Keeping a Clear Separation between Goals and Plans

Costin Caval[1,2], Amal El Fallah Seghrouchni[1], and Patrick Taillibert[1]

[1] LIP6, Paris, France
{costin.caval,amal.elfallah,patrick.taillibert}@lip6.fr
[2] Thales Airborne Systems, Elancourt, France

Abstract. Many approaches to BDI agent modeling permit the agent developers to interweave the levels of plans and goals. This is possible through the adoption of new goals inside plans. These goals will have plans of their own, and the definition can extend on many levels. From a software development point of view, the resulting complexity can render the agents' behavior difficult to trace, due to the combination of elements from different abstraction levels, i.e., actions and goal adoptions. This has a negative effect on the development process when designing and debugging agents. In this paper we propose a change of approach that aims to provide a more comprehensible agent model with benefits for the ease of engineering and the fault tolerance of agent systems. This is achieved by imposing a clear separation between the reasoning and the acting levels of the agent. The use of goal adoptions and actions on the environment inside the same plan is therefore forbidden. The approach is illustrated using two theoretical scenarios as well as an agent-based maritime patrol application. We argue that by constraining the agent model we gain in clarity and traceability therefore benefiting the development process and encouraging the adoption of agent-based techniques in industrial contexts.

Keywords: goal directed agents, goal reasoning, goal-plan tree.

1 Introduction

In the field of intelligent agents, BDI agents are used extensively due to their proactivity, adaptability and similarity between their abstract representation and the human reasoning. These agents are enticed with *beliefs* to cover their view of the world, a reason for their behaviors in the form of *desires* or *goals*, and a description of the means to act, in the form of *plans* or *intentions*.

In the original BDI model proposed by Rao and Georgeff [1], the "matching" between goals and plans is assured through a cycle that considers the options for *desires*, deliberates on them to update the existing *intentions* and then executes the actual actions. In more practical approaches, automata are used to handle the life-cycle of goals from their adoption to the appropriate plan selection and execution [2,3].

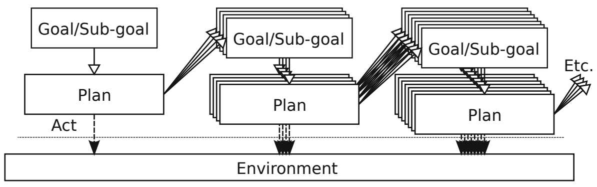

Fig. 1. Agent complexity when goals are adopted in plans acting on the environment

The purpose of an agent is usually to act on the environment, which is done through its plans. Actions can involve the use of actuators, but they also cover the sending of messages[1]. However, in practice, various works [3,4] and programming frameworks (Jason [5], Jadex [6] etc.) employ a model where plans can also adopt new goals, often termed *sub-goals*. A goal can thus have multiple possible plans, whose success depends on the achievement of their respective sub-goals and this can extend on many levels (Fig. 1). Note however that the successful completion of a plan does not necessarily guarantee the achievement of a goal, as goals can have success and failure conditions [7].

While it may be straightforward to design in this way, the fact that in a plan (1) actions on the environment – i.e., with effects "outside" of the agent – and (2) goal adoptions – i.e., with effects on the, possibly long-term, reasoning and behavior of the agent – are used together in the same structure can have adverse effects on the resulting agents: low intelligibility during design, difficult traceability during execution and poor reusability afterwards.

This recursive construction has the advantage of using already existing BDI building blocks and can help abstract certain aspects of an agent's behavior offering the possibility to define the agent in a top-down approach. However, it also creates a structure which is difficult to trace, especially when actions occur at any level, and whose depth may be unpredictable. Important aspects in the behavior of an agent might be hidden from the eyes of a developer or code reviewer due to this intricate design. One might always wonder whether the current plan is a terminal one or whether the model continues with further sub-goals. Given that the adoption of a goal usually implies a new reasoning process with an automaton and further plans, the goal adoption shouldn't be treated the same as an atomic action.

For a change of perspective, let us take the example of the army as a clear-cut multi-level organization. A soldier executes the orders (goals) given from "above" but cannot make high level decisions. Strategies and new objectives (goal adoptions) are decided by the higher ranks. This is due to the separation of responsibilities and competences, as well as the soldier's limited view of the situation. In a similar way, an agent's goals should not be mixed with the acting. This would also allow plans to have limited interdependencies, just as the soldier has a limited view of the situation, with benefits on complexity and fault confinement. A similar analogy can be made with other hierarchical human

[1] We do not consider belief revision to be an action.

organizations such as companies, where the management decides – either on a single or at multiple levels – before requiring the workers to perform the required tasks. Needs that can arise have to be discussed with the manager or managers, who can then decide to take new measures, just as an agent's reasoning would adopt new goals. While small companies with a "flatter" hierarchy can cope with certain issues faster, complex organizations have proven to benefit from this hierarchical composition[2].

Agent oriented development methodologies such as Tropos [8] and Prometheus [9] have top-down approaches where they start with system level characteristics to then "descend" towards agent goals before defining plans and other low level details. Implementing agent systems modeled using such methodologies would also be more natural if reasoning and acting were more clearly separated.

Several works [10,11,12,7] have argued for the interest of using declarative *goals-to-be* together with procedural *goals-to-do*, for decoupling goal achievement (the "to be" part) from plan execution (the "to do" part), giving the agents their *pro-activeness*, but also better flexibility and fault tolerance. Taking this delimitation a step further, we argue for the interest of separating a level where goal reasoning takes place – managing goal adoptions, dependencies, conflict resolution – from an action level where the agent interacts with its peers and environment.

While at runtime it is useful and even inevitable to alternate between reasoning and acting, we argue that these already conceptually distinct levels should be kept separate when designing agents.

To address these issues we propose a subtle change in the agent modeling that simplifies the agent representation by requiring the actions on the environment to be separated from the goal adoptions. We call the approach *Goal-Plan Separation*, or *GPS*. As shall be seen, the direct consequence of this separation is the structuring of the agent into two levels: one concerned with goals and one concerned with actions.

This paper is organized as follows. Section 2 presents the original approach of the paper which is illustrated on two examples. Section 3 discusses implementation issues and Sect. 4 some aspects of the goal execution. Section 5 presents an experimentation in the domain of maritime patrol. In Sect. 6 we discuss some fault tolerance issues with respect to the experimentation. Section 7 addresses the related work and Sect. 8 concludes the paper.

2 The Goal-Plan Separation

In this section we introduce a representation model from the literature which we use to illustrate our proposition through a first generic example. This allows us to discuss the consequence of the Goal-Plan Separation, followed by the more refined example of a Mars rover.

[2] Note: while we are presenting examples of organizations with many people, our scope remains the design of the reasoning of a single agent, which would thus correspond to the army or the company as a whole.

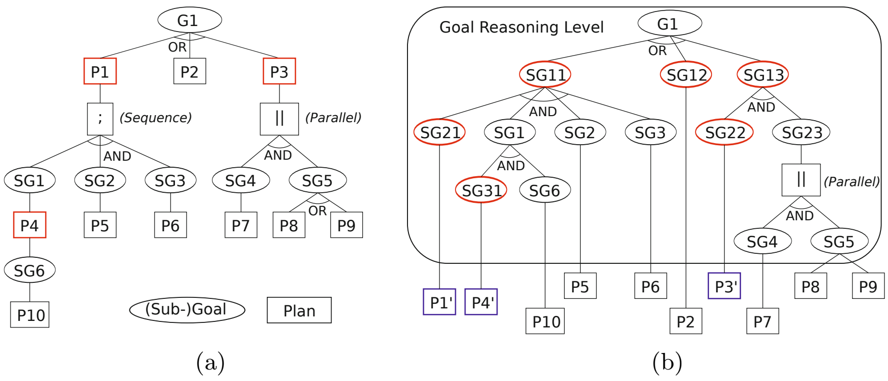

Fig. 2. An example of goal-plan tree (a) and a goal-plan separation of the same example (b)

2.1 Goal-Plan Trees to Goal-Plan Separation

Thangarajah [4,13] formalizes the representation of the agent model in the form of an AND-OR tree: the *goal-plan tree*, or *GPT*. Goals are *OR* nodes since their child nodes, the plans, offer alternative solutions and only one plan suffices for the achievement of a goal. Plans on the other hand are *AND* nodes in order to denote the obligation to achieve all the adopted sub-goals for a successful plan execution. Furthermore, two operators are added to the plan node, to indicate either that the goals have to be achieved in sequence (;) or in parallel (||). A generic example which illustrates all these is given in Fig. 2 (a). Here, the GPT using the two operators spreads in depth across several levels. Note that there can be more than one tree for a given agent, in other words more than one root goal. We chose this model because even if it is used more as an analysis than a development tool (see Related Work in Sect. 7), it shows well the issues we are addressing, in particular how the goal and plan levels alternate.

