GDT4MAS: an extension of the GDT model to specify and to verify MultiAgent Systems

Bruno Mermet, Gaële Simon

GDT4MAS: an extension of the GDT model to specify and to verify MultiAgent Systems

Bruno Mermet
GREYC - UMR CNRS 6072
Campus Côte de Nacre
BP 5186
14032 Caen Cedex
Bruno.Mermet@univ-lehavre.fr

Gaële Simon
GREYC - UMR CNRS 6072
Campus Côte de Nacre
BP 5186
14032 Caen Cedex
gsimon@iut.univ-lehavre.fr

ABSTRACT

The Goal Decomposition Tree model has been introduced in 2005 by Mermet *et al.* [9] to specify and verify the behaviour of an agent evolving in a dynamic environment. This model presents many interesting characteristics such as its compositional aspect and the definition of proven proof schemas making the proof mechanism reliable. Being interested in specifying and verifying multiagent systems, we have decided to extend the GDT model for specifying Multiagent systems. The object of this article is to present this extension. So, after a brief description of the initial GDT model, we show how we extend it by introducing the specification of the whole MAS. We also introduce the notions of agent type and agent, and we show how external goals allow to specify collaborative agents and to prove the correctness of their collaboration. These notions are illustrated on a toy example of the litterature.

Categories and Subject Descriptors

D.2.4 [**Software engineering**]: Software/Program Verification—*Theorem proving*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*; D.2.2 [**Software engineering**]: Design tools and techniques—*GDT*

Keywords

Multiagent systems, Specification, Verification, GDTs, Temporal Logic

1. INTRODUCTION

Verifying computer systems is an important field of software engineering. For a few years, several people have tried to adapt verification techniques to the field of agent design. Works on the verification of multiagent systems are few. Actually, most verification techniques on agents use model-checking. When many agents evolve concurrently in a shared environment, the size of the problem becomes rapidly intractable by exhaustive techniques such as model-checking. Some works try to reduce the complexity by increasing the granularity, but as a consequence, the result is no more a proof of the correctness because many states are no longer analysed.

The model of *Goal Decomposition Trees* [9] is a formal model allowing to formally specify agents and providing *proof schemas* to verify the correctness of the specification[1] *in a compositional way*. Moreover a GDT specification can be automatically translated into a program. The existing version of the GDT model deals with the specification of a small number of agents, each one with its own GDT, and with limited proofs possibilities at the system level, and is well-suited to be extended to the specification of multiagent systems because, an agent specified with a GDT is situated in an environment, and this environment can evolve independently from the agent. Thus, we have chosen to extend this model to specify and verify multiagent systems, and this article presents this work.

In the next part, we briefly present the GDT model and its main characteristics. Then, we present the extensions we propose in order to specify and verify multiagent systems. And finally, we illustrate this on an example.

1.1 The ERoM problem

The Robots on Mars problem (RoM problem) was first presented in [2]. In this problem, two robots have to clean Mars. Mars is represented by a grid. One of the robots, R2, cannot move but can burn garbage. The other robot, R1, can move on the grid. When it discovers a piece of garbage in a cell C, it picks it up, brings it to R2, and then restart its exploration of Mars from the cell C. The behaviours of these two robots using GDTs was given in [8].

The Extended Robots on Mars (ERoM) problem is an extension we propose where multiple R2 robots are present on the grid. When R1 finds a piece of garbage, it brings it to the nearest robot R2.

2. THE GDT MODEL

Here, we give a brief description of the GDT model. More details can be found in articles presenting the model like [9].

2.1 The environment

In the GDT model, an agent is situated in an environment. An environment is specified by a set of typed variables and an invariant property on these variables. The value of the

[1] a proof schema is a rule that allows to generate *proof obligations* that, once verified, guaranty the correctness of the system

Cite as: GDT4MAS: an extension of the GDT model to specify and to verify MultiAgent Systems, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

variables of the environment can evolve independently of the actions of the agent. However, the invariant property is always true. The invariant property is expressed in first-order logic, using arithmetic and set-theory operators.

2.2 Agent and Goal Decomposition Tree

In the GDT model, an agent is specified by a set of variables (internal to the agent or belonging to the environment), an invariant property, an initialisation clause of its variables, a set of actions and a behaviour. The behaviour is specified by a Goal Decomposition Tree.

As the goal notion is central when dealing with agents [5], A GDT is a tree of goals, where each non-leaf goal is decomposed, thanks to an operator, into subgoals. The root goal is the main goal of the agent. Each goal is specified by a name and two logical formulae:

- a *Satisfaction Condition (SC)*: the property that is established if the goal solving process (execution) has succeeded;
- a *Guaranted Property in case of Failure (GPF)*: which is established if the goal solving process has failed.

There are two kinds of goals:

- *state* goals: the satisfaction condition of such goals specify a property that should be verified by the state reached after the success of the goal execution, for instance: $x > 10 \wedge y < x$.
- *progress* goals: the satisfaction condition of such goals express a link between the states of the agent before and after the goal execution. In satisfaction conditions of such goals, a primed variable x' corresponds to the value of the variable x after the goal execution whereas an unprimed variable x corresponds to the value of the same variable in the state before the execution of the goal decomposition. An example of the satisfaction condition of such a goal is: $x' = x+1 \wedge y' > y \wedge z' = z$.

Moreover, a goal is specified by two boolean properties (see [9]) for details: its *lazyness* and its *necessary satisfiability*. A lazy goal (L goal) is a goal whose decomposition is executed only if its satisfaction condition is false and a necessarily satisfiable goal (NS goal) is a goal whose decomposition is guaranted to achieve the goal.

