by* $R \models RS$*, if there exists a resource-identity preserving substitution* θ *w.r.t. RS, such that* $RS\theta \subseteq R$.

[1]That is: *(i)* variables and terms match, i.e., if $x_i \in V_s$ then t_i is a term of sort s; *(ii)* $i \neq j$ implies $x_i \not\equiv x_j$; *(iii)* x_i does not occur as a variable in t_i.

Remark. This definition might appear to be too simple: for example, it might be that *RS* contains resource schemes *rs* with terms that have to be evaluated after a substitution θ in order to show that $rs\theta = r$, for some $r \in R$. We will simply assume that whenever this evaluation process is needed, it is the result of such a substitution process. So in fact $rs\theta = r$ does not indicate pure syntactical equivalence. ■

Sometimes, we need to find a collection of resources that not individually, but together satisfy some conditions. Corresponding classes of resources can be obtained by introducing *extended resource schemes*: an extended resource scheme is a tuple (RS, C) where *(i)* RS is a *set* of resource schemes and *(ii)* C is a set of constraints[2] for terms with variables occurring in basic resources in *RS*.
For example, consider two resource schemes $pass(x : loc, t_1 : time)$: a passenger at some location x and at some time t_1, waiting for a (taxi)-ride $ride(x : from, y : to, c : cap, t_2 : time)$ from x to some place y, having c free seats and starting at time t_2. To help the passenger it would be nice if $c > 0$ and $t_1 \leq t_2$. This is expressed by the following extended resource scheme:

$$(\{pass(x : loc, t_1 : time), ride(x : from, y : to, c : cap, t_2 : time)\}, \{t_1 \leq t_2, c > 0\})$$

Every set R of resources containing at least one ride and one passenger resource meeting these constraints will satisfy this extended resource scheme.

Goals Some resources we have, other resources we want to obtain. In general, we do not care to obtain a specific (unique) resource, but only a resource having some specific properties. So we will conceive an individual goal g as a resource scheme. Even more general, usually we want to obtain a set of goals, meeting some constraints. Therefore, we define *goal schemes*: if $G = \{g_1, g_2, \ldots, g_m\}$ is a set of goals and C a set of constraints for terms occurring in G, $GS = (G, C)$ is a *goal scheme*.
Of course, we would like to know whether a given set *RS* of resource schemes satisfies a given goal scheme *GS*. Here, again, we use resource-identity preserving substitutions.

Definition 2.3 *We say that a set RS of resource schemes satisfies a goal scheme* $GS = (G, C)$*, abbreviated as* $RS \models GS$*, if there exists some resource-identity preserving substitution* θ *w.r.t. G such that*

1. $G\theta \subseteq RS$*, and*
2. $C\theta$ *is a satisfiable set of formulae.*

Skills Suppose we are given some set of resources R and we want to obtain some set of resources R' satisfying a given goal scheme *GS*. We clearly need a way to transform the set R into another set R'. To this end, we introduce *skills* as (elementary) resource consuming and resource production processes: a skill is represented as a rule of the form $s : RS' \leftarrow (RS, C)$. Here, s is the name of the skill and its intuitive meaning is as follows: whenever there exists a set of resources R satisfying the extended resource

[2]We do not feel the need to specify exactly the type of constraints we will use; it suffices to use standard relations.

scheme (RS, C), these resources can be consumed by the skill s and a new set of resources R' satisfying RS' will be produced. To prevent reuse of resources, it is always required that the set of output resource schemes $out(s) = RS'$ is disjunct from the set $in(s) = RS$ of input resource schemes. We will also identify the constraint C occurring in a skill s by C_s.

Definition 2.4 *[skill application to resources]*
Let $s : RS' \leftarrow (RS, C)$ and R, R' be sets of resources. We say that R' can be produced from *R* using *s, abbreviated by $R \vdash_s R'$ if there is a resource-identity preserving substitution θ w.r.t. $RS \cup RS'$ such that*

1. *$RS\theta \subseteq R$, i.e., R contains sufficient resources to activate s,*
2. *$\models C\theta$, i.e., the constraints reduce to true under θ and*
3. *$R' = (R - RS\theta) \cup RS'\theta$, i.e., the resources consumed are removed from R while the resources produced by s are added.*

By the uniqueness postulate, we have to ensure that skills are not able to reproduce resources. Note that the definition above ensures that a single skill application satisfies this postulate:

