Specifying, Verifying and Implementing a MAS: A case study

Bruno Mermet, Gaële Simon, Bruno Zanuttini, Arnaud Saval

GREYC - UMR 6072

Abstract. This paper deals with the design of multi-agent systems. We demonstrate the goal-oriented agent model called Goal Decomposition Tree on an already studied multi-agent example, that of robots which must clean pieces of garbage on Mars. As we show, the model allows to prove that the agents' behaviour indeed achieves their goal. We then compare our approach to other ones.

1 Introduction

Goal Decomposition Trees (GDT) have been introduced by Simon *et al.* [19] as a model for specifying the behaviour of agents in a multi-agent system (MAS) together with a complete approach for the design of MAS. This approach consists in three steps:

1. an **agentification** step which helps the designer to determine the set of agents which must be used to implement a given system;
2. a **behaviour specification** step using an agent design model (GDT) which helps to design an agent behaviour that can be verified by a specific proof system;
3. an **implementation** step using an implementation model based on automata which can be automatically generated from the agent design model.

Thus the aim of this global approach is to provide a complete MAS design process starting from the problem specification and ending with an implementation. Central to this approach is the fact that it allows to produce verified implementations of agents' behaviours.

The goal of this paper is to present two important add-ons to the GDT model and to demonstrate the second point above and the associated proof step on an already studied example: two robots which must clean pieces of garbage on Mars. The main add-on consists in the introduction of *external goals* to the model: this is an important step to the verification of a whole multi-agent system. The second one allows to increase the power of the proof system thanks to the introduction of the *Guaranted Properties in case of Failure*. The scenario studied has been proposed by Bordini *et al.* [2] for demonstrating model checking of Rao's AgentSpeak language [16]. They have proposed agents' behaviours for this scenario and verified them using model checking. We have chosen this scenario because the goal of [2] (i.e. behaviour specification and proof) is very close to ours.

Thus this example allows us to compare the GDT model to the AgentSpeak one. We specify the agents' behaviour using the GDT model, mimicking as much as possible the behaviour specified by Bordini *et al.* in order to facilitate the comparison. It turns out that the GDT model is as rich and concise as AgentSpeak, and allows more elements to be formally taken into account, especially (atomic) actions. Moreover, the GDT model also allows to prove the correctness of the agents' behaviours, whatever the number of pieces of garbage or the size of the grid modelling the surface of Mars. This is to be opposed to the model checking method presented by Bordini *et al.*, which only allows to verify the MAS on a finite number of grids in finite time (5×5 grids with 2 pieces of garbage in the paper).

The paper is organized as follows. In Sections 2 and 3 the GDT model, its new extensions and its proof system are specified. Then we present the Mars scenario, specify the behaviours of the agents using GDTs and verify their correctness (Section 4); in the light of this example, we compare our model to the AgentSpeak language in details. Then we compare our work to other approaches (Section 5), and finally we conclude.

2 Goal Decomposition Trees

In our approach, the behaviour of agents are represented by *Goal Decomposition Trees* (GDT). These are trees whose nodes are goals, defined by a satisfaction condition and associated either to atomic actions or to further decompositions into subgoals. The GDT of an agent specifies its whole behaviour, and the satisfaction condition of its root node is thus its main goal. GDT are presented in details in [19], but we give here the notions relevant to the paper.

Nodes As already said, nodes (either leaves or internal nodes) correspond to the goals of the agent. To each node G a *satisfaction condition* (SC) is associated. Intuitively, a goal is satisfied if and only if its SC is made true. SCs are expressed over a restricted form of temporal logic, in which the states of variables used before and after trying to achieve the goal can be distinguished. For instance, if the SC of goal G is $x' > x \wedge x' \neq 0$, G is achieved if x has been incremented and is now nonnull.

Actions The behaviour associated to a leaf goal (except external goals, detailed later) is described either by a list of assignments or by a *named action* (NA), i.e., an atomic action which consists in a name, a list of parameters, a precondition and a postcondition. Intuitively, the postcondition must entail the SC of the leaf goal.

Operators Each internal node of a GDT is associated to a decomposition into subgoals, linked up with an operator. Eight such operators are defined in [18]. For instance, *SeqAnd* is a classical lazy and ordered logical *And* operator, and *Iter* allows to repeat a subgoal until the parent goal is achieved. Importantly, operators are associated to *automata composition patterns*, which are used incrementally to build the complete automaton which implements the behaviour specified by the GDT. For more details see [18].

Typology of goals In order to add flexibility to the specification and to take nondeterminism into account, goals have *types* according to two criteria. First of all, a goal (internal or leaf) can be *necessarily satisfiable* (NS) or not (NNS). In the former case, the decomposition into subgoals or the action associated to the goal always makes its SC true. In the latter case, the decomposition or action may fail to satisfy the goal. For instance, for an internal goal decomposed thanks to an `AND` operator, the father goal may fail if one of its subgoal fails. But if both subgoals succed, also does the father goal.

Orthogonally, goals can be *lazy* (L) or not (NL). When the agent has to achieve an L goal, it first checks whether its SC is true, and only in the negative executes the decomposition or action. On the contrary, the agent must always execute the decomposition or action of an NL goal. E.g., SCs which directly link the values of the variables before and after the goal execution, such as $x' > x$, can be associated to NL goals only.