A goal can either be linked to an action that should achieve it (leaf goal) or be decomposed into one or more subgoals thanks to decomposition operators (intermediate goals). Among the few operators defined by the authors, there are nondeterministic operators (and, or), sequential operators (seqand, seqor), and an iteration operator (iter).

Notice that rules (temporal logic formulae) are provided with the model to allow to extend this list with new operators if necessary. A detailed description of these operators can be found in [9].

3. EXTENSIONS TO SPECIFY MULTI-AGENT SYSTEMS

3.1 Introduction

We first have to specify what we call a multiagent system. Informally, a multiagent system can be defined as a tuple of two elements: the first one is the environment, and the other one is the population of agents. The following definition of the environment has already be given in [8]:

DEFINITION 1. **Environment** *An* environment *is a triple* $\mathcal{E} = (V_\mathcal{E}, I_\mathcal{E}, s_\mathcal{E})$*, where:*

- $V_\mathcal{E}$ *is the set of environment variables,*
- $I_\mathcal{E}$ *is the invariant of the environment,*
- $s_\mathcal{E}$ *is the set of the* stable properties *of the environment.*

In the previous version of the GDT model, agents and GDTs were not very well distinguished. This is no more suitable as many agents can be of the same type, defined by the same GDT.

3.2 Agent population

When dealing with multiagent systems, a first key point is to be able to formally describe the population of agents. In this article, we will only consider a static population (no birth, no death). Each *agent* is an instance of a *type of agent* which is a key notion of multiagent systems. From the older definition of agents in the GDT model, we give the following definition of a type of agent:

DEFINITION 2. **Agent type(temporary definition)** *Let $\mathcal{E}$ be an environment. A type of agent T is a tuple:*

$$(\texttt{name}_\texttt{t}, V_i, V_\mathcal{E}, init, I, S, Actions, \texttt{Beh}))$$

where:

- $\texttt{name}_\texttt{t}$ *is the name of the type,*
- V_i *is the set of internal variables,*
- $V_\mathcal{E}$ *is the set of environment variables seen,*
- *init is the initialisation of the internal variables,*
- *I is the invariant property,*
- *S is the set of stable properties,*
- *Actions is the set of capabilities,*
- Beh *is the behaviour (namely a GDT).*

NOTATION 1. $\mathcal{T}$
We introduce a constant set $\mathcal{T}$ which contains the types of agents belonging to the system.

NOTATION 2. *If $t = (name_t, V_i, V_\mathcal{E}, init, I, S, Actions, \texttt{Beh})$ is a type, we will write $name_t(t)$ the name of t, $V_i(t)$ the set of internal variables of t, etc. More formally, we consider that name is a function typed by $name \in \mathcal{T} \rightarrow String$, etc.*

The system is made of agents. Each agent is an instance of a type. It means that each agent has its own set of variables. Moreover, even if agents of a given type share a common behaviour, they may differ by the value of some constants. For instance, in the case of the ERoM problem, two robots R2 are on different cells, and so, their positions differ. As a consequence, the behaviour of a type of agent will no more be described by a GDT, but by a parameterized GDT, a notion presented in [8]. So, an agent is specified by an identifier, a type of agents and a list of effective parameters for the parameterized GDT associated to the type.

DEFINITION 3. **Agent** *An agent a is defined by a tuple* $(name, type, param)$ *where:*

- *name is the name of the agent;*
- *type is the type of the agent;*
- *param is the list of effective parameters for the GDT of the type of the agent.*

If $a = (name_a, type_a, param_a)$ is an agent, we will write $name(a)$ its name ($name_a$), $type(a)$ its type and $param(a)$ its list of parameters.

NOTATION 3. **$\mathcal{A}$ and $\mathcal{A}_T$** *We introduce a constant called $\mathcal{A}$which is the set of the agents defined in the system.*

Let $t \in \mathcal{T}$ be a type of agent. We write $\mathcal{A}_t$ the set of agents of type t. Formally, $\mathcal{A}_t = \{a.(a \in \mathcal{A} \ \wedge type(a) = t)\}$.

EXAMPLE 1. *In the case of the ERoM problem, where one instance of R1 exists and two instances of R2 are present in the cells (x_1, y_1) and (x_2, y_2) (where x_i and y_i are constants), the agents of the system will be declared as:*

$\mathcal{A} = \{ \ (r1, R1, \{\})(r2a, R2, (x_1, y_1))(r2b, R2, (x_2, y_2)) \ \}$

3.3 Surface variables

3.3.1 Definition

In the previous version of the GDT model, there was two kind of variables:

- internal variables: these variables belong to an agent and cannot be seen by the others;
- environment variables: these variables are shared by all the agents. So, each agent can see them but also, each agent can modify them.

As a consequence, an agent can not have any information on the others, whereas it is often necessary when agents have to collaborate. So, it seemed necessary to introduce variables that can be seen by all the agents and modified by only one agent, the *owner agent* of this variable. Such variables can be considered as follows:

- from the point of view of the owner agent, these variables can be considered as internal variables, as the agent can perform any action on them and no other agent can modify their values;
- from the point of view of the other agents (called in the sequel *observer agents*), these variables can be partially considered as environment variables, as their value can be seen and can be modified by other agents (namely by the owner agent). But contrary to environment variables, the observer agents cannot modify their values. Notice that these variables **are not** environment variables but they can be considered as environment variables from a proof point of view.

DEFINITION 4. **Surface Variable**

A surface variable is an internal variable of an agent called the owner agent. The owner agent can modify at will the value of this variable. The other agents can see the value of this variable but cannot modify it.