To illustrate the Goal-Plan Separation approach, the generic example was modified to obtain a possible goal-plan separation, as seen in Fig. 2 (b). The plans that are the most important here are *P1*, *P3* and *P4* as they are the ones that can contain both actions on the environment and goal adoptions. The new representation, which decomposes goals into sub-goals is an AND-OR tree (very similar to the one used in [14]) with only the leaf nodes having plans containing actions, but no goal adoptions. To save space, we consider that the default operator for the *AND* nodes is the sequence operator, unless stated otherwise, e.g., in the case of *SG23.* To preserve the original structure, goals are also allowed to be *OR* nodes, in order to depict cases where a goal or sub-goal can be achieved in more than one way. Similarly, goals that have more than one plan are *OR* nodes. While the original goals were preserved, the plans that were not leaves were replaced by sub-goals, e.g., *SG11.* To compensate, plan names of the form *P'* were used to indicate a variation of an original *P* plan which at least removes the goal adoptions. Note, however, that this exact transformation is not unique for the given example as it depends on the plan's specific features[3]. More examples can be seen in Sect. 2.3. *SG12* was introduced to avoid the existence of siblings of different types. This example shows that transforming an existing agent is possible. Nevertheless, as is the case with many such translations and as we discovered during the experimentation we describe in Sect. 5, a complete redesign of the agent produces a more appropriate result.

2.2 The *Goal Reasoning Level*

As can be seen in Fig. 2 (b), a direct consequence of the separation of goal adoptions from the actions on the environment is the appearance of two levels in the definition of the agent: a *goal reasoning level* and an *action level*.

The *goal reasoning level* is the part of the agent concerned with goal adoption, control, dependencies and interactions. In this paper, we are concerned mostly with the specification (by a programmer or designer) of dependencies between goals and issues related to the adoption and life-cycle control. For the purpose of the Goal-Plan Separation, no actions on the environment are present at this level. However, as will be discussed further on, other mechanisms can appear at this level, e.g., for handling perceptions, events or various types of goal dependencies.

2.3 Mars Rover Scenario

To further illustrate the GPS, let us consider a Mars rover example from [13]. Figure 3 (a) represents a goal-plan tree for a Mars rover's goal to analyze soil samples. The depth of the tree varies between *P7: ExpSoilByDelegationPlan* that is at a depth of one and *P6: TransmitTo(Lander)Plan*, at a depth of 5. While all leaf nodes are plans, there are also intermediary plans which adopt goals and can contain actions: *P1: ExpSoilBySelfPlan* and *P4: RecordResultsPlan.* If these two plans had no actions on the environment, the representation would be GPS-compliant as no unwanted action-goal adoption mix would be present. In this case, an alternative representation can also be obtained in the same manner as in the example in Sect. 2. As depicted in Fig. 3 (b), *P1* changes into a sub-goal and *P4* disappears completely as there is already *SG3* to regroup the corresponding sub-tree. For *P7*, a parent sub-goal *SG12* is created to avoid having two siblings of the *G1* node of different types, i.e., a goal and a plan. *SG12* also carries the precondition originally contained by *P7*.

Another approach would be to rewrite the Mars rover's behavior in a format similar to the goal diagram from Tropos [8], as in Fig. 3 (c). The representation can also be seen as a type of plan. It starts with a decision node that corresponds to *P7*'s precondition from the original scenario. The sequence operator is represented through the arrows that depict the dependencies between goals,

[3] E.g., a plan that *turns on a sensor*, adopts a goal to *retrieve data* and then *saves that data*. Such a plan would rather transform into a main goal with three sequential sub-goals, the first corresponding to the beginning of the original plan, and the last corresponding to its final part.

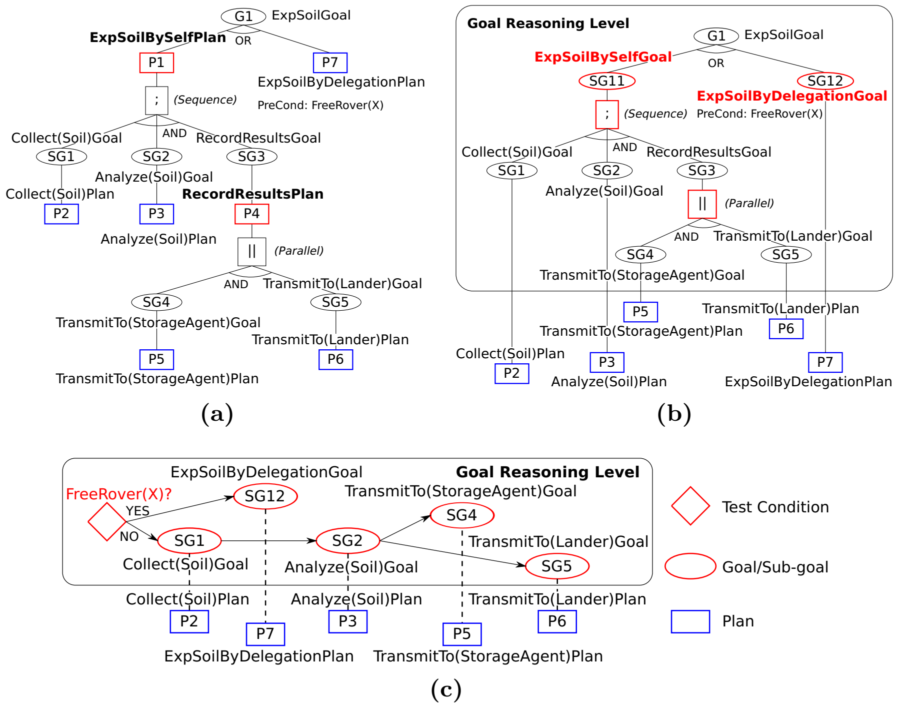

Fig. 3. (a) the goal-plan tree of a Mars rover from [13], (b) a translation of the Mars rover scenario in the form of a GPS-compliant AND-OR goal decomposition and (c) a modified representation of the scenario with a clear goal-plan separation

while the parallelism is implied through the fact that two arrows start from the same entity, in this case *SG2*.

If, however, *P1* and *P4* also contained actions on the environment, the transformation would become more complicated. Figure 4 shows only the sub-tree starting from *SG3* with three simple examples of possible cases: (1) actions in parallel with, (2) before or (3) after the goal adoptions. This shows the hidden complexity associated with the action-goal mix.

The examples in this section obey the GPS principle since in each case, the two levels, the goal reasoning level and the plan level, can be clearly distinguished. This shows the applicability of the Goal-Plan Separation is not restricted to a specific goal reasoning formalism.

3 GPS Method Implementation

Throughout the evolution of programming, languages and development tools often advanced by limiting the programmer's freedom to access lower level elements such as registers and pointers to data, and offering in exchange higher level tools and constructs such as variables and dynamically created references to

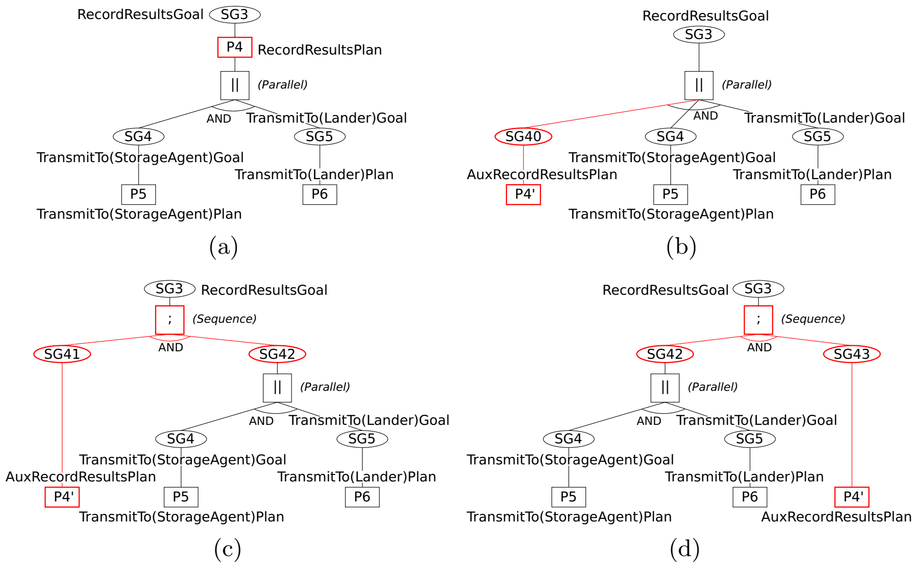

Fig. 4. Transformation of the $SG3$ sub-tree (a) from the Mars rover scenario (Fig. 3) into a GPS-compliant form, in some of the non trivial cases: $P4$ contains actions on the environment that happen in parallel with the goal adoption (b), $P4$ contains actions on the environment that happen before (c) or after (d) the goal adoption

data. These evolutions allowed for the creation of increasingly complex systems while decreasing the possibilities for coding errors. Similarly, we do not refrain from restraining the freedoms of the programmers and designers in the interest of clarity and reliability.

