Specifying and Verifying Holonic Agents with GDT4MAS

Bruno Mermet, Gaële Simon

Specifying and Verifying Holonic Agents with GDT4MAS

B. Mermet

GREYC-UMR 6072 & Université du Havre
Campus Côte de Nacre, Boulevard du Maréchal Juin
BP 5186, 14032 CAEN cedex, FRANCE
e-mail: Bruno.Mermet@univ-lehavre.fr

G. Simon

GREYC-UMR 6072 & Université du Havre
Campus Côte de Nacre, Boulevard du Maréchal Juin
BP 5186, 14032 CAEN cedex, FRANCE
e-mail: Gaele.Simon@univ-lehavre.fr

Abstract:
This paper describes how specific holonic multi-agent systems can be specified and how their correctness can be proven with an extended version of the GDT4MAS model. This model allows the specification of multi-agent systems and the verification of their correctness with theorem proving techniques. Introducing holonic agents in this model allows the enhancement of its expressiveness. Moreover, the proof system associated to this model can be easily extended in order to prove the correctness of multi-agent systems using such agents. The paper first describes the initial GDT4MAS model. Then the need for some kinds of holonic agents and their proposed specification based on specific decomposition operators are presented. It is followed by a focus on how the proof system can be adapted to prove the correctness of the behaviour of these new agents. Last but not least, all these proposals are illustrated on a case study.

Keywords: Holonic agent, Formal specification, Verification

Reference to this paper should be made as follows: B. Mermet and G. Simon, specifying and verifying holonic agents with GDT4MAS, *Int. J. of Agent Oriented Software Engineering*, Vol. x, No. x, pp.xxx-xxx.

Biographical notes: Bruno Mermet received his PhD on compositional verification of telecommunication services in 1997 from the University of Nancy 1, France. He is a lecturer at the University of Le Havre and works for the Computer Science Laboratory of Caen. Gaële Simon received her PhD on corporate memories in 1996 from the University of Nancy 1, France. She is a lecturer at the University of Le Havre and is a member of the Computer Science Laboratory of Caen.

1 Introduction

1.1 Motivations

For several years, the agent paradigm has overcome distributed artificial intelligence research domain. However, industrial applications of multi-agent systems (MAS) are still rare. One of the main highlighted reasons is the lack of confidence people have in multi-agent systems. To solve this problem, at least two ways have been considered: dedicated methodologies to inforce confidence in developed MAS and formal techniques to specify and verify MAS. In this area, two kinds of techniques have been used: model checking and theorem proving. As theorem proving can be performed early in the development process and as it can be performed on infinite models, our proposal is situated in this research area. However, having a compositional model is necessary because it reduces the number of proof to perform together with their complexity.

For some years now, we have developed a compositional formal model with an associated proof system for MAS. This model is called GDT4MAS (Goal Decomposition Trees for Multi-Agent Systems) [Mermet and Simon (2009)], as agents behaviours are specified by Goal Decomposition Trees.

We are now interested in nested MAS because such systems increase compositionality. Indeed, considering a group of agents as a single agent from the point of view of the other agents may simplify the proof of the system. Moreover, the GDT4MAS model can easily allow the verification of such systems. So, the aim of this article is to present how the GDT4MAS model can be modified to specify some kinds of nested agents, and how these agents can be verified. As a consequence, the notions of super-agent and sub-agent used in this article (the terminology is taken from [Gerber et al. (1999)]) must be defined.

Definition 1.1 (super-agent) *A super-agent situated in an environment $\mathcal{E}$ is an agent and looks like any other agent from the point of view of the other agents situated in $\mathcal{E}$. Its general behaviour consists in maintaining maintainance goals and achieving achievement goals, but parts of its behaviour are achieved by sub-agents.*

Definition 1.2 (sub-agent) *A sub-agent n is an agent, the behaviour of which implements a part of the behaviour of super-agent h it belongs to (a sub-agent cannot belong to several super-agents, except by transitivity). n cannot see a part of the environment in which h is situated that h does not see. Formally, let $E(a)$ the environment in which an agent a is and $ES(a)$ the part of $E(a)$ seen by a. Then $ES(n) \cap (E(h) - ES(h)) = \emptyset$.*

With such a structure, our proposal explores the world of holonic MAS. A specification of a holon h is made of either a single agent a (*atomic holon*) or a set of sub-holons (sh_i) (*non-atomic holon*). In both cases however, a holon has a specific behaviour (specified by a GDT), as in [Dennis et al. (2008)]. It means also that a holon does not have a *representative* agent. The behaviour of a non-atomic holon nh depends upon a structure that changes dynamically: sometimes, the behaviour of nh is only specified by its GDT whereas at other moments, it is specified by its sub-holons, the coordination of which is partly controlled by nh.

However, systems specified by GDT4MAS are a subset of holonic MAS. Indeed, we only consider holonic MAS where a sub-holon is included in at most one super-holon and cannot move from a super-holon to another. The super-holon in which it is included specifies its environment. This structure is represented in figure 1.

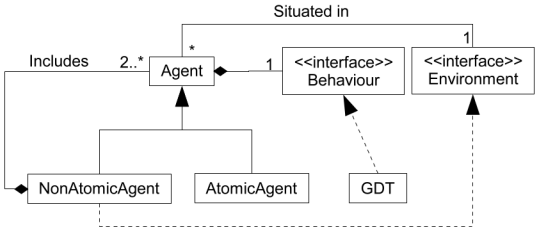


Figure 1 Structure of the new GDT4MAS model

In the GDT4MAS model, sub-holons of a non-atomic holon nh can be either transient or persistent: a transient sub-holon is created when nh needs it, and is destroyed when it is no longer needed. On the contrary, a persistent sub-holon is created and destroyed with nh. In that case, nh controls when the sub-holon is active.