3.3.2 Representation

As surface variables have, from the point of view of the owner agent, the same status as internal variables, they are represented in its GDT as internal variables. However, the definition of a type of agent is modified as follows (this new definition replaces definition 2):

DEFINITION 5. **Agent type (version 2)** *In addition to the previous definition, an agent type is also specified by V_s, the set of surface variables of this type and $V_s \subseteq V_i$.*

A surface variable must also be seen by observer agents. Moreover, it may be used in environment properties. So, as each agent of a given type must have its own surface variables, each surface variable can be represented into the other agents by a functional variable as follows:

NOTATION 4. **External representation of a surface variable**

Let T be a type of agent and vs one of its surface variables. We call t_{vs} the type of this variable. This variable can be represented in the other GDTs or in the invariant of the environment by a pseudo-variable $vs^S_T \in Agents_T \rightarrow t_{vs}$. So, the value of vs for an agent a of type T can be expressed in the other agents by $vs^S_T(a)$.

Notice that, as there are constants belonging to the environment, it is also possible to declare *surface constants*[2].

EXAMPLE 2. *In the RoM problem, the Robot R_2 was specified by three variables: x_{R_2}, y_{R_2} and busy. As its position is constant and must be known by R_1, in the solution given in [8], x_{R_2} and y_{R_2} were defined as environment constants, whereas busy (specifying whether R_2 is busy or not) was specifed as an internal variable of R_2.*

In the extended version, Robots R_2 still do not move, but each one has its own position. So, x_{R_2} and y_{R_2} are now surface constants.

Notice that we can specify the property that robots R2 are on different cells by the following environment invariant:

$$\left\{(a,(x,y)) \,.\, \left(x^S_{R_2}(a) = x \wedge y^S_{R_2}(a) = y\right)\right\} \in Agents_{R_2} \rightarrowtail (T_X \times T_Y)$$

where:

- *T_X, T_Y are the types of the constant x_{R_2}, y_{R2};*
- *$\rightarrowtail$ is the symbol of the set theory to denote injection.*

3.4 Multiagent System

From the previous considerations, we can now give a formal definition of a multiagent system in our model:

DEFINITION 6. **Multiagent System** *A multiagent system $\mathcal{MS}$ is a tuple $\mathcal{MS} = (\mathcal{E}, \mathcal{T}, \mathcal{A})$ where:*

- *$\mathcal{E}$ is an environment,*
- *$\mathcal{T}$ is the set of the agent types in the system,*
- *$\mathcal{A}$ is the set of the the agents of the system.*

3.5 A new specification for External Goals

In a previous paper [8], the notion of external goals was added to the GDT model. Informally, an external goal is a goal that an agent a_1 cannot achieve and that must be achieved by another agent a_2. When a_1 needs this goal to be achieved, it waits until is has been achieved. An example of such a goal is given in example 3.

EXAMPLE 3. **external goal** *Let us consider two agents in a university where classrooms are locked for security reasons. The first agent, the* teacher*, has a course to provide in a classroom. The second agent is the* mace bearer*: it has a pass key to open classrooms. Then a subgoal of the teacher agent is to enter into the right classroom. This subgoal is decomposed thanks to a SeqAnd operator into two subgoals: Unlocking the door, then opening it. So, the first subgoal is an external goal that will be achieved by the mace bearer, the second agent.*

The formal definition of an external goal was the following:

[2]In this article, to reduce the length of the formulae, environment and surface constants are not represented.

DEFINITION 7. **external goal (previous definition)** *Let a be an agent. An* external goal *in the GDT of a is a NS goal* ex *of the form $(name_{ex}, SC_n, ln_n, ea, ea_g)$*

where $name_{ex}$ is the name of the goal, SC_n is the satisfaction condition of the goal, ln_n is the lazyness of the goal, ea is another agent in $\mathcal{E}$, and ea_g is an NS goal in the GDT of ea.

This definition must first be modified as GDTs are not more associated to agents but to agent types. But we also want to change this definition because it violates the compositional aspect of the GDT model: actually, with such a definition, a GDT with an external goal has to know the structure of another GDT (the GDT of the agent ea). So, we do not want to always have to precise for an external goal which node of which agent will achieve the external goal. As a consequence, the new definition of an *external goal* is the following:

DEFINITION 8. **external goal**
Let a be an agent in an environment $\mathcal{E}$. An external goal *in the GDT of a is a NS leaf goal with no action attached to it and declared as being external.*

Before describing the new proof procedure, we have to recall to the reader what a *leads-to* property is.

DEFINITION 9. **leads-to property** *In the temporal logic domain, we define a* leads-to *property as: a $\texttt{leads-to}$ $b \equiv \Box(a \rightarrow \diamond b)$. In the sequel, a (respectively b) will be called the* antecedent *(respectively the* subsequent*).*

The proof procedure of external goals has also to be modified. Recall that each goal g in a GDT has a context C_g[3]. So, if ex is an external goal, with a context C_{ex} and a satisfaction condition SC_{ex}, we have to prove that in the system, the following *leads-to* property is true:

$$\Box(C_{ex}^{env} \rightarrow \diamond SC_{ex}^{env})$$

where F^{env} is the projection of the formula F on environment variables.

First of all, notice the following property of the Linear Temporal Logic:

$$\frac{a \rightarrow b, c \rightarrow d}{\Box(b \rightarrow \diamond c) \vdash \Box(a \rightarrow \diamond d)}$$

As a consequence, to verify that an external goal will eventually be achieved, it has to be shown that there exists in the MAS an agent that satisfies a property $\Box(b \rightarrow \diamond c)$ with: $C_{ex}^{env} \rightarrow b$ and $c \rightarrow SC_{ex}^{env}$.