Proposition 2.5 *Whenever $R \vdash_s R'$ for some skill $s : RS' \leftarrow (RS, C)$ using a resource-identity preserving substitution θ, it follows that $RS\theta \cap RS'\theta = \emptyset$.*

PROOF Since θ is resource preserving w.r.t. $RS \cup RS'$, it follows that $|RS\theta \cup RS'\theta| = |RS \cup RS'|$. Hence, since $RS \cap RS' = \emptyset$, $RS\theta \cap RS'\theta = \emptyset$. ∎

Example 2.6 *To give two concrete examples, take the following skills:*

$$\textbf{drive} : taxi(y : loc, t_1 + t(x,y) : time), ride(x : from, y : to, 2 : cap, t_1 : time) \leftarrow \quad taxi(x : loc, t_1 : time), \\ \{\ \}$$

$$\textbf{travel} : pass(y : loc, t_2 + t(x,y) : time) \leftarrow \quad pass(x : loc, t_1 : time), ride(x : from, y : to, c : cap, t_2 : time), \\ \{t_1 \leq t_2, c \geq 1\}$$

To explain the last skill, **travel***: for a passenger to travel from location x to location y, two resources are needed. First of all, the passenger should be at location x at a certain time t_1 and secondly, a ride (produced by a taxi) should be available at a time $t_2 \geq t_1$ and with capacity at least 1. The time the passenger arrives at location y is calculated using a pre-defined time-distance matrix $t(x,y)$.*
A ride resource can be produced by the first skill: Take the resource set $\{taxi(ams, 10)\}$ suppose that $t(adam, rdam) = 5$ and let $\theta = \{x = ams, y = rdam, t = 10\}$. Then $\{taxi(ams, 10)\} \vdash_{drive} \{taxi(rdam, 15), ride(adam, rdam, 2, 10)\}$.

Application of a skill $s : RS' \leftarrow (RS, C)$ to a set of resources can be *lifted* to the application of s to an extended resource scheme (RS_1, C_1). This enables us to consider *generic* skill applications that are able to transform a given extended resource scheme into another resource scheme[3].

[3]Since a set RS of resource schemes is equivalent to the extended resource scheme $(RS, \{true\})$, application of a skill to a set of resource schemes has to be considered as a special case.

Definition 2.7 *[skill application to extended resource schemes] Let $s : RS' \leftarrow (RS,C)$ be a skill and (RS_1,C_1) an extended resource scheme. We say that (RS'_1,C'_1) can be produced from (RS_1,C_1) using s, abbreviated by $(RS_1,C_1) \vdash_s (RS'_1,C'_1)$ if there is a resource-identity preserving substitution θ w.r.t. $RS \cup RS'$ such that*

1. *$RS\theta \subseteq RS_1$;*
2. *$C_1 \models C\theta$;*
3. *$RS'_1 = (RS - RS\theta) \cup RS'\theta$ and*
4. *$C'_1 = \{c \in C_1 | var(c) \subseteq var(RS'_1)\}$*

Remark. To see that Definition 2.4 is a special case of Definition 2.7, suppose that $R \vdash_s R'$ for a skill $s : RS' \leftarrow (RS,C)$ and using a substitution θ. Note that a set R of resources is equivalent to $(R, \{true\})$. We verify that $(R, \{true\}) \vdash_s (R', \{true\})$: firstly, $RS\theta \subset R$ holds; hence θ is ground; next, $\models C\theta$ is equivalent to $\{true\} \models C\theta$; furthermore $R' = (R - RS\theta) \cup RS'\theta$ and finally, $\{true\} = \{c \in \{true\} | var(c) \subseteq var(R')\} = \{true\}$. ∎

Skills represent generic production processes that may be applied to different sets of resources. Since the same skill may be used more than once, we should be able to distinguish several uses of the same skill; therefore we introduce *skill variants*:

Definition 2.8 (skill variants) *Two skills $s_1 : RS'_1 \leftarrow (RS_1,C_1)$ and $s_2 : RS'_2 \leftarrow (RS_2,C_2)$ are said to be variants if there exist substitutions θ and ψ such that*

1. *$RS'_1\theta = RS'_2$, $RS_1\theta = RS_2$, $C_1\theta = C_2$,*
2. *$RS'_2\psi = RS'_1$, $RS_2\psi = RS_1$, $C_2\psi = C_1$ and*
3. *$var(s_1) \cap var(s_2) = \emptyset$.*

It is not difficult to see that whenever two skills s_1 and s_2 are variants, the substitutions θ and ψ involved have to be resource-identity preserving w.r.t. $RS_1 \cup RS'_1$ and $RS_2 \cup RS'_2$, respectively[4].