1.2 Case study

Throughout this paper, a case study is used to exemplify our model. Please notice that the whole case study can be found as a stand-alone paper on the web [Mermet and Simon (2010)].

The example presented here is an extension of the problem presented in [Bordini et al. (2003)]. Although it may look very simple, it allows to give examples of proofs that are tractable by the reader. The initial problem, called *RoM* (Robots on Mars) in the sequel, is the following: two robots have to clean Mars. Robot $R1$ walks through the surface of Mars (represented by a grid) looking for wastes. If it finds one, it tries to pick it up (this goal is achieved with at most three attempts), brings it to robot $R2$, goes back to the position where it had found the waste, and continues to explore Mars. Robots $R2$ cannot move but it can burn wastes that are given to it.

In their paper, Bordini *et al.* use model checking to verify their specification. So, they have chosen to verify the correctness of their specification on a 5×5 grid containing two wastes. In a previous work, it has been shown that the GDT model allows the verification for a grid of any size and with any number of wastes [Mermet et al. (2007)]. More recently, we have shown that with the GDT4MAS model, the system can easily be verified even if there are several robots of each type and if $R1$ must go to the nearest robot $R2$ [Mermet and Simon (2009)] (Extended RoM – ERoM – problem). The complexity of the proof process does not depend upon the number of robots.

In this article, we briefly present how both types of agents $R1$ and $R2$ can be implemented by super-agents. In order to do so, the specification of the *ERoM* problem is slightly modified as follows:

Problem 1.1 (Compound ERoM Problem (CERoM)) *Robots of type $R2$ are made of two parts: an arm and an incineration system. The arm picks up wastes on the cell in which the robot is situated. It then puts the waste it holds in a tank with a limited capacity of 10 wastes. In the incinerator system, a conveyor belt starts when the tank contains three wastes or more. Then, it brings all the wastes in the tank one by one to an incinerator. During this process, if the incinerator*

supervisor detects that the temperature of the incinerator is abnormal (either too low or too high), it changes the status of the robot R2 to down. *Robots of type R1 bring pieces of garbage they find to the nearest* up *robot R2. While moving to a robot R2, if the target robot breaks down, they compute a new target robot.*

The whole structure of the CERoM system is summarized in figure 2 where only one instance of each type of agent is detailed.

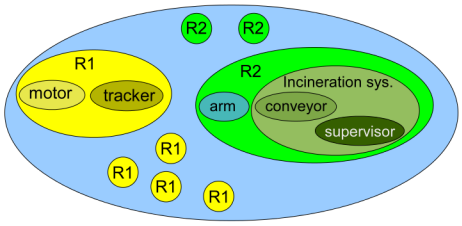


Figure 2 Stucture of the robots

In the next section, GDT4MAS model is briefly presented. In section 3, it is shown how GDT4MAS model is extended to specify specific kinds of holonic systems. Section 4 give some design patterns dedicated to holonic systems that help to use GDT4MAS model. Finally, a brief comparison with other works is presented in section 5.

2 GDT4MAS model

GDT4MAS is an extension of the GDT model [Mermet et al. (2007)] that has been presented in [Mermet and Simon (2009)]. The aim of this model is to allow the specification and the verification of a MAS. To do so, the notion of *proof obligation* that can be found in general-purpose formal methods used in industry as the B method [Abrial (1996)] has been instanciated.

Definition 2.1 (Proof Obligation) *A Proof obligation (PO) is a logical formula that must be verified to guarantee the correctness of a specification.*

One of the roles of a formal method is to provide a mean to generate automatically the set of necessary POs. To do so, GDT4MAS provides *Proof Schemas* (PS), that, applied to a specification, give the required set of POs. As in standard model-oriented formal methods such as Z, VDM or B, GDT4MAS specifications rely on first-order logic, set theory and arithmetic. The need of a formal-method dedicated to MAS is justified in [Simon and Flouret (2006)].

2.1 The MAS structure

Our model defines the notions of *environment*, *agent type* and *agent* as follows:

Definition 2.2 (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 and* $s_\mathcal{E}$ *is the set of* stable properties *of the environment.*

Example 1 *[Environment of CERoM] For reasons of simplicity, stable properties s are not detailed. So, the environment of CERoM is* $\mathcal{E} = (G, G \in x_{min}..x_{max} \times y_{min}..y_{max} \rightarrow \{clean, dirty\}, s)$, *where G is a variable specifying for each cell of Mars whether it is clean or dirty, where* x_{min}, x_{max}, y_{min} *and* y_{max} *are constants of the environment.*

Definition 2.3 (Agent type) *Let $\mathcal{E}$ an environment. A type of agent T is a tuple* $(\texttt{name}_\texttt{t}, V_i, V_s, V_\mathcal{E}, init, I, L, S, Actions, \texttt{Beh})$ *where* $\texttt{name}_\texttt{t}$ *is the name of the type, V_i is the set of internal variables (an internal variable is a variable that can only be seen and modified by the owner agent), V_s is the set of surface variables (a surface variable is a variable that can be seen by every agent but that can only be modified by the owner agent), $V_\mathcal{E}$ is the set of perceived environment variables, init describes how internal and surface variables are initialised, I is the invariant property verified by internal and surface variables, 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 and* Beh *describes the behaviour of the agents of this type (namely a parameterized GDT, see definition 2.5).*