In order to maintain the compositional aspect our model, it implies that a GDT must specify which leads-to properties it satisfies. As GDTs are attached to types, it is in fact the definition of Agent types (definition 5) that as to be modified as follows:

DEFINITION 10. **Agent type (version 3)**
Let $\mathcal{E}$ an environment. A type of agent T is a tuple:
$(\texttt{name}_\texttt{t}, V_i, V_s, V_\mathcal{E}, init, I, \mathbf{L}, S, Actions, \texttt{Beh}))$

Where:
- $\texttt{name}_\texttt{t}$ *is the name of the type,*
- V_i *is the set of internal variables,*
- V_s *is the set of surface variables of this type and we have $V_s \subset V_i$.*
- $V_\mathcal{E}$ *is the of environment variables seen,*
- *init is the initialisation of the internal variables,*
- I *is the invariant property,*
- **L is the set of leads-to properties** *relying on environment and surface variables,*
- S *is the set of stable properties,*
- *Actions is the set of capabilities,*
- $\texttt{Beh}$ *is the behaviour (namely a GDT).*

[3]This context, which is automatically inferred from the GDT structure, allows to make compositional proofs

3.6 Proof of external goals

From the explainations above, the proof process of an external goal is the following:

Let ex be an external goal with a satisfaction condition SC_{ex} and a context C_{ex}. We have to prove, possibly by a proof by case, that there exists in the system an agent with a *leads-to* property $\Box(b \rightarrow \diamond c)$ with $C_{ex} \rightarrow b$ and $c \rightarrow SC_{ex}$.

3.7 Proof of leads-to properties

In [8], where a goal g of an external agent ea was associated to each external goal ex, a proof procedure was given to prove that ea was achieving SC_{ex} when required. This proof procedure can be slightly modified to verify a leads-to property a $\texttt{leads-to}$ b of an agent ag:

1. determine the set of goals S_1 of ag whose context is compatible with a
2. determine the set of NS goals S_2 of ag whose satisfaction condition implies b,
3. show that the execution of each goal s of S_1 leads-to the execution of a goal of S_2 and that a implies the triggering context of th GDT.

Notice that verifying that there is not dead-lock has been removed. Actually, this step cannot be performed on a GDT independently of the other agents in the system. The easier way to guarantee this essential property is to ensure that, in the execution of the GDT, from a goal of S_1 to a goal of S_2, the execution process does not depend on other agents (with external goals). This constraint is quite strong. It can be relaxed by introducing a verification process when the MAS is completely defined. However, this process cannot be presented in this paper for space reasons.

4. TOY EXAMPLE

In this section, we show how the ERoM problem can be specified with the GDT4MAS model. We also present some examples of new proofs due to this specification. Notice that although formulae are given for 2 agents, they could have be given for an unspecified number of agents.

4.1 Multiagent model

As specified in definition 6, the multiagent system is specified by three parts: $(\mathcal{E}, \mathcal{T}, \mathcal{A})$.

Environment.

Here, as in the RoM problem, the environment contains the grid representing the Mars planet on which agents move. The invariant of the environment specifies the type of the grid and the fact that two R2 agents are on different cells. Following the structure presented in definition 1, the environment of the ERoM problem is the following:

$$\mathcal{E}=\left(\begin{array}{l}\{G\},\\ \left(\begin{array}{l} G \in X_{min}..X_{max} \times Y_{min}..Y_{max} \rightarrow \{clean, dirty\}\\ \wedge \left(\begin{array}{l}\{(a,(x,y)) \,.\, ((a,x) \in x_{R_2}^S \wedge (a,y) \in y_{R_2}^S)\}\\ \in Agents_{R_2} \rightarrowtail (X_{min}..X_{max} \times Y_{min}..Y_{max})\end{array}\right)\end{array}\right),\\ \{\}\end{array}\right)$$

Agent types.

Considering the agent types, we have two kinds of agents, $R1$ and $R2$. These types are quite similar to the agents $R1$ and $R2$ of the original RoM problem, so, we let the reader refer to [8] for more details. There are three main differences:

- the position of the $R2$ robots is now described by constant surfaces of the $R2$ type
- the goal decomposition of $R1$ is modified has $R1$ has now to bring the pieces of garbage it finds to *the nearest* R2 robot.
- we have to add a leads-to property to the R2 type.

Adding surface constants to the type $R2$ is straightforward. Considering the new GDT of $R1$ (that will be described later in this section), the set of agent types is then the following:

$$\mathcal{T}=\left\{\begin{pmatrix}(R1, V_{i_1}, \{\}, V_{\mathcal{E}_1}, init_1, \{\}, \{\}, \{\}, Actions_1, GDT_1),\\ \begin{pmatrix}R2, \{busy, x_{R2}, y_{R2}\}, \{x_{R2}, y_{R2}\}, \{G\}, init_2, \{\},\\ \{\Box(G(x_{R2}, y_{R2}) = dirty \rightarrow \diamond G(x_{R2}, y_{R2}) = clean)\},\\ \{\}, Action_2, GDT_2(x, y)\end{pmatrix}\end{pmatrix}\right\}$$

In the formula above, characteristics that do not need to be detailed in this paper are just represented by a variable (this is for instance the case of $init_1$, or $Actions_2$). Notice that the GDT of the $R2$ robots is now parameterized, allowing to specify different positions for different robots.

Agents.