Services Given a set of resources R we may apply skills to it to produce another set of resources R'. We want to study *compositions* of applications of skills that can be used to transform a set of input resources into a set of output resources. *Services* are skills organised in a partially ordered structure, indicating which skills are dependent on others to produce certain resources.
Whenever we have a collection S of skills, by a suitable renaming of variables, we can always assume that their associated sets of variables are disjunct.
So let S be a set of skills (including variants of the same skill). $In(S)$ denotes the union of the input resource schemes $in(s)$ for every $s \in S$ and $Out(S)$ is defined analogously. Note that since $in(s) \cap out(s) = \emptyset$ for every $s \in S$ and for every $s, s' \in S$, $var(s) \cap var(s') = \emptyset$, we have $In(S) \cap Out(S) = \emptyset$.

[4]It suffices to observe that $RS'_1\theta = RS'_2$ implies $|RS'_1| \geq |RS'_2|$ and $RS'_2\psi = RS'_1$ implies $|RS'_2| \geq |RS'_1|$. Hence, $|RS'_1| = |RS'_2| = RS'_1\theta$, implying that θ is resource-identity preserving w.r.t. RS'_1. Analogously for RS_1, RS'_2 and RS_2.

First we will address the *concurrent* application of a set S of skills to a given extended resource scheme (RS,C).

Definition 2.9 *Let (RS,C) and (RS',C') be extended resource schemes. We say that (RS',C') can be obtained form (RS,C) by concurrent execution of the skills in S, denoted by $(RS,C) \vdash_S (RS',C')$, iff there exists some resource-identity preserving substitution θ w.r.t. $In(S), Out(S)$ and RS such that*

1. $In(S)\theta \subseteq RS$,
2. $RS' = (RS - In(S)\theta) \cup Out(S)\theta$,
3. $C \models \bigcup_{s\in S} C_s\theta$ *and*
4. $C' = \{c \in C \mid var(c) \subseteq var(RS')\}$.

Note that since θ is resource-identity preserving w.r.t. $In(S)$ and $Out(S)$ and $In(S) \cap Out(S) = \emptyset$, we also have $In(S)\theta \cap Out(S) = \emptyset$. Hence the collection S of skills behaves as a complex skill under concurrent applications.
A *service Ss* then is a poset $(S,<)$ of skills. If for two skills $s, s' \in S$, we have $s < s'$, this means that s' can only be executed if s has been executed before. If s and s' are not related by $<$ then they can be executed simultaneously. Hence, we can define the execution of a service on a set of resources as the concurrent execution of the minimal elements ($min(Ss)$) of the skills in Ss followed by the execution of the service $Ss' = Ss - min(Ss)$:

Definition 2.10 *Ss produces (RS',C') from (RS,C), abbreviated $(RS,C) \vdash_{Ss} (RS',C')$ if*

1. *$Ss = \emptyset$ and $(RS',C') = (RS,C)$ or*
2. *$Ss \neq \emptyset$ and there exists an extended resource set (RS_1,C_1) such that*
   (a) $(RS,C) \vdash_{min(Ss)} (RS_1,C_1)$
   (b) $(RS_1,C_1) \vdash^*_{Ss-min(Ss)} (RS',C')$.

Example 2.11 *Consider the drive and travel skills discussed before,
let $Ss = \{$**drive** $<$ **travel**$\}$ and let*

$$(RS,C) = (\{taxi(x : loc, t : time), pass(y : loc, t' : time)\}, \{x = y, t > 100, t' < 100\})$$

*be an extended resource scheme. Since $min(Ss) = \{$**drive**$\}$ the following resource scheme (RS_1,C_1), where*

$$RS_1 = \{taxi(u : loc, t + t(x,u) : time), ride(x : from, y : to, 2 : cap), pass(y : loc, t' : time)\}$$

$$C_1 = \{x = y, t > 100, t' < 100,\}$$

can be derived from $min(Ss)$. Hereafter, **travel** *can be applied and we verify that the following extended resource scheme can be obtained:*

$$(\{taxi(u : loc, t + t(x,u) : time), pass(u : loc, t + t(x,u) : time)\}, \{t > 100\})$$