To achieve the goal-plan separation, rather than adopting sub-goals, at execution time an agent's action level (usually composed of action plans) would accomplish the necessary actions and then relinquish control to the higher level where the reasoning and possibly a following goal is adopted. This creates, as illustrated in the examples above, a distinct goal reasoning level where an agent's goals are chosen and their execution is managed.

As shall be discussed in this paper, the representation on multiple levels, either by using sub-goals or through other mechanisms, is important for the scalability and intelligibility of the resulting agents and therefore constitutes an important characteristic of the models that should be at least taken into consideration for the goal reasoning level.

In [15], GPTs are used as support for a study on plan coverage and overlap, with the hypothesis that the plan libraries discussed have no cycles. This is important to note as in the general case adopting goals inside plans may produce cycles, sometimes even with unwanted consequences similar to the infinite loops in classic programming. We, on the other hand, do not restrict cycles, as will be seen in the scenario in Sect. 5. However, the Goal-Plan Separation doesn't allow cycles created through plans that also have actions on the environment.

As the Goal-Plan Separation approach in its simplest form is the requirement to keep a clear distinction between the two abstraction levels, it is general enough so that it can be applied using any of the BDI frameworks that allow goal adoptions in plans. The important condition, however, is to make sure no goals are adopted in plans that act on the environment. Examples of representations that can be used are given next, followed by a more detailed description of a model based on what we call *goal plans* and that we use in Sect. 5.

3.1 Examples of Possible Models for the Goal Reasoning Level

Reasoning through Rules. Using goal trigger rules, an almost "reactive" agent can be created. The goal relationships are implicit but a dependency tree similar to the one seen in Fig. 3 (c) above can be constructed at runtime for tracing purposes. This reasoning model can be implemented in Jadex by simply specifying trigger conditions for each goal but without creating explicit connections between these goals. The advantage of this approach is that the representation can handle more complex systems that act in highly dynamic environments, with new goals added effortlessly. However, this model lacks *look-ahead* capabilities.

Reasoning Using a Planner. Rather than having goals simply triggered by rules, a planner can be used to select among available goals, as for example in CANPLAN [7]. The difference then from the reasoning model described above is that this time the reasoning allows the choice of goals to be prepared in advance starting form the current context. Another difference is that a planner would render the agent proactive, as it would not have to wait for events in order to act. The job of the planner would be to select, order and parallelize goals according to the current needs, and for this it could use certain operators [16]. The example in Sect. 5 does not correspond to this method as no planner is used and its *goal plan* (see below) is defined at design time. Our intuition is also that the GPS approach benefits this model as planning should be easier to perform only on goals, without the interference of details from actions.

3.2 Reasoning through a *Goal Plan*

Between the reactivity of the first model above, and the planning capabilities of the second, we propose here a middle solution that allows for a certain level of *look-ahead* owing to the use of pre-written goal dependencies, just as plan libraries can be used with BDI systems. As required by the GPS method, the goal reasoning level should be kept separate from the plans that handle action composition. Considering that relations between goals can be similar to those between actions, we can envisage using a modified plan language to represent the relations between goal adoptions. We call the resulting plans that handle goal composition *goal plans* and we oppose them to *action plans.*

$$\begin{aligned}
P &= <N, E> \text{ // action plan}\\
N &= A \cup O \cup T \text{ // nodes}\\
A &= \{action \mid action \neq goalAdoption\}\\
O &= \{o \mid o \in \{startNode, finishNode,\\
&\quad AND,\ \ \|, wait(duration)\}\}\\
T &= \{test(stateCond) \mid stateCond \in\\
&\quad \{Beliefs, Events\}\} \text{ // conditions}\\
E &= \{n_1 \rightarrow n_2 \mid n_1, n_2 \in N\} \text{ // edges}
\end{aligned}$$

(a)

$$\begin{aligned}
GP &= <N_g, E> \text{ // goal plan}\\
N_g &= A_g \cup O \cup T \text{ // nodes}\\
A_g &= \{adopt(G) \mid G \in Goals\}\\
O &= \{o \mid o \in \{startNode, finishNode,\\
&\quad AND,\ \ \|, wait(duration)\}\}\\
T &= \{test(stateCond) \mid stateCond \in\\
&\quad \{Beliefs, Events\}\} \text{ // conditions}\\
E &= \{n_1 \rightarrow n_2 \mid n_1, n_2 \in N\} \text{ // edges}
\end{aligned}$$

(b)

Fig. 5. Action plan (a) compared to goal plan (b). Only the action nodes differ.

As defined in Fig. 5, a *goal plan* is an oriented graph with three types of nodes:

– A_g, the *goal adoption nodes*, as the unique action allowed in the goal plan. Each node represents the invocation of an automaton associated with the goal. Note that this is the only distinction from the action plans which have $A = \{action \mid action \neq goalAdoption\}$.
– O, the *operator nodes*, with operations including a unique *start node* and at least one *finish node*. Different finish nodes can be used to indicate final states for a plan, e.g., "successful completion" or "partial failure". There is also an operator for branching parallel threads and one for the logical condition *AND* that can be used to synchronize threads or to indicate the obligation of two or more conditions to be all true, for example to require several goals to be achieved in order for the execution to continue.
– T, the *condition test nodes* that can handle state conditions for belief values and events such belief change and message arrival. They can either be used to test for a momentarily condition, or to wait for a condition to become true or for a message to arrive.

Edges indicate the succession of nodes in the goal plan and, as stated before, cycles are possible, for example to indicate a recurrent goal adoption.

The Mars rover scenario in Fig. 3 (c) with its inline goal dependencies can easily be transformed into a goal plan, as seen in Fig. 6. There are two possible finish nodes, with one for a successful mission where either *G7* or both *SG4* and *SG5* were achieved, and one to indicate all other cases as failures.

While implicit relations between entities (such as the rule-triggered goals above) may be enticing due to their ease of definition and generality, they are also difficult to follow and may hide unwanted interactions. The goal plans, however, favor the use of *explicit* specifications of dependencies between goals. If for example a Mars rover needs to perform an experiment at a location X and it has two goals for achieving this, one being G_1 = *"move to X"* and the other G_2 = *"drill"*,

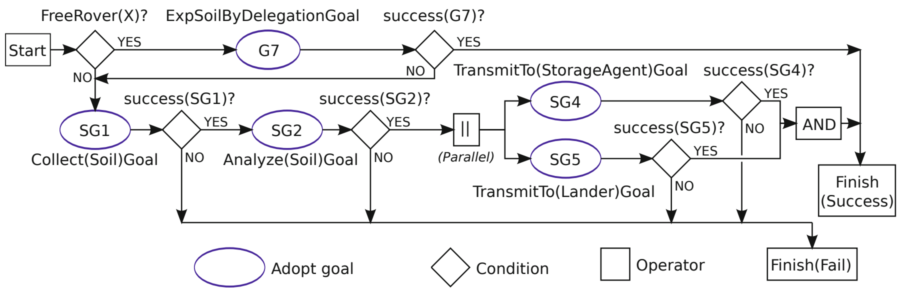

Fig. 6. A goal plan for the Mars rover scenario from Fig. 3

then it is clearer to link the adoption of G_2 to the successful achievement of G_1 rather than for example the belief that the rover is at location X.