Example 2 *[R2 type] Type R2 is defined by the tuple* $(R2, \{T\}, \{x_{R2}, y_{R2}, status\}, \{G\}, init, T \in 0..10 \wedge x_{R2} \in x_{min}..x_{max} \wedge y_{R2} \in y_{min}..y_{max} \wedge status \in \{up, down\}, L, S, Actions, GDT_{R2})$ *where init, L, S, Actions and GDT_{R2} are not detailed and variables have the following meanings:*

- *G represents the grid. It is a variable of the environment of the system;*
- *x_{R2} and y_{R2} are constants of $R2$ that are initialized by parameters;*
- *$status$ is a variable of $R2$ specifying if it is up or down;*
- *T is a variable of $R2$ specifying the number of wastes in the tank;*

Definition 2.4 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 and param is the list of effective parameters for the GDT associated to the type of the agent.*

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

The representation of the agent population in the environment is detailed in a previous article [Mermet and Simon (2009)]. The current version of the GDT4MAS model assumes that the population of agents is stable.

2.2 Specification of a behaviour thanks to a GDT

The behaviour of an agent type is specified by a parameterized Goal Decomposition Tree (pGDT), which means that the behaviour of an agent is specified by a GDT.

Definition 2.5 (parameterized GDT) *A parameterized GDT is specified by a tuple $(params, prec, trig, init, mainGoal)$ where params is the set of formal parameters of the pGDT, prec is the precondition of the pGDT (it must be established by the initialisation of the pGDT. Moreover, it must be true at the end of the execution of the agent), trig is the triggering context of the pGDT (the agent begins to execute its behaviour each time its triggering context becomes true), init specifies how the variables of the agent are initialized when the agent is created, and mainGoal is the main achievement goal of the pGDT.*

Example 3 *[R2 and R2's arm pGDTs] The pGDT of type $R2$ is specified by the tuple $(\{x_0, y_0\}, true, G(x_{R2}, y_{R2}) = dirty, x_{R2} = x_0 \wedge y_{R2} = y_0, mg_{R2})$ where both parameters are used to specify the position of the robot. The main goal of robot $R2$, mg_{R2}, consists, if $R2$ is up, removing the waste from its cell and in maintaining its tank not full (see later).*

The pGDT of type arm is $(\emptyset, true, G(x_{R2}, y_{R2}) = dirty, busy = false, mg_{arm})$. The main goal of the arm mg_{arm} consists in cleaning R2's cell and being not busy.

2.3 Goals

In GDT4MAS, maintenance goals are specified by invariants, and so, a pGDT only contains achievement goals. A goal in a pGDT is specified as follows:

Definition 2.6 (Goal) *A goal is specified by a tuple* $(name, SC, GPF, L, NS, dec)$ *where: name is the name of the goal, SC is the satisfaction condition of the goal (it expresses the property established by the goal if its execution succeeds), GPF is a logical formula that expresses the property that is guaranted to be verified by the goal execution process if it fails, L is a boolean describing the lazyness of the goal (when a lazy goal – L goal – must be achieved, its decomposition is not executed if the SC of the goal is already true), NS is a boolean specifying the satisfiability of the goal (a necessarily satisfiable goal – NS goal – is a goal whose execution always succeeds, contrary to NNS goals), and dec describes how the goal is decomposed (see section 2.3.3).*

Example 4 *[Main goals of R2 and of its arm] We recall that the GPF of an NS goal is true. Moreover, both considered goals are NL and NS goals. So we obtain both following goals where* $SC_{mg_{R2}}$ *and* SC_{arm} *are defined in example 5 whereas* $dec_{mg_{arm}}$ *is defined in example 6 (*$dec_{mg_{R2}}$ *can be found in example 11):*

- $(mg_{R2}, SC_{mg_{R2}}, true, false, true, dec_{mg_{R2}})$
- $(mg_{arm}, SC_{arm}, true, false, true, dec_{mg_{arm}})$

The satisfaction condition of a goal is specified using first-order logic. An SC may contain unprimed and primed occurences of variables: an unprimed occurence expresses the value of the variable before the goal execution whereas a primed occurence specifies the value of the variable after goal execution. So, the SC of a goal g that consists in increasing the value of a variable v by an undetermined value is: $SC_g \equiv (v' > v)$.

Example 5 *[Satisfaction conditions] Considering the meaning of the main goal of R2 given in example 3, we have* $SC_{mg_{R2}} \equiv (status = up \rightarrow (G(x_{R2}, y_{R2}) = clean \wedge T < 10))$. *We also have* $SC_{mg_{arm}} \equiv (G'(x_{R2}, y_{R2}) = clean \wedge \neg busy')$.

Two kinds of achievement goals are distinguished: *progress goals* expressing a link between the states before and after goal execution ($x' = x + 1$ means that the goal is to increase the value of x) and *state reaching goals* that express a desired state (such as $x' = 10$).

2.3.1 Intermediate goals

An intermediate goal is a goal that is decomposed into subgoals using *decomposition operators*. Height operators have been defined. Some of them are summarized here:

- *SeqAnd*: a sequential (and lazy) *and*. Subgoals are executed from left to right. As soon as one subgoal fails, execution of the parent goal ends. If all subgoals are achieved, parent goal is also achieved. A synchronized version, *SyncSeqAnd* allows the locking of some environment variables (locked variables are written as subscripts of the operator);
- *SeqOr*: a sequential (and lazy) *or*. Subgoals are executed from left to right. As soon as one subgoal is achieved, parent goal is also achieved and its execution stops. There is also a synchronized version;
- *Case*: a condition is associated to each subgoal. One subgoal whose condition is true is executed. If it is achieved, the parent goal is also achieved;

- *Iter*: an iteration operator. The single subgoal is executed until the parent goal is achieved.