Along with these two criteria, the types of each internal node in a GDT can be automatically determined by the types of its children together with the semantics of the decomposition operator. Consequently, if specified by the designer, types can be used to check the consistency of the specification.

External goals External goals have been added to the model presented in [19]. Such a goal E in the GDT of an agent A is one which A cannot achieve (for instance because it depends on variables that A does not control). Thus an SC is as usual associated to E, but no action or decomposition, because another agent (verifying an *external property* P) is expected to make it true. Consequently, in this article, an external goal is an NS leaf goal together with a link to a goal G in another GDT. The semantics is that when it must achieve E, agent A waits until an agent with this latter GDT achieves its goal G, which will make the SC of E true.

External goals are a way to express dependencies between agents, that is to say collaborative agents. In particular, it can be seen as a specialization of "nonlocal tasks" of TAEMS (there is no contracting with our external goals). Moreover, an external goal as the left operand of a *SeqAnd* can be seen as a specification of an "enables" interrelationship in TAEMS. A more detaild comparison with TAEMS can be found in [19].

GDTs A GDT is a tree built up from nodes as specified above. In addition, the following are associated to a GDT.

A set of variables specifies all environment variables together with a set of internal variables of the agent. All formulas are built upon this set. A *triggering context* (TC) specifies when the agent must execute its GDT (either the first or each time it becomes true, depending on the agent). A *precondition* ($PrecGDT$) is also given, which must be satisfied before the execution begins. In particular, a given *initialisation clause* achieves it when the agent is created, and for GDT executed several times, the precondition must be true again after each execution (in other words, before any GDT execution, $PrecGDT$ is true). Finally, an *invariant* describing the constraints of the problem is given, which must be preserved through the whole execution.

3 The Proof process

Our aim is to prove the correctness of the GDT built for an agent (i.e., to prove that the behaviour specified by the tree always achieves the main goal of the agent). For each operator described in section 2, several *proof schemas* are defined according to types of goals. These schemas are intended to produce *Proof Obligations* that describe what must be proved in order to validate a goal decomposition. These schemas have been proved to be correct with respect to a semantics of GDTs in LTL not described here and under the assumption that actions are atomic and that parallelism can be represented by an interleaving model.

Since a future goal is to use a theorem prover to perform proofs, proof schemas are built in a rigorous manner to be automatized. Moreover, the compositional aspect of proofs makes proof simple, maximizing the success rate of an automatic theorem prover.

Applying these schemas to a GDT results in an agent's behaviour compositional proof. Each proof is performed using a context that can be computed by a *context propagation schema* associated to each operator that is not described here but that can be found in [13].

3.1 Notations

Variables The set of environment variables is denoted by V_e, and the set of internal variables of a given agent by V_i. For each agent, we assume $V_i \cap V_e = \emptyset$, and whewe define $V = V_i \cup V_e$. Internal variables cannot be modified by another agent, while environment variables are variables that the agent can see and modify, but so can other agents or the environment itself.

Goals The SC of a goal G is written SC_G. For the proof process, a *context* is also associated to G, which intuitively expresses what is known to be true when G is about to be attempted. In particular, the context of the main goal is $TC \wedge PrecGDT$ (defined in section 2), and the context of the other nodes is inferred from the GDT by *context propagation schemas* no detailed here. The context of G is written C_G. Finally, still for the proof process, something has to be known about the outcome of the resolution attempt of an NNS goal. This is expressed by a *Guaranted Property in case of Failure* (GPF). The semantics is that if the solving process of a goal fails (to satisfy its SC), then its GPF is still true. The GPF of a goal G is written GPF_G.

GDTs The triggering context of a GDT is written TC, its precondition is written $PrecGDT$, its initialisation clause (an assignement) is written `init`and its main goal is denoted MG. Its invariant is written $I = I_S \wedge I_A$, where I_S is the invariant of the system (over V_e) and I_A is that of the agent over V.

Temporal notation In the SC of a goal G, the value of a variable x before and after executing the actions or decomposition associated to G are distinguished by primes: e.g., $x' < y$ means that the value of x after the agent has tried to achieve G is less than that of y was before this attempt.

However, if the SC does not relate both moments, only unprimed variables are used. This is for sake of consistency when considering the evaluation of the

SC of a lazy goals, before any execution. For instance, if the goal is to set x to at least 2, then its SC is written $x \geq 2$. In the proof schemas we thus use a function, denoted T, such that for any goal G, primed variables in $T(SC_G)$ describe relations between the variables when G is achieved. Thus, for instance, $T((x' < y)) = (x' < y)$, while $T((x \geq 2)) = (x' \geq 2)$.

substitution We note $[x := y]P$ the syntactic substitution of any free occurence of x by y in P.

Transition between two goals solving process When considering two goals G_1 and G_2 resolved sequentially (e.g., when proving a *SeqAnd* decomposition), From the point of view of the agent, three states can be distinguished: S, the state right before the solving process of G_1, S^{tmp}, the state between the G_1 and G_2 solving processes, and S', the state right after the solving process of G_2. to unify variables v' after the G_1 solving process with the variables v before the G_2 solving process corresponding both to the value of variable v in state S^{tmp}, we replace them by variables v^{tmp} by the two following substitutions: $[v' := v^{tmp}]SC_{G_1}$ and $[v := v^{tmp}]SC_{G_2}$.