In a framework like Jadex, this model can be implemented using a plan that is triggered at agent's birth. The plan would specify the dependencies between sub-goals and adopt them *without any other actions*.

In practice, this model can become difficult to manage as the agent grows in complexity. A solution to this problem is to group together parts of the goal plan and abstract them into *sub-goal plans*, that are to be expanded only when needed. In this way, the representation can still be conceptually on one level, while having the advantages, in particular the scalability, of a hierarchical representation.

This kind of reasoning is suitable for agent systems where the behavior can be thoroughly specified at design time so that all dependencies can be accurately included. Adding new goals and other modifications, however, are difficult to apply. The first implementation described in Sect. 5.2 corresponds to this approach.

3.3 Reasoning through Multiple Goal Plans

The method above has the advantage of providing a "big picture" of the agent's behavior but, as stated before, does not scale well to complex agents. Designing the behavior of an agent that can run for hours can for example create a large goal plan that is difficult to follow and which risks being too rigid in case of unforeseen events. The solution is then to decouple the sub-goal plans from their "parent" goal plans by using goals to manage the expansion, in other words, by allowing any goal not only to have action plans, but also goal plans. This means using the "classic" BDI mechanisms – i.e., goals, plans and automata – with just the subtle difference in the construction of plans: no goal adoption will be in the same plan as an action on the environment. Note, however, that in this case the states indicated through finish nodes do not necessarily reflect the achievement or failure of the parent goal, as the goal would normally have its own conditions for success and failure. Figure 7 (a) shows the Mars rover's behavior represented

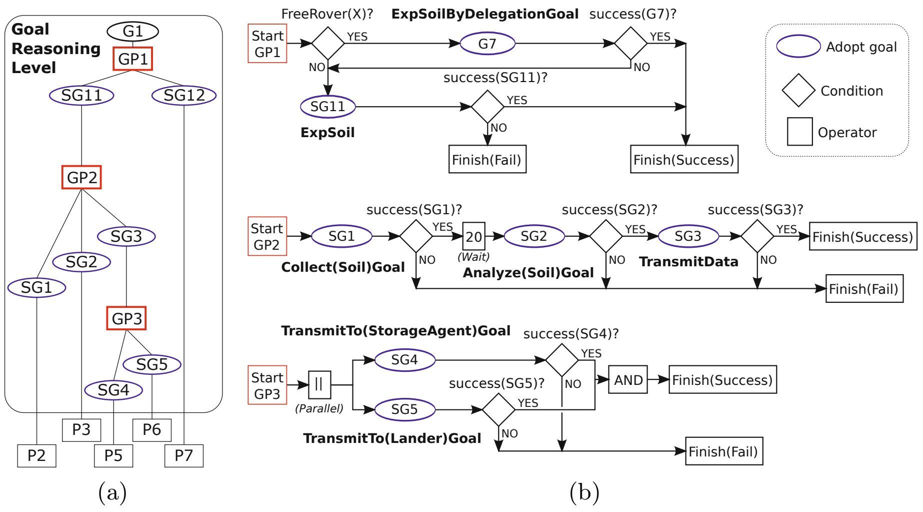

Fig. 7. A multiple level goal plan for the Mars rover scenario from Fig. 3, with (a) the resulting tree (similar to a goal-plan tree) representation and (b) the corresponding goal plans. Note the separation in (a) between the action plans, i.e., $P2$, $P3$, $P5$, $P6$ and $P7$, and the goal reasoning level comprising the goals and the three goal plans, i.e., $GP1$, $GP2$ and $GP3$.

with this model. The resulting model can be represented through a structure that is similar to the GPT as can be seen in the Fig. 7 (b), but this tree contains fewer details as more logic is included in the goal plans, while in the same time complying with the GPS approach.

There are many advantages of this multiple goal plan model. First of all, splitting the behavior into more levels of goals and sub-goals with the corresponding plans improves flexibility and fault tolerance – in case a plan fails, the BDI logics can require a retry using the same or a different plan, provided that such plan is available. Then, splitting the behavior into more manageable chunks leaves less room for hidden faults. The use of goal plans for managing goal dependencies allows for a more refined specification than what was available through the *AND*, *OR* and the operators in the GPT. For example, in Fig. 7, the suite of goal adoptions in *GP2* does represent the sequence that was originally in the GPT, but other operators – such as the *delay* in the example – can be added through this specification, and precise goal failures can be handled accordingly (while not present in the given example, one could add other goals to account for these specific sub-goal failures). This model is therefore preferred to the simple goal plans presented above, and is illustrated in the second implementation in Sect. 5.2.

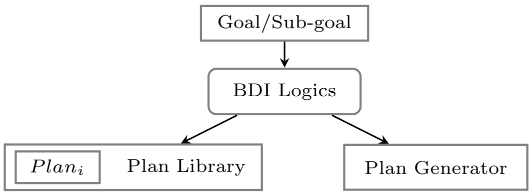

Fig. 8. BDI logics: handler of the goal-plan relation at runtime

4 Execution

While not explicitly presented in the GPT, as stated before and seen in Fig. 8, between the goal and plan levels there are the BDI logics or more commonly a goal automaton [2,3] which handles the goal life-cycle. This life-cycle usually starts with the adoption of the goal and includes the choice and execution of plans.

An example of a goal life-cycle for which an automaton is used is depicted in Fig. 9. It uses a series of beliefs for state changes, such as *desirable* (*des*) to indicate the presence in the automaton, *selected* (*sel*) to indicate the passage in an active state and *satisfaction* (*sat*) that indicates if the goal was achieved. We use these beliefs to control the execution of goals by linking them to other beliefs that justify them, for example the goal adoption conditions for *desirable.* In case any of these conditions is no longer valid, the belief is no longer justified so the automaton changes its state automatically, which in the case of the *desirable* belief means that the goal is aborted. If we take the example in Fig. 6, supposing that during the execution of *G7* the condition *FreeRover(X)* is contradicted by an observation, the adoption of the goal will no longer be justified and the goal will fail automatically. It is also important to note that as a higher level means of control, the goal reasoning level has precedence over the action level.

Beliefs can also be used to control the goal automaton from the higher level in a more straightforward manner, if for example we added another operator that causes a goal to abort its execution.

For the GPS approach the automaton is a black box that is given a goal to adopt and possible plans to execute and this is why we represent only goals and plans in our modeling examples. The execution can cause side effects such as belief changes that can lead the reasoning level to take actions with respect to current goal or even the adoption or execution of other goals. For example, this can cause the goal to be aborted in case it is estimated to take the agent in an unsafe state, or it can cause the adoption of a reparation or compensation goal to counter certain unwanted effects. Note that several automata can function at a given moment as parallelism is allowed in our method. While conflicts are normally treated at goal reasoning level and can even be explicitly handled in the goal plans, conflict management is not within the scope of this paper.

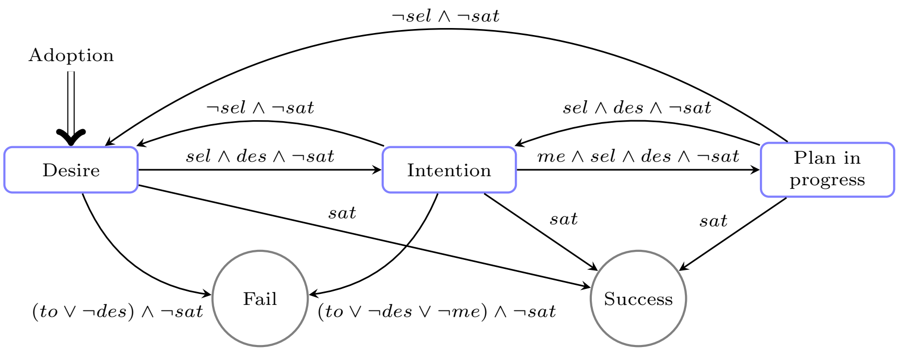
Adoption
Desire
Intention
Plan in progress
Fail
Success
$\neg sel \wedge \neg sat$
$\neg sel \wedge \neg sat$
$sel \wedge des \wedge \neg sat$
$sel \wedge des \wedge \neg sat$
$me \wedge sel \wedge des \wedge \neg sat$
sat
sat
sat
$(to \vee \neg des) \wedge \neg sat$
$(to \vee \neg des \vee \neg me) \wedge \neg sat$

Fig. 9. Our generic goal life-cycle with transition conditions on state beliefs (des = $desirable$, sel = $selected$, sat = $satisfied$, me = $means$, to = $timeout$)

5 Experimenting with GPS

The GPS approach has been experimented in an industrial context at Thales Airborne Systems on an application designed for experimenting on AI in general and more precisely on Interval Constraints propagation and multi-agent systems (MAS). The purpose of this application, Interloc, is the localization of boats from a maritime patrol aircraft. It is implemented as a MAS and can contain dozens of agents implemented as Prolog processes.