2.3.2 Leaf goals

A leaf goal can be either an elementary goal or an external goal. An elementary goal is a goal that is not decomposed into subgoals. Instead, an action is attached to it. Actions are described in [Mermet et al. (2007)]. An external goal in a pGDT is an NS goal that must be achieved for a correct behaviour of the agent but that it cannot achieve itself: another agent in the system must achieve it, and the inner agent waits until it is achieved.

2.3.3 Goal decomposition

The decomposition of a goal G is one of the following: *a decomposition operator and a sequence of sub-goals* for an intermediate goal, *empty* for an external goal or for a goal whose decomposition is not specified or *an action* for an elementary goal.

Example 6 *[Decomposition of the main goal of the arm] The arm must first pick up the waste on its cell (goal* picking up*, becoming busy and then it must drop the garbage it is carrying in the tank (goal* dropping*). These two steps must lock the cell of R2 to prevent R1 from dropping a new waste to R2. As a consequence we have* $dec_{mg_{arm}} = (SyncSecAND_G, (picking\ up, dropping))$.

Example 7 *[Decomposition of the goal picking up] The leaf goal* picking up *can be achieved by the action* pick up *of the arm. So we have* $dec_{picking\ up} = (pick\ up)$.

2.4 Graphical representation

Intermediate nodes and leaf nodes are represented by ellipses containing the name of the goal or its SC. *NS* goals are surrounded by a double ellipse. Then names of the actions associated to elementary goals are written below the goal they are associated to. External goals are represented by a double-surrounded rectangle.

Example 8 *[Representation] Behaviours of the arm, the conveyor belt and the incinerator supervisor are represented in figure 3. busy is a variable of the arm specifying whether it is carrying a waste or not and temp is a variable of the incinerator supervisor specifying its temperature.*

2.5 Formal semantics

A formal semantics has been defined for GDTs using linear temporal logic (LTL). To do so, four temporal variables have been associated to each goal n: $init_n$ is true when the execution of goal n begins, end_n is true when the execution of goal n ends, in_n is true during the execution of goal n and sat_n is true when the execution of goal n ends with success.

General formulae that must be verified by each operator have been defined. Other formulae dedicated to each operator have been specified. Here is for instance the formulae defining the binary SeqAnd operator with a nonlazy parent goal:

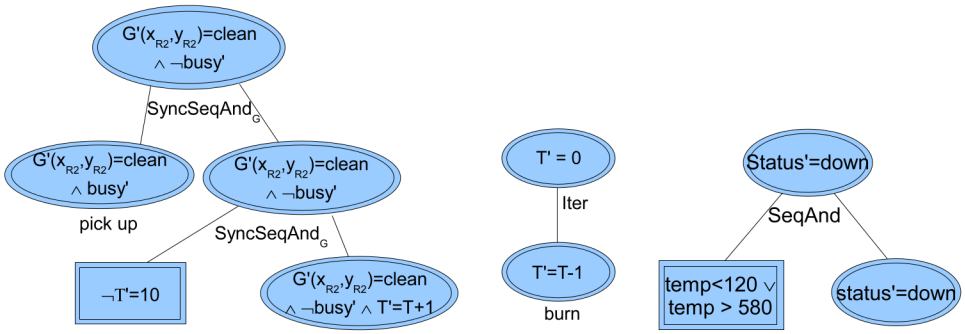
$$\Box(init_n \rightarrow init_{n_1}) \tag{1}$$

$$\Box(end_{n_1} \wedge sat_{n_1} \rightarrow o(init_{n_2})) \tag{2}$$

$$\Box(end_{n_1} \wedge \neg sat_{n_1} \rightarrow o(end_n)) \tag{3}$$

$$\Box(end_{n_2} \wedge sat_{n_2} \rightarrow (end_n \wedge sat_n)) \tag{4}$$

$$\Box(end_{n_2} \wedge \neg sat_{n_2} \rightarrow o(end_n)) \tag{5}$$



(a) pGDT of the arm

(b) pGDT of the conveyor belt

(c) pGDT of the incinerator supervisor

Figure 3 Some pGDTs of agent types involved in the system

2.6 Proofs, Proof schemas and Proof obligations

Proofs allowed within the GDT4MAS model are the following:

- The specified behaviour of each agent type terminates.
- The invariants (of the agent and of the environment) are preserved.
- The leads-to properties of the agents are established.
- Each decomposition of a goal into subgoals is correct (for instance, if a goal g is decomposed into two subgoals g_1 and g_2 with a SeqAnd operator, it must be proven that if goal g_2 is achieved after goal g_1 has been achieved, then goal g is also achieved).
- The success of the action associated to a leaf goal achieves it.
- An external goal of an agent in a MAS will eventually be achieved by another agent in the MAS.

In order to establish these properties, several Proof Schemas have been associated to the GDT4MAS model. These proof schemas can be applied automatically to pGDTs and in order to generate Proof Obligations (in the sequel, we only consider proof schemas dedicated to the verification of decompositions). These POs are predicates that can be proven either manually or automatically by any first-order logic (or higher-order logic) prover (a software allowing the usage of the *krt* and *PVS* provers is underway).