Projection if F is a temporal logic formula and S_v a set of variables, we write F_{S_v} the projection of F to the variables of S_v. For instance, if $F = x < y \wedge x > 3$, $F_x = x > 3$. If $S_v = V_i$, we simply F_{V_i} by F_i.

3.2 Proof schemas

In this section, a few proof schemas are detailed. The normal proof process requires to prove that the invariant is preserved by each goal of the GDT. However, when a GDT is fully specified, performing this proof for leaf goals is enough.

In addition, since most of the goals of R1 in the following case study are NS, we give proof schemas only for this kind of goals, and so GPFs are not involved. However, as one of the *Iter* operator in the example has an NNS subgoal, the *Iter* proof schema is given for this kind of subgoal.

Initialisation one must prove:

$$[\texttt{init}](PrecGDT \wedge I_A) \tag{1}$$

Moreover, for agents that can execute their GDT several times, the following property must also be proved:

$$I \wedge T(SC_{MG}) \wedge T(I\ ) \Rightarrow T(PrecGDT) \tag{2}$$

SeqAnd: Proving $A \Leftarrow B\ SeqAnd\ C$ (when A is NL) requires to prove:

$$I \wedge C_A \Rightarrow \left(\left\{ \begin{array}{l} [v' := v^{tmp}]\ T(SC_{B_i}) \\ [v := v^{tmp}]\ T(SC_C) \end{array} \right\} \Rightarrow T(SC_A) \wedge T(I) \right) \tag{3}$$

From the point of view of the agent, the state of its internal variables (and only them) after its attempt to achieve B is unchanged when it begins to try to achieve C; hence the substitutions of v' by v^{tmp} and of v by v^{tmp} and the projection SC_{B_i} onto the internal variables. Finaly, if A is lazy, the schema is the same, with $\neg\ SC_A$ as an additional hypothesis.

For instance, let consider the following example, where x is an internal variable of the agent:

$$\begin{array}{ll} I = x \in \mathbb{N} & SC_A = x' = 2x + 2 \\ C_A = true & SC_B = x' = x + 1 \\ & SC_C = x' = 2x \end{array}$$

The proof schema generate the following proof obligation:

$$x \in \mathbb{N} \wedge true \Rightarrow ((x^{tmp} = x + 1 \wedge x' = 2x^{tmp}) \Rightarrow (x' = 2x + 2 \wedge x' \in \mathbb{N}))$$

SyncSeqAnd$_{V_s}$: This operator is a synchronized version of the *SeqAnd* operator with a lock on a subset V_S of V_e. Its proof schema is similar to the *SeqAnd* one, but the projection onto internal variables V_i is replaced by a projection onto $V_i \cup V_s$.

Iter To prove the decomposition $A \Leftarrow Iter(B,\ \mathcal{V})$, a variant $\mathcal{V}$ is needed. It corresponds to the formalisation of the *progress* notion in the resolution of the parent goal. Formally, a *variant* is a decreasing sequence defined on a well-founded structure. A *well-founded* structure is an ordered set such that each strictly decreasing sequence defined on this set has a lower bound. In the following, we will denote the variant lower bound by $\mathcal{V}_0$. Thus proving that $A \Leftarrow IterB$ is correct requires proving that:

- if $\mathcal{V}$ reaches its lower bound, then A is achieved:
$$I \wedge (C_A \vee C_B) \wedge T(SC_B) \Rightarrow (T(\mathcal{V}) = \mathcal{V}_0 \Rightarrow T(SC_A)) \tag{4}$$
- C_B is stable during the loop until A is achieved:
$$I \wedge (C_A \vee C_B) \wedge T(SC_B \vee GPF_B) \wedge \neg T(SC_A) \Rightarrow T(C_B) \tag{5}$$
- the variant decreases: this may be proved whatever the success of B by proving:
$$I \wedge (C_A \vee C_B) \wedge \neg T(SC_A) \Rightarrow (T(SC_B \vee GPF_B) \Rightarrow T(\mathcal{V}) < \mathcal{V}) \tag{6}$$
thanks to this last proof schema, the termination of A is guaranted, if B succeeds one or more time, or even if B never succeeds.

External Goals The proof schema associated to external goals is quite different from the other ones. Let EG_A be an external goal of an agent A associated to an external property P and referencing a goal G_B of another agent B.

The proof consists in showing that:

- G_B is NS;
- achievement of G_B entails achievement of EG_A;
- when A waits for the achievement of EG_A, B will eventually achieve G_B.

The first item is a syntactic and trivial verification. The second one amounts to proving:

$$\left.\begin{array}{l} I_S \wedge I_A \wedge I_B \wedge C_{EG_A} \\ C_{G_B} \wedge P \end{array}\right\} \Rightarrow ([v := v_0]T(SC_{G_B}) \Rightarrow [v := v_1]T(SC_{EG_A})) \tag{7}$$

where substitutions $[v := v_0]$ and $[v := v_1]$, replacing non-primed variables with free variables, allow to memorize the state of the system before the execution of goals EG_A and G_B.