Interloc was initially designed as a set of non goal-directed autonomous agents. This means that the agents had only one purpose that was achieved through a set of associated plans. Subsequently, it was redesigned in order to improve the level of autonomy of the agents by endowing them with goals. The pursuit of intelligibility brought along the idea of having a clear separation between the levels of abstraction of goals and plans.

A first implementation in the spirit of GPS used a goal plan formalism as the one described in Sect. 3.2. This meant designing a plan where the only possible action was goal adoption. For the ease of use, sub-goal plans – which anticipate the hierarchical approach later implemented – were also used, adding their activation to the goal adoption as the only possible "actions" in the goal plan. The intention of the designer (prior to the GPS methodology presented in the present paper) was to exhibit an abstract (goal) level describing the main features of the behavior of agents so that one would find it sufficient to only read the goal level description in order to understand the salient behavior of the agents. Agents were then implemented following the idea described in Sect. 3.3 as the flexibility and robustness of goals seemed preferable to the simple invocation of sub-goal plans.

In the pursuit of a more formal representation, we abstracted the goal plans into Time Petri Nets, TPNs [17], seen in Figs. 10 and 11 (b-e). We chose the TPNs because they present many advantages through their graphical and intuitive representation, as well as their expressive power (parallelism, sequence, synchronisation etc.). This extension over classic Petri nets gives the possibility of assigning firing time intervals to the transitions, which we used for representing waiting in the agent behavior. Furthermore, the TPNs allowed us to structurally verify the goal plans and ensure their correctness. We also used a type of Petri net that resemble the Recursive Petri Nets (already used for representing agent plans [18]) where we distinguished two types of transition: the elementary transitions to be fired according to the standard semantics of Petri nets and the abstract ones corresponding to the action of adoptiong a goal. However, the expansion of this action, the goal adoption, is not handled in this network, and its transition corresponds to a call to the associated automaton, e.g., the one in Fig. 9.

We first present the application itself, then the particular case of one of the main agents, the aircraft, in the two goal plan-based implementations mentioned above. This section concludes with a discussion on the advantages of the GPS approach in the specific case of the Interloc application.

5.1 Interloc

The main goal of the application is the localization of boats using a *goniometer*[4] on-board a maritime patrol aircraft. The sole use of a goniometer allows for a stealth detection, i.e., detect without being detected, of boats which is important for some missions such as gas-freeing prevention[5]. If the boats were steady, the problem would be simple. The fact that they move necessitates a reliance on non-linear regression methods, as is the case of existing commissioned implementations, or interval constraint propagation, in Interloc. Most of the agents, i.e., boats, the goniometer and the data visualization agent, were designed for the purpose of simulation. The main agent, the aircraft, must (1) follow all the boats visible from its location, (2) compute in real-time their position by accumulating bearings and interacting with computation agents (more precisely *artifacts* [19]) operating interval propagation, (3) adapt its trajectory to observations and contingencies and (4) transmit results to the visualization agent. For the patrol aircraft, boats may appear or vanish at any time. Several aircraft might be present at the same time, but so far they do not communicate with each other. Typically 20 to 30 agents or artifacts are active in the system at a given time.

5.2 The Aircraft Agent

Boats and aircraft have been designed following the GPS method. We present here the aircraft, which is the most complex agent type and hence the most interesting for illustrating the methodology.

Five goals corresponding to five main activities of the agent were identified:

[4] Tool which displays the direction towards the source of a signal, in this case a boat and its radar.

[5] Deterring tankers from polluting the environment by cleaning their fuel tanks at sea.

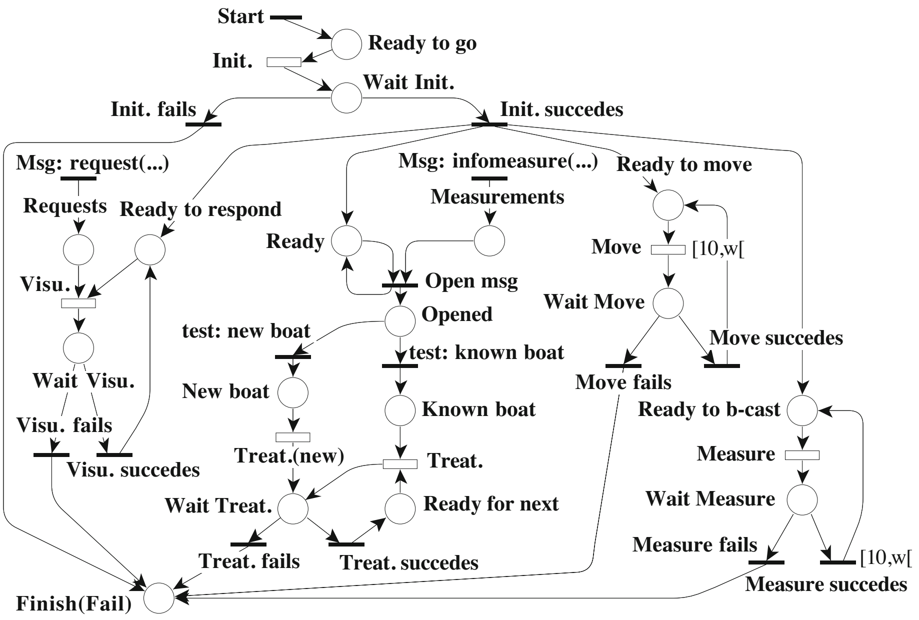

Fig. 10. Petri net representation of the goal plan for the aircraft agent with goal adoptions represented as hollow transitions. The *Treat.* (for "treatment") goal is adopted in two different contexts in order to insure that messages from each boat are treated sequentially, but in parallel with the other boats. Multiple instances of the goal with different beliefs are thus created.

- *Init.* (for "initialization") of the system: get data related to the aircraft trajectory (pre-defined, planned or human-guided) and various parameters characterising the simulation
- *Move*: execute one *step* forward
- *Measure*: initiate measurement of the bearing of all the visible boats
- *Treat.* (for "treatment"): process a received measurement
- *Visu.* (for "visualization"): process a single request from the visualization agent

The sole knowledge of the various goals present in the system is not sufficient to understand (and define) its behavior. One must also describe the way in which these goals are adopted and what happens when they are achieved, for example by specifying their chronology, conditions for becoming a desire, conditions for becoming an intention. This knowledge may be provided in different forms, corresponding to the different ways of applying the GPS approach.

Using a Single Goal Plan. For the first implementation we present here, the aircraft agent in Interloc was designed using a goal plan with four sub-plans to indicate the dependencies of the goals above. These dependencies correspond to the goal reasoning level in the GPS approach.

Informally, the goal plan is the following (a more formal description of this plan is given in Fig. 10 as a Petri net): the achievement goal Init. is adopted. If the goal is not achieved, the system is halted. Otherwise, four sub-branches implemented as sub-goal plans are activated in parallel: *main_move*, *main_measure*, *main_visualization* and *main_analyze*.

The *main_move* sub-plan:

- Wait for a *move_time_step* delay
- Adopt the Move goal, whose associated plans will compute and execute the next time step
- Wait for the Move goal achievement
- Loop

The *main_measure* sub-plan:

- Adopt the Measure goal, where the associated plans will measure the bearings of all the visible boats through interactions with the measurement artifact and the (simulated) boat agents
- Once achieved, the goal will be re-adopted after a given time delay

The *main_analyze* sub-plan:

- Wait for a measurement, in the form of a message that arrives randomly after a measurement request message is issued
- Record the newly present boats
- Adopt the Treat. goal, whose associated plan will generate a constraint to be added to the previously received measurements and send it to an interval constraint propagation artifact which will compute a more and more precise boat location
- Loop, in order to process waiting measurements

The *main_vizualization* sub-plan:

- Wait for a request from the visualization agent
- Adopt the Visu. goal in order to process the request
- Wait for the achievement
- Loop to process pending requests

Using the Multiple Levels of Goal Plans. When the pursuit for flexibility and robustness pushed us further and we separated the goal plans and their sub-goal plans through new goals, we obtained the tree structure seen in Fig. 11 (a). *GP1*, in Fig. 11 (b), guides the adoption of four intermediary goals that are internal to the goal reasoning level, i.e., they do not have action plans. *GP2*-*GP5* correspond roughly to the sub-goal plans described above and can easily be matched with the corresponding branches in the initial one-level goal plan (Fig. 10).