As presented in the introduction, there will be only one agent of type $R1$. As explained above, we will suppose in the sequel that there are only two agents of type $R2$, but we could let this number unspecified. So, the set of agents in the system is the following:

$$\mathcal{A} = \{\ (r1, R1, ()), (r2_a, R2, (x_a, y_a)), (r2_b, R2, (x_b, y_b))\ \}$$

4.2 Type R1

4.2.1 Old version with only one robot R2

The previous version of the GDT of the robot $r1$ is detailed in [8]. Its graphical representation is shown on figure 1. Here is a brief description of the GDT:

goal 1 is the main goal: it consists in having cleaned the whole grid. It can be achieved by iterate on the following sequence of subgoals: cleaning the current cell (goal 3) and going to the next cell (goal 25).

To clean the current cell, if it is already cleaned, $R1$ just have to record this fact (goal 4). Otherwise, R1 must pick the piece of garbage, (goal 6), bring it to $R2$ (goal 11) and go back to the cell where the piece of garbage was.

From the initial specification [2], it is said that the *pick* action can fail but succeed after at most three attempts. This is described by the subtree with goal 6 as root goal.

After having noted the current position (goal 12), bringing the piece of garbage to $R2$ consists in going to $R2$'s cell (goal 14) and then, when $R2$'s cell is empty (goal 19), drop the piece of garbage on this cell.

Decompositions of goals 14 and 21 detail how $R1$ reaches a cell by iterating on either a vertical or horizontal move.

Going to the next cell (goal 25) consists in iterating on the subgoal 28 (if the end of the grid has not yet been reached). Actually, subgoal 28 makes the agent moving from one cell either horizontally or vertically. But doing so, $R1$ may reach R2's cell. That is why an iteration is necessary until $R1$ reaches a cell on which $R2$ is not.

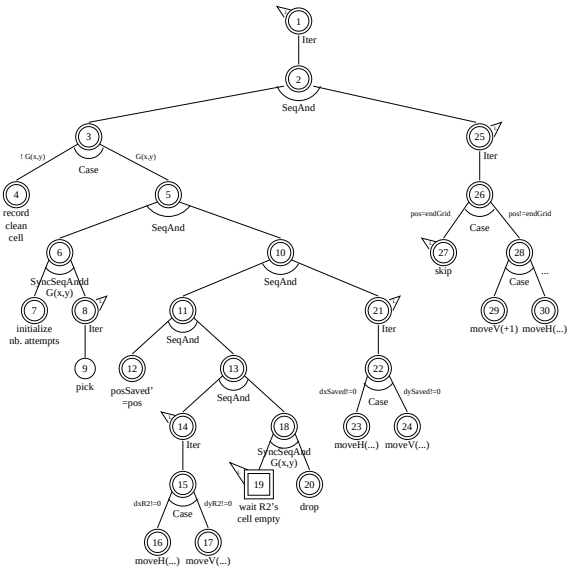

Figure 1: Old version of the GDT of robot R1

4.2.2 New version

The GDT or $R1$ must be modified for two reasons: references to $R2$ must be modified as there is now many robots of type $R2$; and the specification of $R1$ has changed: it has to bring the pieces of garbage to the *nearest* robot of type $R2$. The first modification implies to slightly modify the satisfaction conditions of the goals of the GDT of $R1$, whereas the second one requires to (partly) modify the GDT.

Modification of some Satisfaction conditions.

The set of satisfaction conditions of the GDT of $R1$ can be found in [8]. We just give here a few examples:

The satisfaction condition of goal 2 was the following[4]:

$$SC_2 = \bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}(\neg busy' \wedge (x,y) \neq (x_{R2}, y_{R2})\\ clean' = clean \cup \{(x,y)\}\\ clean' = lt((x',y'))\end{array}\right)\\ (\neg busy' \wedge clean' = dom(G))\end{array}\right).$$

Informally, this satisfaction condition expresses that after goal 2 has been achieved, the robot does not hold any object and either it is not on the cell of the robot $R2$, and the current cell is considered as clean or the whole grid is clean.

To help the reader in reading next formulae, we will use the shortcut $PosR2$ (the set of positions of robots of type $R2$) with the following definition:

$$PosR2 = \{(x,y).(\exists a \in \mathcal{A}_{R2}.(x^S_{R2}(a) = x \wedge y^S_{R2}(a) = y))\}$$

As GDTs are now associated to types and many agents of a given type can be present in the system, this satisfaction condition becomes:

$$SC_2 = \bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}(\neg busy' \wedge (x,y) \notin PosR2\}\\ clean' = clean \cup \{(x,y)\}\\ clean' = lt((x',y'))\end{array}\right)\\ (\neg busy' \wedge clean' = dom(G))\end{array}\right).$$

[4] in this formula, $lt(x,y)$ is the set of cells travelled by $R1$ from the beginning of the grid until the cell (x,y) and $dom(f)$ represents the domain of the function f.

Most satisfaction conditions referring to R2'cell have to be modified in the same manner. For instance, the satisfaction condition of the goal 14 (Go to $R2$) was:

$$SC_{14} \equiv (x, y) = (x_{R2}, y_{R2}) \wedge busy$$

and the new version should be:

$$SC_{14} \equiv (x, y) \in PosR2 \wedge busy$$

Modification of the Goal Decomposition Tree.