To make proofs of decompositions as easy as possible, we have chosen to make them compositional using *Proof Contexts*. A *Proof context* is a predicate which is true in states in which the decomposition has to be executed. The Proof context C_g of a goal g is inferred from the structure of the pGDT [Mermet and Simon (2009)].

Three special notations are used in these proof schemas:

- The predicate transformer $T_i^j(p)$ that substitutes in the predicate p each unprimed variable v by the variable v_i and each primed variable v' by v_j.
- The notation ISV_i^j for $\bigwedge_{v \in V_i \cup V_s}(v_i = v_j)$ where V_i and V_s are respectively the sets of internal variables and surface variables of an agent type. $ISLV_i^j$ is the natural extension of ISV_i^j to locked variables when dealing with synchronized operators.

- The predicate transformer At^i substitutes in the predicate to which it is applied each unprimed variable v by v_i.

Example 9 *[SyncSecAND proof schema and its application] Notions of proof schema and proof obligation are examplified on the* SynqSeqAnd *operator (NL version). Let consider a goal g decomposed thanks to a* SeqAnd *operator into g_1 and g_2. Four states are considered for such a decomposition: state* 1 *is the state in which executions of goals g and g_1 begin, state* 2 *is the state in which the execution of goal g_1 ends, state* 3 *is the state in which the execution of goal g_2 begins and state* 4 *is the state in which executions of goals g_2 and g end. Then the proof schema associated to the SyncSeqAnd operator is (in a simplified version) the following (C_g is the context of the goal g, and we recall that G is locked):*

$$At^1(C_g) \wedge T_1^2(SC_{g_1}) \wedge ISLV_2^3 \wedge T_3^4(SC_{g_2}) \rightarrow T_1^4(SC_g) \tag{6}$$

By applying this proof schema to goal mg_{arm}, the decomposition of which is presented in figure 3(a) (and assuming that the context of mg_{arm} is true), we obtain the following PO, which is obviously true:

$$\begin{array}{l}(true) \wedge (G_2(x_{R2}, y_{R2}) = clean \wedge busy_2) \wedge (G_3 = G_2 \wedge busy_3 = busy_2) \wedge \\ (G_4(x_{R2}, y_{R2}) = clean \wedge \neg busy_4) \rightarrow (G_4(x_{R2}, y_{R2}) = clean \wedge \neg busy_4)\end{array} \tag{7}$$

Please notice that to simplify formulae, the universal quantification of the variables occuring in the formula is omitted.

3 Specifying holonic agents with GDT4MAS

3.1 Introduction

Two main reasons lead us to be interested in holonic multi-agent systems:

- In GDT4MAS model, an agent has only one main achievement goal. Specifying an agent with several achievement goals is possible but it is not easy to ensure a real interleaving between the actions contributing to the execution of the different goals.
- In our research team, robots with several actuators being able to act concurrently are used. Such systems cannot be specified by the GDT4MAS model.

On another hand, a main characteristic of our model is its compositional aspect, that is a feature that may be reinforced by using holonic agents. However to keep our proof system valid for the rest of the MAS, a super-agent must still look like another GDT4MAS agent and thus:

- It must have an invariant.
- It must have a unique main achievement goal.
- It may have leads-to properties.
- Its recursive structure must not be visible by the other agents in the system.

As a consequence, here is how super-agents have been introduced in GDT4MAS: *Parallel Decomposition Operators* that decompose a goal of a super-agent into several sub-agents have been defined. As in other holonic systems, each sub-agent has its own goals but must contribute to the resolution of a goal of the super-agent in which it is included.

A sub-agent being also an agent, it must be situated in an environment. This environment clearly depends upon its super-agent. Indeed, a super-agent type $ta = (n, V_i, V_s, V_{\mathcal{E}}, init, I_a, L, S_a, a, B)$ situated in an environment $e = (V_e, I_e, S_e)$ can be seen as an environment $e_a = (V_{e_a}, I_{e_a}, S_{e_a})$ for its sub-agent types with: $V_{e_a} = V_i \cup V_s \cup V_{\mathcal{E}}$, $I_{e_a} = I_a \wedge I_e$ and $S_{e_a} = S_a \cup S_e$.

The main consequences of this definition are the following:

- Internal and surface variables of a super-agent are considered as environment variables for its sub-agents.
- Environment variables of *ta* that are not seen by an agent a of type *ta* cannot be seen by sub-agents of a.

Example 10 *[Environment of the arm of R2] The arm of R2 is a subagent of R2. So, it is situated in an environment* $(\{G, x_{R2}, y_{R2}, status, T\}, (G \in x_{min}..x_{max} \times y_{min}..y_{max} \rightarrow \{clean, dirty\} \wedge x_{R2} \in x_{min}..x_{max} \wedge y_{R2} \in y_{min}..y_{max} \wedge status \in \{up, down\} \wedge T \in 0..10), S)$*. Please notice that we do not detail stable properties S).*

A last important point about sub-agents is that they have their autonomy during the execution of their goals. However, their super-agent can control a part of their execution:

- It decides when its sub-agents begin to execute their behaviour.
- Although a sub-agent can ends its behaviour by itself, its super-agent can also end it.