Hence,

$$(RS,C) \vdash_{Ss} (\{taxi(u : loc, t + t(x,u) : time), pass(u : loc, t + t(x,u) : time)\}, \{t > 100\})$$

Plans Given some set of input resources R and a goal scheme $GS = (G,C)$, we would like to have a *plan* P for transforming R into a set of resources R' satisfying GS. Such a plan would tells us exactly how the transformation process is performed. In our framework, plans are simply ground instances of services.
Note that a service Ss, like a skill, can be applied to a given set of resources R to produce a set of resources R'. During the application of the skills to the resources, ground instances of these skills are obtained by resource-identity preserving ground substitutions θ. Hence, we define a plan P for GS using Ss as follows:

Definition 2.12 *Given a set of resources R, a goal scheme $GS = (G,C)$ and a service Ss, $P = Ss\theta$ is a* plan for Gs given R using Ss *iff there exist extended resource schemes (RS,C) and (RS',C') and a* ground *substitution* θ *such that*

1. $(RS,C) \vdash_{Ss} (RS',C')$;
2. $RS\theta \subseteq R$;
3. $\models C\theta$ *and* $\models C'\theta$, *i.e., both constraints evaluate to true under* θ
4. θ *is resource-identity preserving w.r.t.* $RS \cup RS'$;
5. $RS'\theta \models GS$.

Note that as a consequence, every resource scheme occurring in a plan P is ground and therefore a resource. So both $In(P)$ and $Out(P)$ are sets of resources. Some of the resources occurring in $Out(P)$ are used to satisfy some goals, other resources are not used so and occur as *free resources* or *side-products*.

Example 2.13 *Consider again the drive and travel skills discussed before,let $Ss = \{\textbf{drive} < \textbf{travel}\}$ and let $t(adam,rdam) = 40$. Then $P = Ss\theta$ with $\theta = \{x = adam, y = rdam, t = 100, t' = 90\}$ is a plan for $GS = (\{pass(rdam,t'')\},\{t'' < 150)\})$ using Ss.*

3 Cooperation in the resource skill framework

Multi agent planning At this point the reader might wonder why we used services as an intermediate step before introducing plans. The reason is that we want to use services to enable cooperative planning.
Consider a number of agents $A_1, A_2, \ldots, A_n$ having the sets of resources $R_1, R_2, \ldots, R_n$, respectively, at their disposal. Suppose the agents have agreed upon to split up their common goal scheme GS into n goal schemes $GS_1, GS_2, \ldots, GS_n$, such that $GS = (\bigcup G_i, \bigcup C_i)$.
Using services $Ss_1, Ss_2, \ldots, Ss_n$, the agents have constructed plans $P_1, P_2, \ldots, P_n$ to realise their goals schemes. For each plan P_i, it is possible to compute the costs of P_i. For example, these costs can be computed as follows:

$$costs(P_{i)} = \sum_{r \in In(P_i)} costs(r) + \sum_{s \in P_i} costs(s) - \sum_{g \in G_i} value(g)$$

Assuming that they have been able to create these plans on their own, we might consider the situation that the coalition of agents tries to reduce their plan costs, while maintaining goal realisability without having to rely on additional resources.

Assuming that costs of input resources and values of the goals are the same for every agent involved, the only possibility to reduce the plan costs is to reduce the number of skill applications in the plans.
We might consider the following possibilities to reduce plan costs:

1. *exchanging goals*
   Suppose that an agent A_i has a plan P_i such that one of the final skill applications in this plan realizes a collection G' of goals. Without this skill application the remaining part P_i' of his plan is sufficient to realize the remaining set of goals. If the other agents have free resources in the output of their plans that are able to realize G', A_i can use P_i' as a cheaper plan, while the agents collectively are still able to realize the total set GS of goals.

2. *exchanging resources*
   More in general, each agent A_i could try to find a smaller plan by *(i)* selecting some occurrence of an instance of a skill s in his plan P_i, *(ii)* computing the set of resources that are missing to complete his plan if s would be removed from his plan and *(iii)* asking the other agents for delivering the missing resources. If A_i succeeds in this process, while the other agents are still able to complete their plans, we have obtained a better plan by *fusing* the original plans of the agents. Note that 2. contains 1, as a special case.