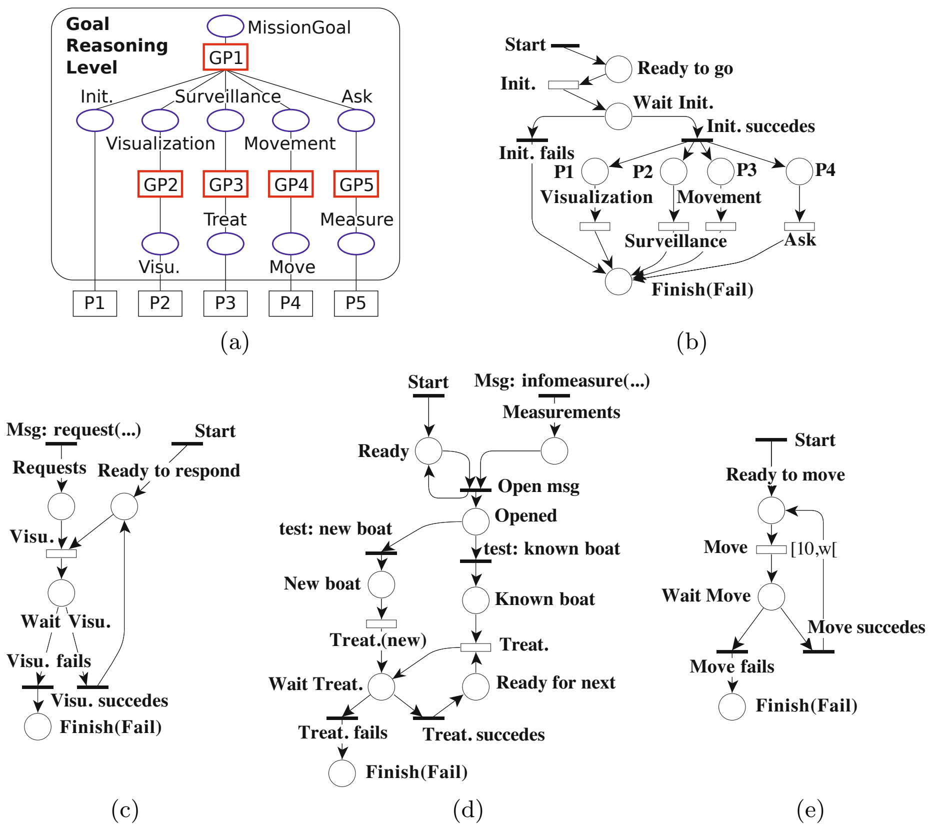

Fig. 11. (a) the goal-plan structure of the aircraft agent, (b-e) Petri net representations of the *GP1* (b), *GP2* (c), *GP3* (d) and *GP4* (e) goal plans. *GP5* is not presented because it is very similar to *GP4*, as can be deduced from Fig. 10. Goal adoptions are represented as hollow transitions.

5.3 Discussion

With GPS, Iterative and Timed Behaviors Appear at Goal Level: In the pre-GPS version of the application, the natural tendency was to incorporate dynamic aspects into the plans, making them fairly complex. For instance, the Move goal was not conceived as a single step as presented above, instead, it was charged with the complete management of the aircraft's trajectory, including the loop sequencing individual steps. This rather straightforward design would close the loop inside the plans and after the actions on the environment – e.g., the movement or broadcast of measure request messages – were performed. The *move time-step*, which is important for the global understanding of the behavior of the aircraft, was also "buried" in the plan pursuing the goal. In the GPS-compliant versions, deciding to rewrite the plan and change the scope of the goal to the achievement of a single movement step, created the need for the definition of the time-step and the iterative behavior at the goal reasoning level, leading to a clearer design. The fact that such details are at an upper level of abstraction emphasizes their importance and improves the understanding of the agent behavior.

With GPS, Relevant Perceptions of the Environment Are Required at the Goal Reasoning Level: It is the case of messages coming from the visualization or the measurement agents. Here again, it emanates from the fact that certain perceptions can be essential for the global understanding of the agent behavior. In Interloc, measurements trigger the adoption of a goal whose achievement is more or less secondary since other measurements can occur rapidly. That is the reason why it seems to be a good approach to handle these measurements at the upper level of abstraction. A perception filtering strategy, to avoid unnecessary inputs or even overloading the agent, can also appear in this goal plan, possibly by the adoption of a specific goal prior to the adoption of the Measure goal itself.

With GPS, Handling Errors Is Easier to Take into Account: This is because errors, whatever their cause, often manifest through the failure of goals. This provides an adequate range of exception mechanisms in the language in which plans are written. Hence, the programmer's effort with regard to fault tolerance is mainly to take into account the processing of non-achieved goals. Of course, this does not concern the goal plan itself, which has to be designed traditionally by explicitly introducing fault tolerance actions. However the amount of code regarding the classic plans is far greater than the amount of the goal plan code. In the Interloc application, no specific fault tolerance effort has been carried out but a clean processing of non-achieved goals in order to stop the system rather than have it crash. As a consequence, application debugging was greatly facilitated. For the same reasons, the GPS approach proved to facilitate the evolution of the multi-agent system. Thus, the aircraft agent was easily changed into a UAV (Unmanned Autonomous Vehicle), with a larger autonomy in the trajectory choice. Here again, the abstraction obtained by separating goals and plans seems to be the reason.

In Interloc, we used an in-house agent programming language (Alma) to implement the goal plans. All the required primitives were available, since a goal plan is a type of plan. Nonetheless, it appears that specific primitives could be introduced to facilitate the programming of the goal level. These concern mainly iterative and time-controlled behaviors.

6 Discussion on the Fault Tolerance with Goal Reasoning

In real life applications agents tend to have more refined representations than the ones discussed in Sect. 2. In particular, when it comes to handling errors, the specification easily grows in complexity as specific cases have to be taken into consideration [20]. Goals give agents a level of abstraction that is beneficial for a system's robustness as errors, exceptions, anomalies etc. usually occur during plan execution which, in a robust[6] system, only cause the plan to fail and the goal automaton to react normally and reattempt to achieve the goal. While there are studies that treat the more general case of partial goal satisfaction [21] (described below), if we only consider a binary goal definition, a goal's adoption has only two possible outcomes at reasoning level: the goal is either achieved or not. Requiring the programmer to specify not only the actions to take after the achievement of a goal, but also the actions to take in case the goal fails enhances the reliability of the agent without dramatically increasing its complexity.

In the Mars rover scenario represented in Fig. 6, the failure to delegate the task to another agent, i.e., the failure of *G7*, causes the rover to attempt to accomplish the mission by itself through the adoption of *SG1*. Similarly, in the aircraft specification of the Interloc application (Sect. 5), both the successful achievement and the failure of goals are represented in the Petri net and also in the implementation. However, for simplicity reasons, in our example, no special actions are taken and the only result of a goal failure is to ensure the agent does not reach unforeseen states. Also, the current format implies an infinite life for the agent, which is not necessarily desirable in a real application.

In the paper cited above [21], goal satisfaction is evaluated using a *progress metric*. Partial goal satisfaction could be integrated with our model by enforcing the coverage of the whole range of possible values for the progress metric used. For example for a Surveillance goal, instead of specifying success and fail behaviors, it could be interesting to estimate the percentage of the assigned area that was covered and to use thresholds for the desired behaviors: less than 30% would be considered a mission failure with the area announced as *unsafe*, a coverage between 30 and 80% would require a call for backup to finish the job, while a coverage of more than 80% would be considered a success. Note that this does not concern the intermediary stages such as those that are handled by the goal automata, but final goal failures, i.e., when all alternatives have been tried and no positive outcome resulted.