3.2 Parallel Decomposition Operators

Standard decomposition operators presented above decompose a goal of an agent a into subgoals that are executed by a. A new type of decomposition operators that decompose a goal of an agent a into several sub-agents must be defined (these sub-agents are graphically represented by hexagons). More formally:

Definition 3.1 (Parallel Decomposition Operator (PDO)) *Let A be a super-agent and* $scca = \{a_i, 1 \leq i \leq n\}$ *the set of its sub-agents. A Parallel Decomposition Operator is an operator allowing the decomposition of a goal G of A into a list of pairs* (SC_j, a_j) *where* a_j *is an agent of scca and* SC_j *is the satisfaction condition of the main goal of* a_j*.*

Several PDOs are presented informally here. Their formal semantics are presented in section 3.3. In the following definitions, A is the super-agent, G is the goal decomposed with the PDO, and *ca* is the set of sub-agents associated to the PDO.

Definition 3.2 (ParAND PDO) *When A has to achieve G, all the agents of ca start the execution of their behaviour. As soon as one of these agents ends with failure (it fails to achieve its main goal), A stops all the other agents of ca and ends the execution process of G. G may be achieved or not. If all the agents of ca end successfully, the execution of G ends and G is achieved.*

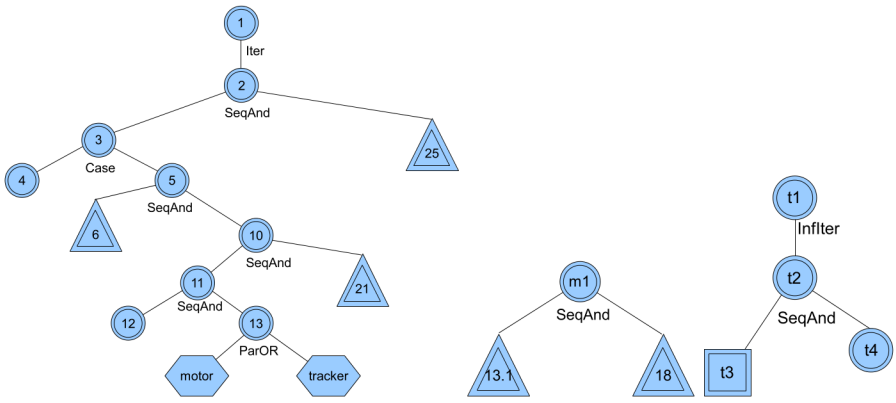
Definition 3.3 (ParOR PDO) *When A has to achieve G, all the agents of ca start the execution of their behaviour. As soon as one of these agents ends successfully, A stops all the other agents of ca, ends the execution process of G and G is achieved. If all the agents of ca end with failure, the execution of G ends. G may be achieved or not.*

Definition 3.4 (SyncParAND) *SyncParAND PDO (Synchronized Parallel And) is a synchronized version of the ParAND PDO. It means that disjoint subsets of the environment variables perceived by A can be associated to each agent of ca. These variables are locked during the execution process of G in order to guarantee that at most one sub-agent can modify them. Moreover, no other agent in the system can modify these variables during the execution of G.*

Definition 3.5 (ParCondOR) *ParCondOR PDO (Parallel Conditional Or) allows a decomposition of G into a set of pairs* $sca = \{(SC_j, a_j)\}$ *(with* $ca = \cup(a_j)$*). A condition* c_j *(a predicate) is associated to each of these pairs. The behaviour of agent* a_j *is executed only if* c_j *is true when G begins to be executed by A. As soon as one of the started sub-agents ends with success, goal G is achieved. Moreover, if all the started sub-agents end with failure, the execution of G also ends.*

Definition 3.6 (ParCondAND) *ParCondAND PDO (Parallel Conditional And) allows a decomposition of G into a set of pairs* $s = \{(c_j, sa_j)\}$*.* c_j *is a condition (predicate) and* sa_j *is a set of pairs* (SC_{j_k}, a_{j_k}) *with the same meaning as above. Behaviours of agents in* a_{j_k} *are executed only if* c_j *is true when G begins to be executed. If all the started sub-agents associated to a given condition* c_j *end with success, goal G is achieved. On the contrary, if one of these sub-agents ends with failure, the executions of the other agents associated to* c_j *end. If this occurs for all the conditions associated to goal G, goal G ends.*

Example 11 *[Usage of PDOs] We can now give the behaviours of R1 (and as a consequence of its sub-agents,* motor *and* tracker*), as well as the behaviours of R2 and of the whole incineration system (figures 4 and 5). The pGDT associated to type R1 is summarized in figure 4(a). Some parts, summarized by a triangle, are not detailed here. The complete description can be found in [Mermet and Simon (2009)]. For the purpose of this article, we only precise that goal 13 is to bring the carried waste to the nearest robot R2. The behaviour of the motor is presented in figure 4(b). It is a part of the behaviour of R1 in the ERoM problem given in [Mermet and Simon (2009)] and is not detailed here. The behaviour of the tracker is presented in figure 4(c). As it is a standard usage of the design pattern Combining cognitive and reactive behaviours, it will be explained in section 4.*

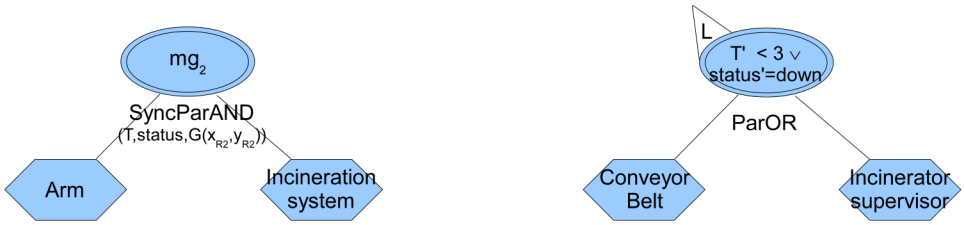


(a) pGDT of type $R1$ (b) pGDT of the motor (c) pGDT of the tracker

Figure 4 pGDTs of $R1$ and its sub-agents

3.3 Formal semantics of PDOs

As explained above, temporal variables have been introduced to associate a formal semantics to standard goal decomposition operators. To specify PDOs, these