3. *exchanging resources and skills*
   except resources, we could also exchange skills or services: suppose that agent A_i has to realise a goal scheme GS_i. Now it might occur that he can produce some intermediate set R_1 of resources in a cheap way, but the remaining part of the production process is rather costly. On the other hand, agent A_j has a service he can use very cheaply to produce a set of resources R_2 from R_1 to realize GS_i. Then the agents could exchange services in order to make the total plan more profitable. Note that is different from 2. since the service owned (known) by A_j might not be actively used by A_j. Furthermore, the costs of a plan based on this service might be different for the agents.

Fusion: an overview We have studied the second possibility for multi-agent cooperation, based on reducing plans by removing skills and exchanging resources ([4, 5]). This process results in a fusion of separate plans, where some of the resources needed by agent A_i are delivered as outputs of plans of other agents.
This fusion process can be modeled as an iterated plan revision process. During a single plan revision step, a given agent A_i will try to reduce its plan P_i by removing a skill application. Removing a skill implies that

1. the resources produced by this skill have to be provided either by using other (output) resources produced in his plan P_i or by asking other agents to provide them.

2. the input skills consumed by this skill are added to the free resources of agent A_i.

Now A_i first will try to use his own free resources in order to provide for the resources missing; if there are still missing resources, he will ask the other agents if they can provide free resources he can use. Note that using the service Ss_i on which his plan is based, A_i does not need to rely on specific resources; instead, using Ss_i, the agent asks for resources satisfying an extended resource scheme that is computed from the remaining plan and the underlying service. If A_i succeeds, he will use a smaller plan P_i' to realise his goals, else he will keep his original plan (momentarily).
As the result of such a revision process, nothing has been changed globally, that is the collection of agents still is able to satisfy the collection of goal schemes given the total collection of resources; the difference is that the total cost involved may have been reduced.
In each revision step goal-realisability is guaranteed: if a goal node g is affected, other agents are asked either to take over g or to handover missing resources to realise g fusion process stops if no agent is capable to remove a single skill from its plan.
This fusion process has been implemented in a fusion algorithm described elsewhere ([4, 5]).

4 Conclusions and future work

One of the unique features of multi-agent systems is that individual agents can *cooperate* in order to achieve their goals. One reason to cooperate is that agents cannot realize their goals individually; another reason is that cooperation leads to a more efficient means to realize their goals. In this paper, we concentrated on the latter.
We described a computational framework, consisting of resources and skills, to model cooperation processes between different agents. Central in this framework is that we model *side products* explicitly, so that other agents can exploit unused resources.
Another result of this framework is an algorithm that in polynomial time[5] *fuses* the individual plans of a set of agents, such that the joint profit of the agents does not decrease.
Future research will be focused on studying cooperation *during* the plan construction process. In our framework, this means that except exchanging resources, we will also model cooperation between agents w.r.t. the skills and services they manage. Extensive literature about joint plan construction is available (see, e.g., [2, 3, 1, 6, 7, 8]), and we believe this formalism helps us to model the negotiation process.

References

[1] M. d'Inverno, M. Luck, and M. Wooldridge. Cooperation structures. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan*, pages 600–605, 1997.

[2] Eithan Ephrati and Jeffrey S. Rosenschein. Multi-agent planning as the process of merging distributed sub-plans. In *Proceedings of The Twelfth International Workshop on Distributed Artificial Intelligence*, pages 115–129, May 1993.

[5] Provided that constraints can be checked efficiently.

[3] D.E. Foulser, Ming Li, and Qiang Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2–3):143–182, 1992.

[4] H. Tonino M.M. de Weerdt, A. Bos and C. Witteveen. A plan fusion algorithm for multi-agent systems. In *CL-2000 Workshop on Computational Logic in Multi-Agent Systems, (CLIMA-00)*, London, 2000.

[5] B.-J. Moree, A. Bos, H. Tonino, and C. Witteveen. Cooperation by iterated plan revision. In *Proceedings of the ICMAS 2000*, 2000.

[6] J.P. Müller. *The Design of Intelligent Agents: a layered approach*. Springer, 1996.

[7] E. Werner. Distributed cooperation algorithms. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I.*, pages 17–31. Elsevier Science Publishers B.V., 1990.

[8] M. Wooldridge and N.R. Jennings. The cooperative problem solving process. *Journal of Logic & Computation*, 9(4), 1999.