Finally, the third verification is divided into two steps: identifying the set of goals S_B whose contexts are consistent with C_{EG_A} (and checking G_B is in this set) and then verifying for each trace corresponding to a behaviour of B that each time a state of B corresponds to the achievement of a goal in S_B then another state in the future corresponds to the achievement of G_B. The formalisation of this part of the proof schema is not detailed in this paper because it would require to expose the semantics of our operators in temporal logic, which is quite too long to be exposed here.

4 Application

In [2], a scenario with two agents that must collaborate is described. An implementation using Agentspeak and a verification based on model-checking are proposed. This case study has not been chosen to prove the applicability of our model to a real case, but to allow a comparison with another agent specification language and verification system. Using Bordini *et al.*'s description as a specification, we designed GDT models for the two agents of this scenario. In the following, some highlights of these GDTs and the associated proofs are presented. Finally, a comparison with the work exposed in [2] is detailed.

4.1 The scenario

Agentspeak The Agentspeak(L) language has been designed by Rao [16]. The goal of Agentspeak is to express the behaviour of BDI agents. An Agentspeak agent has a base of goals, a base of beliefs and a set of plans. A plan, in Agentspeak, is represented by a rule made of three parts: a triggering event (a goal or belief insertion or deletion), a context, and a list of actions. Agentspeak(L) allows to describe agents in a quite implementable way, but is not suitable to perform proofs by theorem proving for many reasons. For instance, goals are not described formally and most actions have to be implemented directly in the target language (Java for instance).

The Robots on Mars (RoM) scenario The RoM scenario involves two robots that must remove garbage on Mars. Mars is represented by a rectangular grid on which pieces of garbage are randomly distributed. Each robot has different skills.

The first one, R1, moves on the grid to search for pieces of garbage. It can grab them only one by one. When it finds one, it picks it up (in at most three attempts), brings it to the position of R2, then drops it and finally goes back to its previous location and resumes its search. The second robot, R2, cannot move: it can only burn a piece of garbage situated in its cell. Of course, R1 does not grab garbage that are on R2's cell.

R1's behaviour can be summarized as follows (corresponding plans in the Agentspeak implementation are given):

1. it checks its position for a piece of garbage, if there is nothing, it goes to the next slot (plan p1);
2. otherwise (plan p2 to p7):
   (a) it tries to grab this garbage at most three times,
   (b) it brings the garbage to R2 and drops it,
   (c) it goes back and repeats (1).

For instance, plan p1 is the following:
`+pos(R1,X1,Y1):checking(slots) & not(garbage(r1)) ← next(slot).`

The behaviour of R2 is the following:

1. it waits for a piece of garbage to be in its cell,
2. it takes the new piece of garbage,
3. it burns it and repeats (1).

4.2 GDTs for the RoM scenario

Add-ons to the initial specification In [2], a few parts of the robots behaviour were unspecified or under-specified. So we had to make the following choices:

Garbage distribution In Bordini *et al.*'s work, it is not specified whether each cell of the grid can contain at most one or more pieces of garbage. We decided that there is at most one piece of garbage in each cell.

Grabbing success The informal specification states that a piece of garbage is grabbed by R1 in at most 3 attempts, but this is not explicit in the Agentspeak model. We made it explicit in our GDT.

Grid exploration In [2], R1 explores the grid thanks to the `next(slot)` action, which is not specified. We chose to provide R1 only with actions allowing it to move one cell horizontally or vertically. This led us to specify its behaviour when moving, included for avoiding R2's cell (we chose to make it go through the grid row by row, from top to bottom, from left to right on odd lines and from right to left on even lines). Moreover, in [2], `next(slot)` seems to always succeed, but the action performed when R1 reaches the end of the grid is not specified. In the GDT presented here, R1 stops. We designed another GDT where R1 goes back to the first cell, but it is not presented here.

Synchronisation between R1 and R2 Since we specified that a cell cannot contain more than one piece of garbage, R1 cannot drop a new piece of garbage on R2's cell if R2 has not picked up the previous one yet. This synchronisation is not specified in [2]. Moreover, R2 is satisfied (and so cannot act before R1 has dropped a piece of garbage in its cell). So, there are two synchronisations:

- R1 waits for R2 to pick up the piece of garbage.
- R2 waits for R1 to drop a piece of garbage,

The first one is specified by an external goal in the GDT of R1 whereas the second one is specified thanks to the triggering context of the GDT of R2.

The environment The environment is described by a variable G. $G(x, y)$ is true if there is a piece of garbage in the cell at position (x, y) and false otherwise. R2's position, which is constant, is also described by two environment variables x_{R2} and y_{R2}, and so are the minimum and maximum coordinates of the grid (variables $x_{min}, x_{max}, y_{min}, y_{max}$).

Robot R1 The goal of this robot is to clean the grid. To ensure it, it uses a variable named *clean*, which is a set of cells. This set is initially empty, and a cell can be added to it only by the action of picking a garbage or when it is observed to be clean. R1's main goal is $MG_{R1} = (clean = grid)$, where *grid* is the set of all cells on the grid except R2's.

R1 also has variables x and y describing its position on the grid. Its other variables are not presented here.