As specified in the introduction of this article, the robot $R1$ has now to reach *the nearest* robot $R2$. Thus, in the GDT of $R1$, the link between goal 13 ("go and give R2") and goal 14 ("go to R2") has to be modified: whereas the satisfaction of goal 13 is quite unchanged (its informal specification becomes "go and give to a robot of type R2", and its formal specification is modified as specified above), the left child goal of its decomposition is now a new goal 13.1 ("go to the chosen R2"). This goal is itself decomposed with a *SeqAnd* operator between a goal 13.2 ("determine the nearest robot R2") and the old goal 14 slightly modified (its satisfaction condition is now "go to the chosen R2"; the same modification is performed for the satisfaction conditions of all the subtree, that is to say goals 15 to 17). This decomposition is shown on figure 2.

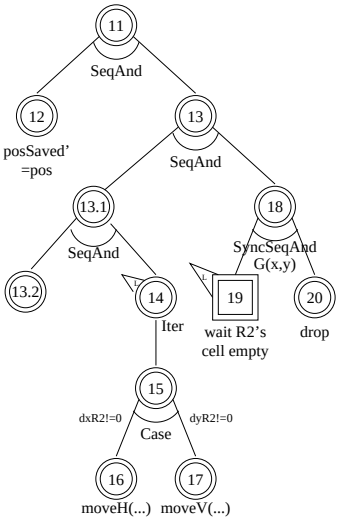

Figure 2: Partial new version of the GDT of robot R1

Most satisfaction conditions are given below. We will use the *dist* function specified by:

$$dist : \left| \begin{array}{ccc} \mathcal{A}_{R2} & \rightarrow & \mathbb{R} \\ a & \mapsto & \left|x^S_{R2}(a) - x\right| + \left|y^S_{R2}(a) - y\right| \end{array} \right.$$

Moreover, $nr2$ will represent the nearest $R2$ robot of $R1$.

$$SC_{13} = \left\{ \begin{array}{l} (x', y') \in PosR2 \wedge \neg busy' \wedge G'(x', y') = dirty \\ (xSaved', ySaved') = (xSaved, ySaved) \end{array} \right.$$

$$SC_{13.1} = \left\{ \begin{array}{l} (x', y') = (xR2chosen, yR2chosen) \\ (xR2chosen, yR2chosen) \in posR2 \\ (xSaved', ySaved') = (xSaved, ySaved) \\ busy' \end{array} \right.$$

$$SC_{13.2} = \left\{ \begin{array}{l} nr2 = choice(dist^{-1}[\{min(dist[\mathcal{A}_{R2}])\}]) \\ xR2chosen = x^S_{R2}(nr2) \\ yR2chosen = y^S_{R2}(nr2) \end{array} \right.$$

$$SC_{14} = \left\{ \begin{array}{l} (x', y') = (xR2chosen, yR2chosen) \\ busy' \end{array} \right.$$

$$SC_{18} = \left\{ \begin{array}{l} \neg busy' \\ G'(x', y') = dirty \\ (x', y') = (x, y) \\ (xSaved', ySaved') = (xSaved, ySaved) \end{array} \right.$$

$$SC_{19} = \left\{ \begin{array}{l} busy' \\ G'(x, y) = clean \end{array} \right. \qquad SC_{20} = \left\{ \begin{array}{l} \neg busy' \\ G'(x', y') = dirty \end{array} \right.$$

4.2.3 New Proofs

Most of the proofs that were made by the authors of the initial version (and that can be found in [8]) can be performed on this new version (the changes in satisfaction conditions presented in the paragraph *Modification of some satisfaction conditions* of the previous section generate only syntactic modifications in the proofs). So, the only proofs that has to be performed are the following:

- validity of the decomposition of the goal 13 and 13.1;
- validity of the external goal 19.

Decomposition of goal 13*.*

First of all, we give the simplified proof schema of the *SeqAnd* operator when A is decomposed into B *SeqAnd* C:

$$LH \vdash (([v' := v^{tmp}]SC_B \wedge [v := v^{tmp}]SC_C) \rightarrow SC_A)$$

where $[v := v^{tmp}]P$ represents the substitution of all the free occurences of all the variables in P by fresh variables having the same name but superscripted by tmp and LH (local hypotheses) summarizes the context of the goal A.

The application of this proof schema to the decomposition of the goal 13 by goals 13.1 and 18 requires to establish the following property:

$$LH \vdash (([v' := v^{tmp}]SC_{13.1} \wedge [v := v^{tmp}]SC_{18}) \rightarrow SC_{13})$$

That is to say:

$$LH \vdash \left| \begin{array}{l} \left\{ \begin{array}{l} (x^{tmp}, y^{tmp}) = (xR2chosen, yR2chosen) \\ (xR2chosen, yR2chosen) \in posR2 \\ (xSaved^{tmp}, ySaved^{tmp}) = (xSaved, ySaved) \end{array} \right. \\ \wedge \\ \left\{ \begin{array}{l} \neg busy' \wedge G'(x', y') = dirty \\ (x', y') = (x^{tmp}, y^{tmp}) \\ (xSaved', ySaved') = (xSaved^{tmp}, ySaved^{tmp}) \end{array} \right. \\ \rightarrow \\ \left\{ \begin{array}{l} (x', y') \in PosR2 \wedge \neg busy' \\ G'(x', y') = dirty \\ (xSaved', ySaved') = (xSaved, ySaved) \end{array} \right. \end{array} \right.$$

which is true for the following reasons:

- as $(x', y') = (x^{tmp}, y^{tmp})$, $(x^{tmp}, y^{tmp}) = (xR2chosen, yR2chosen)$ and $(xR2chosen, yR2chosen) \in posR2$ is in hypotheses, we have $(x', y') \in PosR2$;
- $\neg busy'$ and $G'(x', y')$ are in hypotheses;
- as $(xSaved', ySaved') = (xSaved^{tmp}, ySaved^{tmp})$ and $(xSaved^{tmp}, ySaved^{tmp}) = (xSaved, ySaved)$ are in hypotheses, $(xSaved', ySaved') = (xSaved, ySaved)$.