(a) pGDT of $R2$ (b) pGDT of the incineration system

Figure 5 pGDTs of $R2$ and of its incineration subsystem

variables are used again, but a new one, named $ended_{g,i}$, must be defined for each sub-agent $(a_i)_{1\leq i\leq n}$ (with a main goal mg_i) of a parent goal g. This variable is true if sub-agent a_i has ended the execution of its behaviour since the beginning of the execution process of g. The value of this variable can be computed with the following temporal formulae, explained in [Simon and Mermet (2009)]:

$$\Box(\forall i \in [1..n].(init_g \rightarrow \neg ended_{g,i})) \tag{8}$$
$$\Box(\forall i \in [1..n].(\neg ended_{g,i} \wedge o(\neg end_{mg_i}) \wedge o(\neg end_G) \rightarrow o(\neg ended_{g,i}))) \tag{9}$$
$$\Box(\forall i \in [1..n].(end_{mg_i} \rightarrow ended_{g,i})) \tag{10}$$
$$\Box(\forall i \in [1..n].(ended_{g,i} \wedge in_g \rightarrow o(ended_{g,i}))) \tag{11}$$

In a similar way, for each sub-agent a_i a variable $started_{g,i}$ is defined: it is true when sub-agent a_i has begun to execute its behaviour, even if it has ended:

$$\Box(\forall i \in [1..n].(init_g \wedge \neg init_{mg_i} \rightarrow \neg started_{g,i})) \tag{12}$$
$$\Box(\forall i \in [1..n].(in_g \wedge init_{mg_i} \rightarrow started_{g,i})) \tag{13}$$
$$\Box(\forall i \in [1..n].(\neg started_{g,i} \wedge o(\neg init_{mg_i}) \rightarrow o(\neg started_{g,i}))) \tag{14}$$
$$\Box(\forall i \in [1..n].(started_{g,i} \wedge in_g \rightarrow o(started_{g,i}))) \tag{15}$$

Formula 12 specifies that when the execution of g begins, if sub-agent a_i does not start, it is said to be *non-started*. Formula 13 specifies that during the execution of goal g, if sub-agent a_i starts, it is said to be *started*. Formula 14 specifies that a sub-agent cannot be considered as *started* magically. Finally formula 15 specifies that a *started* sub-agent remains *started* until the execution of g ends.

A formal semantics can now be given to each PDO. The formal semantics of the parAND operator is not given here but can be found in [Simon and Mermet (2009)]. The other ones are presented in tables 1 to 3.

$$\Box\left(init_g \rightarrow \bigwedge_{i=1}^{n}(init_{mg_i})\right) \tag{16}$$
$$\Box\left(\forall i \in [1..n].\,(ended_i) \wedge \forall i \in [1..n].\,(end_{mg_i} \rightarrow \neg sat_{mg_i}) \rightarrow o\,(end_g)\right) \tag{17}$$
$$\Box\left(\forall i \in [1..n].\left(\begin{array}{l} end_{mg_i} \wedge sat_{mg_i} \\ \rightarrow \\ (o(end_g \wedge sat_g) \wedge \forall j \in [1..n].(\neg ended_j \rightarrow o(end_{mg_j}))) \end{array}\right)\right) \tag{18}$$

Table 1 Formal semantics of the parOR PDO

Formula 16 specifies that when parent goal g must be achieved, all the sub-agents start their execution. Formula 17 specifies that if all the sub-agents end with failure, then execution process of g also ends. Finally, formula 18 specifies that as soon as a sub-agent ends with success, then execution process of g ends with success and the other sub-agents also end.

$$\Box\left(init_g \rightarrow \forall a \in agents(g).\left(truth_{cond_a} \rightarrow init_{mg_a}\right)\right) \tag{19}$$

$$\Box\left(\forall a \in agents(g).\left(\begin{array}{l} end_{mg_a} \wedge sat_{mg_a} \rightarrow \\ \left\{\begin{array}{l} end_g \wedge sat_g \\ \forall b \in agents(g).\{started_b \wedge \neg ended_b \rightarrow o\left(end_{mg_b}\right)\} \end{array}\right. \end{array}\right)\right) \tag{20}$$

$$\Box\left(\forall a \in agents(g).\left(\left(started_a \rightarrow ended_a\right)\right) \wedge \neg end_g \rightarrow o(end_g)\right) \tag{21}$$

Table 2 Formal semantics of the parCondOR PDO

In table 2, the following notations are used: $cond_a$ is the condition associated to sub-agent a, $agents(g)$ is the set of sub-agents associated to goal g, mg_a is the main goal of agent a and $truth_c$ is the valuation of condition c. Formula 19 specifies that when g must be achieved, all the agents attached to a *true* condition begin to execute their behaviour. Formula 20 specifies that as soon as one of the started sub-agent ends with success, g is achieved and the other sub-agents are stopped. Finally, formula 21 specifies that if all the started sub-agents end with failure, then the parent goal also ends.