To design the failure possibility of the arm grabing the garbage, we used an *Iter* operator where the subgoal describing the attempts is a NNS leaf one, so it can fail. In the proof, we show that the parent goal is achieved after, at most, three iterations.

Robot R2 The GDT of R2 is simpler than R1's one. Its GDT correponds exactly to the behaviour described in [2]. It just picks up the garbage and burns it. So, its main goal is to be non busy and to have its cell clear: the satisfaction condition of its main goal is $SC_{MG_{R2}} = (\neg busy_{R2} \wedge \neg G(x_{R2}, y_{R2}))$. The synchronisation needed to ensure that R2 waits for a piece of garbage to burn is not directly expressed in the structure of the GDT but thanks to its triggering context $TC(R2) = G(x_{R2}, y_{R2})$. In fact, the GDT will be executed if and only if there is a piece of garbage at the position of R2. Let us notice that since R2 must not be busy before each of its executions, the property $\neg busy_{R2}$ is in the $PrecGDT_{R2}$ property.

4.3 Examples of proofs

We now present three detailed local proofs of nodes in R1's GDT with, for each one, an informal description and the subtree associated to the parent goal. The full proof of the two GDTs can be found at [14].

SyncSeqAnd example We first consider the part of the GDT where R1 drops a piece of garbage onto R2's cell (Figure 1 (a), where the rectangle denotes an external goal). R1 first waits for R2's cell to be empty (external goal B, "Empty cell", detailed below), then drops the piece of garbage it holds (leaf goal C, "Drop"). Variable $G(x, y)$ is synchronized in order to ensure that once R1 has observed the cell is empty, it stays so until R1 drops it garbage.

We have:

$$\begin{cases} C_A = (x, y) = (x_{R_2}, y_{R_2}) \wedge busy \\ SC_A = \neg busy' \wedge G'(x', y') \wedge (x', y') = (x, y) \\ SC_B = \neg G(x, y) \wedge busy \\ SC_C = \neg busy' \wedge G'(x', y') \end{cases}$$

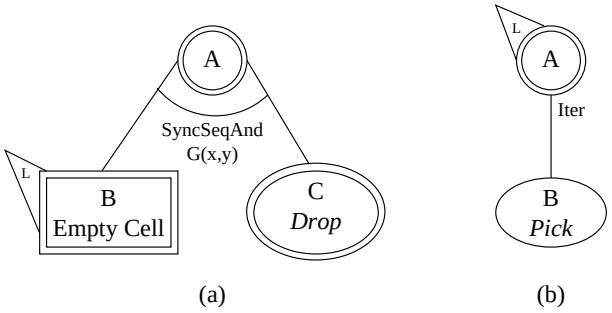

Fig. 1. Subtrees of R1's GDT

Moreover, the parent Goal A is NL and NS. So, to prove the decomposition of A, according to the schema in Section 3 we have to prove:

$$I \wedge C_A \Rightarrow [v' := v^{tmp}]T(SC_{B_{i,G(x,y)}}) \wedge [v := v^{tmp}]T(SC_C) \Rightarrow T(SC_A)$$

That can be rewritten:

$$H \Rightarrow (\neg busy' \wedge G'(x', y') \wedge (x', y') = (x, y))$$

Conjuncts $\neg busy'$ and $G'(x', y')$ are entailed directly by $[v = v^{tmp}]T(SC_C)$. Now since variables x, y are internal and do not occur in SC_B and SC_C, we have $(x', y') = (x, y)$ and finally, $T(SC_A)$. Let Observe that synchronisation of $G(x, y)$ is not used in this proof; it is used only in the context propagation, for ensuring that the context of C entails $\neg G(x, y)$ as established by SC_B.

Iter example We now consider the part of the GDT where R1 tries to pick up the piece of garbage on the current cell until success (Figure 1 (b)). R1 iterates over subgoal B, "Pick", which consists in applying the named action `pick` (recall that this action may fail, but succeeds after at most three attempts).

Since this subtree ends up with picking a piece of garbage, variable *clean* (set of positions R1 has cleaned or observed to be clean) is involved. In order to simplify the presentation, we however chose to remove it from the contexts and satisfaction conditions here, as well as condition $(x, y) \neq (x_{R_2}, y_{R_2})$, which is stable in this subtree.

The variant used in the proof involves variable *nbAttempts*; this variable counts the number of times R1 has already tried to pick up the piece of garbage. Moreover, the Guaranteed Property upon Failure (GPF) of an NNS goal is a formula which is true when the goal fails.

We have:

$$\begin{cases} C_A = G(x,y) \wedge \neg busy \wedge nbAttempts = 0 \\ C_B = G(x,y) \wedge \neg busy \wedge nbAttempts < 3 \\ SC_B = \neg G'(x',y') \wedge busy' \\ GPF_B = \begin{cases} G'(x',y') \wedge busy' = busy \\ \wedge\ nbAttempts' = nbAttempts + 1 \\ \wedge\ nbAttempts < 3 \end{cases} \end{cases}$$

Moreover, the parent node is lazy and NS, and its satisfaction condition is $SC_A = \neg G'(x',y') \wedge busy' = SC_B$.