7 Related Work

The aspect of the Goal-Plan Separation that handles goal reasoning is situated at what Harland et al. [3] and Thangarajah et al. in earlier works [4,13] call *agent deliberation level*. This is where agent goals are *considered*, which constitutes the point where goals start their life-cycle. It is the same level where *top level commands* are issued to interfere with the goal life-cycle, e.g., when deciding to drop or suspend the goal. As the cited authors point out, goal deliberation can deal with issues such as goal prioritization, resource management and even user intervention. These aspects are beyond the scope of this paper but can be considered for future developments of our approach. We note, however, that in [3] changes in the goal state have preference over any executing plans, just as in

[6] In this case, we understand by *robust* an agent system in which an error or exception in a plan is caught and only causes that plan to fail, while the rest of the agent continues to function normally, i.e., does not cause the whole agent to fail.

the case of GPS, where the goal reasoning level takes precedence over the lower levels that it controls, i.e., the goal life-cycle automata and the plan execution.

The arguments for planning in BDI agents at goal level employed by Sardina and Padgham [7] offer more reasons for the existence of the goal reasoning level (be it hardcoded, created through planning or other means) that the GPS approach delimits: *"(a) important resources may be used in taking actions that do not lead to a successful outcome; (b) actions are not always reversible and may lead to states from which there is no successful outcome; (c) execution of actions take substantially longer than "thinking" (or planning); and (d) actions have side effects which are undesirable if they turn out not to be useful"*. All these advocate for an agent that behaves strategically and proactively rather than react based on a limited context, and it is at goal reasoning level that such a strategic reasoning is possible. The multi-level goal plan structure proposed in Sect. 3.3 allows for both complex "strategic" and simple "reactive" behaviors ($GP3$ vs. $GP2$ in Fig. 11).

While this paper discusses the goal reasoning level in the need to better organize the levels "below", i.e., the plans, Morandini et al. [14] approach the same level from a different perspective: the need to fill in the gap between goal based engineering and goal implementations. They propose a tool for transforming an agent designed using the Tropos methodology [8] into Jadex code, for which they introduce a formalism based on rules for the life-cycle of non-leaf goals in a goal hierarchy. This segregation between leaf and non-leaf goals creates a goal level that corresponds to our goal reasoning level and thus their work is consistent with the GPS approach. This further confirms our statement with respect to the utility of a goal-plan separation for the implementation of goal-based methodologies. Furthermore, our proposition of using goal plans on multiple levels means that even goals that are internal to the goal reasoning level will have the same life-cycles as goals that use action plans. A specific life-cycle, as proposed by Morandini et al. is therefore no longer needed, deeming the development process easier, as there are less types of goals to consider. One of the interesting aspects is that Morandini et al. take into account the fact that even if the sub-goals are achieved, the parent goal may still fail due to its own achievement condition, which is often not taken into consideration when discussing the goal-plan trees. While this formalism is rich and GPS-compliant, as our application example shows, our approach aims to provide a model that allows for a more refined representation, with more diverse goal relations, event-based goal reasoning and time constraints.

There are many parallels that can be drawn between our approach and the one employed by the Prometheus agent development methodology [9] in the detailed design phase. This is where functionalities identified in the previous phases of the methodology – system specification and architectural design – are used as a starting point for designing capabilities. A capability is a module within the agent that can contain further capabilities, and at the bottom level plans, events and data, e.g., capability C_1 uses data D or plan P_1 sends message to plan P_2. Internal messages are used to connect between different design artifacts, such as plans and capabilities. This functionality is assured by either beliefs or direct goal dependencies in our work. This nested structure of capabilities is similar to the sub-goal plans (Sect. 3.2) in its pursuit of *"understandable complexity at each level"*, and while semantically different, it does provide a very similar functionality to our goal reasoning level. Furthermore, the use of internal messages to indicate dependencies between internal artifacts (mostly capabilities and plans) creates a very similar structure to our goal plans where we explicit dependencies between goals, often guided by tests on beliefs and messages. In Prometheus, BDI goals at agent level can be represented through a specific type of event, because events can trigger plans. As events, i.e., goal events, but also messages, percepts and internal messages, can be produced in plans as well as in outside the agent, a clearly defined goal reasoning level in the GPS sense cannot be delimited in the current form of the methodology. The Goal-Plan Separation approach would, however, benefit from the integration with the first two phases of the Prometheus methodology: the system specification and the architectural design. Due to the fact that these two phases correspond to a top-down design approach, and also, as we showed above, the fact that there are already similarities in the current form of Prometheus, we feel that such an integration would be possible, resulting in a methodology tailored for goal-directed GPS agents.

In [22] Pokahr et al. address the issue of *goal deliberation*. This concept is not equivalent but rather included in our goal reasoning level as they consider only goals that have already been adopted. Their work focuses on the similar issue of goal interactions, i.e., when goals interfere positively or negatively with each other, and they base their proposed strategy on the extension of the definition of goals. They include for example inhibition arcs that block the adoption of a certain goal or type of goal when another goal is adopted. Such mechanisms can be integrated when specifying the goal reasoning level discussed in our approach.

The goal automaton proposed by Braubach et al. [2] presents a goal state labeled "New" with a "Creation condition" acting as a triggering condition for the goal before the adoption and the actual goal life-cycle. This state, together with the condition are at the level of our goal reasoning level. A goal that was defined for the agent is considered to be in the "New" state, as opposed to a goal that can for example be received from the exterior or generated through the agent reasoning. Only when such a goal is received does it pass into the "New" state. All the goals discussed in the examples in this paper are already in this state.

The goal-plan trees have been used in various works for representing agent specifications and as a basis for further treatments. In [4] GPTs are used to gather resource requirements called summary information and identify possible goal interactions. This is due to the hierarchical structure of the tree where summary information can be propagated upwards towards the root of the tree. Further works on the subject [3] reuse the model to illustrate their operational semantics for the goal life-cycle. Furthermore, Shaw et al. propose different approach for handling goal interactions using Petri Nets [23] and constraints [24] instead of GPTs. These, as well as other works that use GPTs, such as [25] on intention conflicts, can be used with GPS, and our intuition is that by separating the goal reasoning level, goal interactions can be managed more easily.

Singh et al. [26] use learning for plan selection in BDI agents. They also use GPTs to describe the agents and even note briefly that only *"leaf plans interact directly with the environment"*, which is consistent with the GPS approach. This allows for a representation where, given the results – i.e., *success* or *fail* – of the executions of all leaf nodes, the success or failure of the root node is decided by simply propagating these logic values in the AND-OR tree. This is a confirmation of the benefits of the GPS approach, for, if actions were included in intermediary plans, even if all sub-goals of a plan were achieved, the plan would not necessarily cause the achievement of its parent goal. The GPT is therefore already a simplification of the system, as it uses the rather strong hypothesis that there are no perturbations, such as the one in the afore-mentioned case, in the AND-OR tree. Another example of "perturbation" in the propagation of success values in the tree can be the use of specific achievement and failure conditions for each goals [7,14].

Note that, while we use the GPT representation to justify our approach, the GPS is concerned with more general agent models. Also, this paper does not argue against the GPT formalism, neither does it dispute the plethora of works that use it as a model, but rather discusses the more general issue of specifying agents with interwoven goal and action levels. The current paper complements the cited works on goal interactions as it concerns the agent specification rather than the runtime mechanisms that aim to improve the efficiency, proactivity, reactivity etc. of the agents.

Another representation used for resource handling is the *task expansion tree* described in [27]. This tree represents the decomposition of a task (a concept similar to goals in our work) into subtasks. The particularity is the introduction of special *composite tasks* that are used to compose other tasks in a functional manner. These include, besides the sequence and parallel operators present in the GPT model described in this paper, other tasks that allow other types of branching and tests. The use of these operators in a tree structure situates their model between classic goal hierarchies and our goal plan.

Clement et al. [28] champion the advantages of abstraction for solving various problems such as large scale planning and scheduling. They argue that by abstracting the less critical details in a large problem, the overall solution is easier to find, and can then be expanded to the actual detailed solution. This applies well to our Goal-Plan Separation approach, as well as to their approach on planning in a hierarchical way. They extend HTNs (hierarchical task networks) to take time into consideration and use summary information at higher levels in the HTN to identify possible interactions between plans while working with abstract actions (which are similar to the BDI concept of goal). HTNs are quite similar to goal hierarchies in that they too offer a gradual refinement for the behavior of an agent from the more abstract to the actual actions. The advantage of using goals instead of "abstract plans" is given by the flexibility and resilience offered through the goal life-cycles where a goal's achievement can be attempted through various plans, with different constraints etc. Nevertheless, our work does not exclude the possibility of using HTNs for plan selection, for example in a similar fashion with CANPLAN [7].