Notice that this proof does not depend on the number of $R2$ robots in the system.

Decomposition of goal 13.1.

This decomposition uses the same proof schema as the previous one. Moreover, the proof is straightforward. So, we decide not to present it here.

External goal 19.

The context of this goal is $(x, y) \in posR2 \wedge busy$ and its satisfaction condition is $busy \wedge G(x, y) = clean$. So, proving the correctness of the behaviour of an agent $R1$ will require to prove that in the system, an agent satisfies the following property:

$$\Box((x, y) \in posR2 \rightarrow \diamond G(x, y) = clean) \quad (1)$$

This property cannot be proven now because the type $R2$ has not yet be defined. It will be proven in section 4.4.

4.3 Type R2

4.3.1 Old Version

The old version of the GDT of those robots is a simple GDT where the goal 1 is decomposed into 2 *SeqAnd* 3 with the following satisfaction conditions and contexts:

$$SC_1 \equiv \neg busy_{R2} \wedge G(x_{R2}, y_{R2}) = clean$$
$$SC_2 \equiv busy_{R2} \wedge G(x_{R2}, y_{R2}) = clean$$
$$SC_3 \equiv \neg busy_{R2} \wedge G(x_{R2}, y_{R2}) = clean$$
$$C_1 \equiv \neg busy_{R2} \wedge G(x_{R2}, y_{R2})) = dirty$$
$$C_2 \equiv \neg busy_{R2} \wedge G(x_{R2}, y_{R2})) = dirty$$
$$C_3 \equiv busy_{R2} \wedge G(x_{R2}, y_{R2})) = clean$$

In this version, the position of the unique robot R2 was specified by two environment constants x_{R2} and y_{R2}.

Recall the following properties of this GDT:

- its triggering context is $TC_{R2} \equiv G(x_{R2}, y_{R2}) = dirty$;
- its precondition is $Prec_{R2} \equiv \neg busy_{R2}$.

4.3.2 New Version

As specified above, as we have now many agents of type $R2$, and as each one has to know its own position, whereas the agent of type $R1$ must know the positions of all the robots of type $R2$, the positions of the robots of type $R2$ will be represented by surface variables.

The other parts of the type $R2$ have been given in section 4.1. We just want to focus on the following changes:

- The position is specified by surface variables;
- a *leads-to* property is associated to the robots $R2$.

4.3.3 Proof

As this GDT has not been modified, proofs already performed in [8] are still valid. So, the only proof remaining is the proof of the *leads-to* property:

$$\Box(G(x_{R2}, y_{R2}) = dirty \rightarrow \diamond G(x_{R2}, y_{R2}) = clean)$$

From the proof principle presented in section 3.7, we must:

1. determine goals S_1 whose context is compatible with $G(x_{R2}, y_{R2}) = dirty$;
2. determine NS goals S_2 whose satisfaction condition implies $G(x_{R2}, y_{R2}) = clean$,
3. verify that when an agent is trying to achieve a goal of S_1, it will eventually try to achieve a goal of S_2 (this requires to prove that the antecedent of the leads-to property implies the triggering context of the GDT).

In a straightforwad manner, we can compute that $S_1 = \{1, 2\}$ and $S_2 = \{1, 2, 3\}$.

The third step is verfied from the semantics of the *SeqAnd* operator and from the fact that goal 1 and 2 are non-lazy goal, and that they are NS goal. Moreover, as the triggering context of this GDT is $TC_{R2} \equiv G(x_{R2}, y_{R2}) = dirty$ and as the antecedent of the leads-to property to verify is $G(x_{R2}, y_{R2}) = dirty$, the fact that the antecedent of the leads-to rule must implies the triggering context is obvious.

4.4 Proof of collaboration between robots

Many proofs can be performed at the system level (we have performed not only proven invariant properties but also liveness properties), but for space reasons, we just present here the proof of the fact that the external goal of $R1$ will be achieved by another robot. The hypotheses are the following:

- the exernal goal of $R1$ is specified by the following satisfaction condition and context (restricted to environment variables or surface variables of other agents):

$$SC_{19} \equiv G(x, y) = clean \quad (2)$$
$$C_{19} \equiv (x, y) \in posR2 \quad (3)$$

- the set of $R2$ agents is:

$$\mathcal{A}_{R2} = \{(r2_a, R2, (x_a, y_a)), (r2_b, R2, (x_b, y_b))\} \quad (4)$$

- from the set of agents in the system and from the specification of the agents types, the two following properties stand in the system[5]:

$$\Box(G(x_a, y_a) = dirty \rightarrow \diamond(G(x_a, y_a) = clean)) \quad (5)$$
$$\Box(G(x_b, y_b) = dirty \rightarrow \diamond(G(x_b, y_b) = clean)) \quad (6)$$

Now, we have to prove that the external goal 19 of $R1$ will be achieved by another agent in the system, that is to say prove property 1.

So, if G(x,y) is already clean, the goal is achieved. Otherwise, $G(x, y) = dirty$ and, as $(x, y) \in posR2$, from equation 4, we have two cases: either $(x, y) = (x_a, y_a)$ or $(x, y) = (x_b, y_b)$:

- if $(x, y) = (x_a, y_a)$, then $\diamond G(x, y) = clean$ (property 5),
- if $(x, y) = (x_b, y_b)$, then $\diamond G(x, y) = clean$ (property 6).