Let $\mathcal{V} = (3 - nbAttempts)$ be the variant and let its lower bound be $\mathcal{V}_0 = 0$. $\mathcal{V}$ is well-defined because the invariant I of R1 entails $nbAttempts \leq 3$. According to the schema in Section 3, we have to prove:

$$\left.\begin{array}{l} I \wedge (C_A \vee C_B) \\ \neg SC_A \end{array}\right\} \Rightarrow \begin{cases} T(SC_B) \Rightarrow T(\mathcal{V}) = \mathcal{V}_0 \Rightarrow T(SC_A) & (1) \\ \neg T(SC_A) \Rightarrow (T(SC_B) \vee T(GPF_B)) \Rightarrow T(\mathcal{V}) < \mathcal{V} & (2) \\ \neg T(SC_A) \Rightarrow (T(SC_B) \vee T(GPF_B)) \Rightarrow T(C_B) & (3) \end{cases}$$

Entailment (1) is obvious since $T(SC_B) = T(SC_A)$.

Entailment (2) holds when $T(SC_B)$ is true because in that case, $\neg T(SC_A)$ is false (as SC_A=SC_B). It also holds with $T(GPF_B)$ because $T(GPF_B)$ entails that $nbAttempts' = nbAttempts + 1$, which in turn entails $T(\mathcal{V}) < \mathcal{V}$ (as $\mathcal{V} = 3 - nbAttempts$ and $T(\mathcal{V}) = 3 - nbAttempts'$).

Entailment (3) holds when $T(SC_B)$ for the same reason as entailment (2). Finally, it also holds with $T(GPF_B)$ because $T(GPF_B)$ together with either C_A or C_B clearly entails $T(C_B)$.

External goal example We finally consider the external node B on Figure 1 (a) in the GDT where R1 waits for R2's cell to be empty, in order to be allowed to drop the piece of garbage it holds onto it. Since R1 cannot empty this cell itself, it must wait for R2 to do it.

The goal of R2 achieving R1's desire is its main goal MG_{R2}. This goal is decomposed into two subgoals using a *SyncSeqAnd* operator, namely goal "Pick" and goal "Burn". R2 has an internal variable $busy_{R_2}$ which is true if R_2 currently holds a piece of garbage and false otherwise.

As a consequence, we have:

$$\begin{cases} C_B = (x,y) = (x_{R_2}, y_{R_2}) \wedge busy \\ SC_{MG_{R2}} = \neg busy_{R_2} \wedge \neg G(x_{R_2}, y_{R_2}) \\ SC_B = \neg G(x,y) \wedge busy \end{cases}$$

According to the proof schema in Section 3, we first have to check that R2's goal MG_{R2} is NS, which is the case.

Now we have to check that the achievement of MG_{R2} entails the achievement of R1's goal B by applying proof schema 7, that can be here approximatively simplified in $C_B \wedge T(SC_{MG_{R2}}) \Rightarrow T(SC_B)$. Again, this is true since:

- $busy'$ in $T(SC_B)$ is entailed by $busy$ in C_B together with the fact that $busy$ being an internal variable of R1, $busy' = busy$,
- $\neg G'(x', y')$ in $T(SC_B)$ is entailed by $\neg G'(x_{R_2}, y_{R_2})$ in $T(SC_{MG_{R2}})$ together with $(x, y) = (x_{R_2}, y_{R_2})$ in C_B and the fact that x, y are internal variables of R1 and x_{R_2}, y_{R_2} are constants.

Finally, we have to check that every state of R2 which is compatible with C_B finally ends up with the achievement of MG_{R2}. When R1 waits for B to be satisfied, as B is lazy, we have:

$$C_B \wedge \neg SC_B$$

that is to say:

$$(x, y) = (x_{R2}, y_{R2}) \wedge busy \wedge \neg(\neg G(x, y) \wedge busy)$$

that can be rewritten:

$$(x, y) = (x_{R2}, y_{R2}) \wedge busy \wedge (G(x, y) \vee \neg busy)$$

that can be simplified:

$$(x, y) = (x_{R2}, y_{R2}) \wedge busy \wedge G(x_{R2}, y_{R2}) \tag{8}$$

As a consequence, the context of the *Burn* node of R2 entails $\neg G(x_{R2}, y_{R2})$, which is not compatible with equation 8. So the set of compatibles goals of R2 with B is $\{Pick, MG_{R2}\}$. If R2 is executing goal MG_{R2}, this goal is NS, and so will be achieved, implying that SC_B will be true. If R2 is executing goal $Pick$, as this goal is NS and is followed (thanks to a $SeqAnd$ operator) by another NS goal (Burn), the parent goal (which is MG_{R2}) will also be satisfied and so will be B.

Finally, we have to considered what happens if R2's GDT execution is ended. Recall that the triggering context of R2 defined in section 4.2 is $TC(R2) = G(x_{R2}, y_{R2})$. this is obviously entailed by equation 8 above. And so, R2, will reach the execution of its main goal, which satisfies goal B.