8 Conclusion and Future Work

In this paper, we argued that the separation of reasoning and acting is important for the specification and construction of BDI agents. It was shown that the possibility to mix actions on the environment with goal adoptions in various agent models and languages can have negative effects on the resulting representation and can hinder the development process. A series of examples illustrated what an agent would look like when complying with the *Goal-Plan Separation* approach, with emphasis on the two resulting levels: a *goal reasoning level* and an *action level*. As a possible representation for the former, *goal plans* were introduced. The GPS therefore imposes a constraint on agent design that does go against the reflex of adopting a goal in any place it is needed but produces a better-structured result. The GPS also results in agents that "step back and look at the overall picture" rather than react "rashly" to their current situation, making it suitable for "strategic", proactive and complex behaviors, without necessarily neglecting the reactive ones, e.g., *GP2* in Fig. 11. The importance of tidy agent representation lies with the ease of development, which can, in turn, facilitate the wide-scale adoption of the development model. Furthermore, a clean representation that helps diminish the number of design and development faults and also improves maintainability helps bring the overall project costs down.

As discussed in the paper, on the side of BDI agent modeling there are many studies on goal representations and goal life-cycles. However, the higher level that is placed above these automata is less examined in the literature and constitutes a point of this paper that we plan as a further study. For this, a more in-depth research on specifying the agent's goal reasoning will have to be undertaken. Among other primitives, the handling of temporal constraints is important for agent systems and should be taken into consideration. Furthermore, as stated above, there are fault tolerance aspects related to this direction in agent development that can be exploited. We are particularly interested in the use of GPS and goal-directed agents in general for designing multi-agent systems that can better cope with faults that were not foreseen at design. As presented in Sect. 5, we have already began the empirical evaluation of the approach and its advantages on agent design on a real time application. However, more evaluations will be necessary in order to extend and generalize the GPS approach. In the long run, the goal is to integrate this approach in an agent development methodology.

Acknowledgement. The authors would like to thank the anonymous reviewers for their insights on this work. The authors would also like to acknowledge the usefulness of the discussions at the the EMAS14 Workshop that helped refine and advance our research on this topic.

References

1. Rao, A.S., Georgeff, M.P.: BDI-agents: From theory to practice. In: Proceedings of the First International Conference on Multiagent Systems, pp. 312–319. AAAI Press, San Francisco (1995)
2. Braubach, L., Pokahr, A., Moldt, D., Lamersdorf, W.: Goal representation for BDI agent systems. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2004. LNCS (LNAI), vol. 3346, pp. 44–65. Springer, Heidelberg (2005)
3. Harland, J., Morley, D.N., Thangarajah, J., Yorke-Smith, N.: An operational semantics for the goal life-cycle in BDI agents. Autonomous Agents and Multi-Agent Systems 28(4), 682–719 (2014)
4. Thangarajah, J., Padgham, L.: Computationally effective reasoning about goal interactions. Journal of Automated Reasoning 47(1), 17–56 (2011)
5. Bordini, R., Hübner, J., Vieira, R.: Jason and the golden fleece of agent-oriented programming. In: Bordini, R., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15, pp. 3–37. Springer US (2005)
6. Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: A short overview. In: Net.ObjectDays 2004: AgentExpo. (2004)
7. Sardina, S., Padgham, L.: A BDI agent programming language with failure handling, declarative goals, and planning. Autonomous Agents and Multi-Agent Systems 23(1), 18–70 (2011)
8. Giunchiglia, F., Mylopoulos, J., Perini, A.: The tropos software development methodology: Processes, models and diagrams. In: Giunchiglia, F., Odell, J.J., Weiß, G. (eds.) AOSE 2002. LNCS, vol. 2585, pp. 162–173. Springer, Heidelberg (2003)
9. Winikoff, M., Padgham, L.: Developing Intelligent Agent Systems: A Practical Guide. Wiley Series in Agent Technology. John Wiley and Sons (2004)
10. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Agent programming with declarative goals. In: Castelfranchi, C., Lespérance, Y. (eds.) Intelligent Agents VII. LNCS (LNAI), vol. 1986, pp. 228–243. Springer, Heidelberg (2001)
11. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and procedural goals in intelligent agent systems. In: Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning, pp. 470–481. Morgan Kaufman, Toulouse (2002)
12. Dastani, M., van Riemsdijk, M.B., Dignum, F., Meyer, J.-J.C.: A programming language for cognitive agents goal directed 3APL. In: Dastani, M. M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2003. LNCS (LNAI), vol. 3067, pp. 111–130. Springer, Heidelberg (2004)
13. Thangarajah, J.: Managing the Concurrent Execution of Goals in Intelligent Agents. PhD thesis, RMIT University, Melbourne, Australia (2005)
14. Morandini, M., Penserini, L., Perini, A.: Operational semantics of goal models in adaptive agents. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 129–136. International Foundation for Autonomous Agents and Multiagent Systems, Budapest (2009)
15. Thangarajah, J., Sardina, S., Padgham, L.: Measuring plan coverage and overlap for agent reasoning. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 2, pp. 1049–1056. International Foundation for Autonomous Agents and Multiagent Systems, Valencia (2012)

16. Chaouche, A.C., El Fallah Seghrouchni, A., Ilie, J.M., Sadouni, D.E.: A higher-order agent model for ambient systems. Procedia Computer Science 21(0), 156–163 (2013), The 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks and the 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare
17. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time petri nets. IEEE Transactions on Software Engineering 17(3), 259–273 (1991)
18. El Fallah Seghrouchni, A., Haddad, S.: A recursive model for distributed planning. In: Proceedings of the 2nd International Conference on Multi-Agent Systems, pp. 307–314. AAAI Press, Kyoto (1996)
19. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the a&a meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems 17(3), 432–456 (2008)
20. Torres-Pomales, W.: Software fault tolerance: A tutorial. Technical report, NASA Langley Research Center, Hampton, Virginia, USA (2000)
21. van Riemsdijk, M.B., Yorke-Smith, N.: Towards reasoning with partial goal satisfaction in intelligent agents. In: Collier, R., Dix, J., Novák, P. (eds.) ProMAS 2010. LNCS, vol. 6599, pp. 41–59. Springer, Heidelberg (2012)
22. Pokahr, A., Braubach, L., Lamersdorf, W.: A goal deliberation strategy for BDI agent systems. In: Eymann, T., Klügl, F., Lamersdorf, W., Klusch, M., Huhns, M.N. (eds.) MATES 2005. LNCS (LNAI), vol. 3550, pp. 82–93. Springer, Heidelberg (2005)
23. Shaw, P., Bordini, R.H.: Towards alternative approaches to reasoning about goals. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2007. LNCS (LNAI), vol. 4897, pp. 104–121. Springer, Heidelberg (2008)
24. Shaw, P., Bordini, R.H.: An alternative approach for reasoning about the goal-plan tree problem. In: Dastani, M., El Fallah Seghrouchni, A., Hübner, J., Leite, J. (eds.) LADS 2010. LNCS (LNAI), vol. 6822, pp. 115–135. Springer, Heidelberg (2011)
25. Shapiro, S., Sardina, S., Thangarajah, J., Cavedon, L., Padgham, L.: Revising conflicting intention sets in BDI agents. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 2, pp. 1081–1088. International Foundation for Autonomous Agents and Multiagent Systems, Valencia (2012)
26. Singh, D., Sardina, S., Padgham, L., James, G.: Integrating learning into a BDI agent for environments with changing dynamics. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, vol. 3, pp. 2525–2530. AAAI Press, Barcelona (2011)
27. Morley, D.N., Myers, K.L., Yorke-Smith, N.: Continuous refinement of agent resource estimates. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 858–865. ACM, Hakodate (2006)
28. Clement, B.J., Durfee, E.H., Barrett, A.C.: Abstract reasoning for planning and coordination. Journal of Artificial Intelligence Research 28(1), 453–515 (2007)