So, the external goal of agent $R1$ will eventually be achieved by the system when it is required by $R1$.

5. COMPARISON WITH OTHER WORKS

For a few years, works dealing with the formal specification and the verification of agents appear in the multiagent community [1, 7, 5]. However, those on the verification of multiagent systems are rare. The main reason is certainly the difficulty of the problem: model-checking cannot be applied to systems with too many agents for complexity reasons, and theorem proving is not easy to implement.

A first solution to this dilemma is proposed in [3], where the authors propose to do model-checking on a finite and small set of traces, performing so testing and not proof. If this solution provides an intersting validation method, it

[5]Notice that we detail here both properties but whatever the number of agents is, one property would be enough.

does not respond to the problem of the verification of multiagent systems.

In order to be able to manage the problem, a proper language is required. As specified by Fisher in [6], this language must have the following characteristics:

- it must be a high level language and concise;
- it must provide to the designer few but powerful operators;
- its semantics must be intuitive;
- it must allow to specify static and dynamics aspects;
- it must not limit the designer by operational constraints (for instance, the language must allow the designer to implement agents doing several tasks in parallel).

A few agent languages satisfy all these properties. Our model, for instance, does not fulfil the last point (we claim however that it satisfies the other charcteristics). For instance, the semantic of Concurrent MetateM [7] is not so intuitive and 2APL [4] is not very concise.

An important characteristic of a language allowing to specify a multiagent system is that it must allow to formally specify... the system! Different solutions are proposed. Most of them [7, 3] do not introduce the notion of agent type, and so of instances. With these models, the designer must specify a behaviour for each agent in the system. On the contrary, 2APL [4] allow to create several instances for each type of agent, but contrary to our model, instances cannot be parameterized.

The environment is not always clearly specified. For instance, within 2APL, the environment is specified by a Java class, and so, theorem proving can not be done at the specification level. In Concurrent MetateM, the environment is formally specified and moreover, an agent can evolve in several environments.

When dealing with the architecture of a single agent, a few models propose intra-agent parallelism (several concurrent plans in 2APL for instance), a feature that the GDT4MAS model does not propose yet.

Collaborations between agents, is not always clearly specified. This is for instance the case for TTL [3]. In MetateM, communication via messages can be specified but messages are implemented by environment variables, and can be so compared with our external goals.

Considering the support of the methods, contrary to 2APL, MetateM or TTL, we only have a prototype of software environment to handle our specifications.

A last comparison item is the proof capacity of the method. As explained above, TTL does not allow to make a real proof. Within 2APL, a large part of the system is written in Java, and so, only model-checking on the final system can then be done, and can only be performed on very small systems. Considering MetateM, which allows to make proof, no compositional proof system is specified and so, the proof can only be performed on the whole system, which leads rapidly to proofs untractable by existing theorem provers.

6. CONCLUSION

In this article, we have shown how the GDT model can be extended to specify multiagent systems. The key point of this extension is the fact that it preserves the compositional aspect of GDTs. As specified by Fisher in [6], this aspect is essential to verify multiagent systems: "the work on compositional temporal specification and verification will have some bearing on our ability to effectively verify large multiagent systems". We have also shown on a small example that the GDT4MAS model can be applied and that it allows to manage proofs at the system level in a compositional way, a characteristic that is not implemented by the other proof systems we have read about. However, a few features of these other proof systems could be added to our model, and we will focus on this in the future: expressing true communications and parallelism inside the specification of an agent are two examples. As well, a better software support would be interesting. Further research dealing with the dynamicity of the agents population should also be considered as well as a structuration of the agents population as in MetateM [7].

7. REFERENCES

[1] N. Alechina, M. Dastani, B. Logan, and J.-J. C. Meyer. A Logic of Agent Programs. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 795–800. AAAI Press, 2007.

[2] R. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model-checking AgentSpeak. In *AAMAS-03*, Melbourne, Australia, 2003.

[3] T. Bosse, C. Jonker, L. van der Meij L., A. Sharpanskykh, and J. Treur. Specification and verification of dynamics in agent models. *International Journal of Cooperative Information Systems*, to appear.

[4] M. Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[5] F. de Boer, K. Hindriks, W. van der Hoek, and J.-J. Meyer. Agent Programming with Declarative Goals. In *7th International Workshop on Intelligent Agents. Agent Theories Architectures and Language*, pages 228–243, 2000.

[6] M. Fisher. Representing and Executing Agent-Based Systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, volume 890 of *LNCS*, pages 307–323. Springer, 1994.

[7] M. Fisher. MetateM: the story so far. In *Programming Multi-Agent Systems, Third International Workshop, ProMAS 2005*, volume 3862 of *LNCS*, pages 3–22. Springer, 2005.

[8] B. Mermet, G. Simon, B. Zanuttini, and A. Saval. Specifying and verifying a mas: The robots on mars case study. In M. Dastani, A. El-Fallah, A. Ricci, and M. Winikoff, editors, *Programming Multi-Agent Systems, 5th International Workshop, ProMAS 2007*, volume 4908 of *LNAI*, pages 172–189. Springer, 2008.

[9] G. Simon, B. Mermet, and D. Fournier. Goal Decomposition Tree: An agent model to generate a validated agent behaviour. In M. Baldoni, U. Endriss, A. Omicini, and P. Torroni, editors, *Declarative Agent Languages and Technologies III: Third International Workshop, DALT 2005*, volume 3904 of *LNCS*, pages 124–140. Springer Verlag, 2006.