4.4 Comparison with Bordini et al.'s work

Goal decomposition The body of an Agentspeak plan is "*a sequence of basic actions or (sub)goals that the agent has to achieve (or test) when the plan is triggered*". There are two kinds of such subgoals: the first kind must be achieved by other plans whereas the second one consists in beliefs additions or deletions. In a GDT, the first kind is specified by a goal decomposition and the second one corresponds to variable modifications inside leaf goals. So, there is a similarity between our goal decompositions and Agentspeak plans. However, as shown in the next section, the verification of behaviours is completely different.

Verification and proof Verification in AgentSpeak is based on model-checking which takes place after the implementation step with JPF2. In [2], it has been made on an instance of this scenario where the size of the grid is 5x5 and with only 2 pieces of garbage. On the contrary, our proof is performed only once for all instances of this scenario without any constraint on the number of pieces of garbage, on the size of the grid and on the position of R2. Of course, when working with first-order logic, theorem proving is not decidable, leading some true properties unproved whereas model-checking can be automatically performed with a computable complexity. However, to obtain the same level of proof as ours on the RoM problem, it would be necessary to test an infinite number of situations, because the grid can be of any size.

Another interesting aspect of our proof process is that it helped us to find some problems in our first designs of GDTs before their implementation. Indeed, proof failures give local clues to solve inconsistencies or to highlight lacks in the specification. In that case, the compositional aspect of our proof process implies that required modifications of GDTs do not make the whole proof fail but only parts associated to goals involved in this modification. For instance, the presence of $\neg busy_{R2}$ in $PrecGDT_{R2}$ was not specified in a first version, generating a proof failure when R2 had to pick up a piece of garbage (he might have already been busy). Modifying this property also implied to modify the `init` clause of R2 to avoid a new proof failure.

Finally, model-checking performed on the RoM scenario can only be done on a complete implementation. For instance, an implementation of next(slot) must be provided in order to verify properties presented in [2]. Thanks to the compositional property of our proof process, the proof of the correctness of the behaviour of R1 can be made in two steps. In a first step, the correctness of the behaviour of R1 can be obtained under the assumption that *next_cell*, the goal corresponding to the $next(slot)$ basic action, provides a behaviour ensuring that R1 moves to a never visited cell different from R2's. To do this, *next_cell* was specified by a goal with only a satisfaction condition but no subtree. In a second step, we have specified the *next_cell* goal by a subtree. The proof process has then allowed to prove the correctness of the subtree with respect to the *next_cell* satisfaction condition.

Expressiveness and conciseness Since satisfaction conditions used in a GDT are based on sets theory, arithmetics and temporal logic, they are at least as expressive as Agentspeak and they allow to completely specify behaviours of BDI agents, the base of beliefs being represented by the set of the variables of the agent. For instance, we did not find any lack of expressiveness when we applied the model to the RoM scenario.

Another interesting comparison deals with the conciseness of Agentspeak and GDT. At first glance, Agentspeak seems quite more concise: the Agentspeak model for robot R1 is made of 9 plans whereas the GDT of R1 contains 30 nodes. However, the Agentspeak model uses unspecified actions (drop, grab, nextslot, etc.) that we fully specified in our GDT (since we wanted an automatic translation to an implementation). If these *implementation details* are removed

from the GDT, its size becomes equal to 13 nodes with at most 2 subgoals each, which is comparable to the number of plans in the Agentspeak model, where each plan has up to 3 subgoals.

Finally, we wish to emphasize that our model allows to take into account the semantics of the actions, thanks to preconditions and postconditions. Every modification of a variable corresponding to an actuator which is used in the agent's goal can only be done through named actions.

5 Related Works

The GDT model of an agent behaviour and the associated proof system are differently connected to several different kinds of works. First of all, there are links with formal agent models like MetateM [9], Desire [4] or Gaïa [22]. These works are focused on agent models on which it is possible to reason which is necessary for analysis and especially for proof problems. A part of these formal models like [21, 8, 20] are focused on a declarative description of goals, which is exactly our point of view. A detailed comparison of these approaches with the GDT model can be found in [19]. There are also links between our proposal and agent programming languages like AgentSpeak [16], 3APL [7], ConGolog [10]. These languages allow to specify agents behaviours which can be directly executed which is one of the goals of the GDT model. However, 3APL does not allow to prove the specified behaviour and ConGolog is dedicated to situation calculus. Our approach can also be compared to goal oriented MAS development methods like Prometheus [12], MaSe [6], KAOS or Tropos. Indeed, our proposal is also intended to provide a complete MAS design process from the specification to the implementation. Moreover our approach takes place in the framework of "formal transformation systems" as defined in [6]. A detailed comparison of these works with the GDT model can be found in [19]. Last but not least, our proposal can be directly compared to SMA verification methods. Two subtypes can be distinguished: theorem proving based (like in PROSOCS) and model checking.

PROSOCS [3] agents are agents whose behaviour is described by goal decomposition rules *à la* Prolog. Rules are parameterised by time variables allowing to perform proofs about the evolution of the system state. Many characteristics of PROSOCS agents are very interesting for performing proofs, and a proof procedure has been implemented in Prolog. However, the system is limited to propositional logic formulas. The Goal [8] method also has a proof model, but is limited to propositional logic too.

Model checking is a verification method consisting in testing all the situations which may be encountered by the system. Two kinds of model checking can be distinguished: bounded model-checking [2] and unbounded-model checking [1, 15, 11]. However, with the two types of model-checking, proofs can only be performed on finite models or on models that can be considered as finite ones.

6 Conclusion

We have demonstrated the GDT model on an already studied example, and shown that it is as interesting as Agentspeak in terms of expressiveness and

conciseness, but also allows to prove behaviours (as opposed to model checking or no verification at all).

Arguably, GDT are graphically rather complex to manipulate. This is why we have created an application, called `GdtEditor`, which allows to edit GDTs and all related information (satisfaction conditions, variables, actions...), and to export them in various formats. The application also automatically generates the implementation model (in Java), as is allowed by the model. This application is available at [17]. Current work aims at integrating it a theorem prover. Once the substitutions applied, our proof schemas generate proof obligations similar to those of the B method. As a consequence, using a prover of this method like krt [5] should be straightforward. Preliminary tests confirm this. An automatic connexion with this prover should be presented in a future article. A small percentage of the proofs of true properties may fail, but krt provides an interactive mode that allows to help the prover in these proofs.

Current work on the method aims at generalizing the model of external goals in order to allow specification and proof of more interactions, in particular when several other agents are needed to achieve an external goal. We are also working on parameterizing GDTs together with their proofs, so as to be able to factorize similar subtrees or more generally to reuse behaviours.

References

1. N. Alechina, B. Logan, and M. Whitsey. A complete and decidable logic for resource-bounded agents. In *Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, 2004.
2. R.H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifiable multi-agent programs. In M. Dastani, J. Dix, and A. Seghrouchni, editors, *ProMAS*, 2003.
3. A. Bracciali, U. Endriss, N. Demetriou, T. Kakas, W. Lu, and K. Stathis. Crafting the mind of prosocs agents. *Best of 'From Agent Theory to Agent Implementation 4', Applied Artificial Intelligence*, to appear, 2004.
4. F.M.T. Brazier, P.A.T. van Eck, and J. Treur. *Simulating Social Phenomena*, volume 456, chapter Modelling a Society of Simple Agents : from Conceptual Specification to Experimentation, pages pp 103–109. Lecture NOtes in Economics and Mathematical Systems, 1997.
5. Clear-Sy. B for free. http://www.b4free.com/public/resources.php.
6. Scott A. Deloach Clint H. Sparkman and Athie L. Self. Automated derivation of complex agent architectures from analysis specifications. In *Proceedings of AOSE'01*, 2001.
7. M. Dastani, F. de Boer, F. Dignum, and J.-J. Meyer. Programming agent deliberation: An approach illustrated using the 3apl language. In *Proceedings of the Second International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'03)*, 2003.
8. F.S. de Boer, K.V. Hindriks, W. van der Hoek, and J.-J.Ch. Meyer. Agent programming with declarative goals. In *7th International Workshop on Intelligent Agents. Agent Theories Architectures and Language*, pages 228–243, 2000.
9. M. Fisher. A survey of concurrent METATEM – the language and its applications. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic - Proceedings of*

the First Intemational Conference (LNAI Volume 827), pages 480–505. Springer-Verlag: Heidelberg, Germany, 1994.
10. Giuseppe De Giacomo, Yves Lesperance, and Hector J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.
11. M. Kacprzak, A. Lomuscio, and W. Penczek. Verification of multiagent systems via unbounded model checking. In *Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, 2004.
12. J. Khallouf and M. Winikoff. Towards goal-oriented design of agent systems. In *Proceedings of ISEAT'05*, 2005.
13. B. Mermet, D. Fournier, and G. Simon. An agent compositional proof system. In *From Agent Theory to Agent Implementation (AT2AI'06)*, 2006.
14. B. Mermet, G. Simon, A. Saval, and B. Zanuttini. GDTs and Proofs for Robots on Mars. Technical report, GREYC, 2006. http://scott.univ-lehavre.fr/~mermet/GDT/applications/proofRoM.pdf.
15. F. Raimondi and A. Lomuscio. Verification of multiagent systems via orderd binary decision diagrams: an algorithm and its implementation. In *Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, 2004.
16. A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde and J. Perram, editors, *MAAMAW'96*, volume 1038, Eindhoven, The Netherlands, 1996. LNAI.
17. A. Saval. Robots on mars: implementation, 2006. http://arnaud.saval.free.fr/backup/applet/page.html.
18. G. Simon and M. Flouret. Implementing validated agents behaviours with automata base on goal decomposition trees. In *Agent Oriented Software Engineering VI*, volume 3950 of *LNCS*, pages 124–138. Springer Verlag, 2006.
19. G. Simon, B. Mermet, and D. Fournier. Goal decomposition tree: An agent model to generate a validated agent behaviour. In Andrea Omicini Paolo Torroni Matteo Baldoni, Ulle Endriss, editor, *Declarative Agent Languages and Technologies III: Third International Workshop, DALT 2005*, volume 3904 of *LNCS*, pages 124–140. Springer Verlag, 2006.
20. M.B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J.Ch. Meyer. Dynamics of declarative goals in agent programming. In *Proceedings of Declarative Agent Languages and Technologies (DALT'04)*, 2004.
21. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative & procedural goals in intelligent agent systems. In *8th International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, 2003.
22. M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.