$$\Box\left(init_g \rightarrow \forall c \in cond(g).\left(truth_c \rightarrow \forall a \in agents(c).\left(init_{mg_a}\right)\right)\right) \tag{22}$$

$$\Box\left(\begin{array}{l} \exists c \in cond(g).\left(\left\{\begin{array}{l} \forall a \in agents(c).\left(ended_a\right) \\ \forall a \in agents(c).\left(end_{mg_a} \rightarrow sat_{mg_a}\right) \end{array}\right.\right) \\ \rightarrow \\ \left\{\begin{array}{l} o(end_g) \\ o(sat_g) \\ o\left(\forall c \in cond(g).\left(\forall a \in agents(c).\left(started_a \rightarrow ended_a\right)\right)\right) \end{array}\right. \end{array}\right) \tag{23}$$

$$\Box\left(\forall c \in cond(g).\left(\forall a \in agents(c).\left(\begin{array}{l} end_{mg_a} \wedge \neg sat_{mg_a} \rightarrow \\ \forall a \in agents(c).\left(o(end_{mg_a})\right) \end{array}\right)\right)\right) \tag{24}$$

$$\Box\left(\left.\begin{array}{l} \forall c \in cond(g).\left(\forall a \in agents(a).\left(started_a \rightarrow ended_a\right)\right) \\ \neg end_g \end{array}\right\} \rightarrow o\left(end_g\right)\right) \tag{25}$$

Table 3 Formal semantics of the parCondAND PDO

In table 3, the following notations are used: $cond(g)$ is the set of conditions associated to the parCondAND operator, $agents(c)$ is the set of agents associated to condition c, mg_a is the main goal of agent a and $truth_c$ is the valuation of condition c. Formula 22 specifies that when goal g must be achieved, all the sub-agents associated to a true condition begin to execute their behaviour. Formula 23 specifies that if all the sub-agents associated to a given condition have ended with success, then the parent goal ends with success and other sub-agents are stopped. Formula 24 specifies that if a sub-agent ends with a failure, all the sub-agents attached to the same condition end. Finally, formula 25 specifies that if all the sub-agents ends with failure, parent goal also ends.

3.4 Proof schemas

From the formal semantics given above, we have defined a proof schema for each PDO, using notations T_i^j, At^i, ISV_i^j and $ISLV_i^j$ presented in section 2.6. Moreover, we also write C_g the context of a goal g and TC_a the triggering context of the GDT of an agent a.

In the formal semantics of *parAND* operator, it is specified that if all the subagents end with success, the parent goal is achieved. This must be verified by the proof process. Moreover, it is also specified that when a goal g is decomposed with a *parAND* operator, all the subagents start their behaviour, which means that their triggering context is true. These two properties lead us to write the proof schema presented in table 4 for the parAND operator, where g is the goal decomposed thanks to a *parAND* operator into a set $\{a_i\}_{1\leq_1\leq n}$ of n agents. SC_i represents the satisfaction condition of the main goal of agent a_i.

$$C_g \rightarrow \bigwedge_{i=1}^{n} (TC_{a_i}) \tag{26}$$

$$At^0(C_g) \wedge \bigwedge_{i=1}^{n} \left(T_0^i(SC_i)) \wedge ISV_i^{(n+1)}\right) \rightarrow T_0^{(n+1)}(SC_g) \tag{27}$$

Table 4 Proof schema for the parAND PDO

The proof schema associated to the syncParAND Operator is quite the same, but values of *locked* variables are preserved. It is given in table 5.

$$C_g \rightarrow \bigwedge_{i=1}^{n} (TC_{a_i}) \tag{28}$$

$$At^0(C_g) \wedge \bigwedge_{i=1}^{n} \left(T_0^i(SC_i)) \wedge ISLV_i^{(n+1)}\right) \rightarrow T_0^{(n+1)}(SC_g) \tag{29}$$

Table 5 Proof schema for the syncParAND PDO

Example 12 *[Application of the SyncParAND PS to the main goal of R2] We have to recall that:*

- *The triggering context of R2 is $G(x_{R2}, y_{R2}) = dirty$ and its precondition is true.*
- *The context of the main goal of an agent is the conjunction of its triggering context and its precondition. So here, $C_{mg_2} \equiv (G(x_{R2}, y_{R2}) = dirty)$.*
- *The triggering context of the arm is $G(x_{R2}, u_{R2}) = dirty$.*
- *The triggering context of the conveyor belt is true.*

The proof schema of the SyncParAND PDO given in table 4 leads to generate automatically the following proof obligations:

$$G(x_{R2}, y_{R2}) = dirty \rightarrow (G(x_{R2}, y_{R2}) = dirty \wedge true) \tag{30}$$

$$G_0(x_{R2}, y_{R2}) = dirty \wedge \left\{\begin{array}{l} G_1(x_{R2}, y_{R2}) = clean \wedge \neg busy_1 \\ (T_1 < 3 \vee status_1 = down) \\ busy_2 = busy_1 \wedge G_2(x_{R2}, y_{R2}) = G_1(x_{R2}, y_{R2}) \\ status_2 = status_1 \wedge T_2 = T_1 \end{array}\right\} \tag{31}$$

$$\rightarrow$$

$$\left(status_2 = up \rightarrow \left\{\begin{array}{l} G_2(x_{R2}, y_{R2}) = clean \\ T_2 < 10 \end{array}\right\}\right)$$

Both formulae have been proven automatically by the theorem prover krt [Digilog (1997)]. The first one is obvious. The second one is true for the following reasons:

- *It is obvious for* $status_2 = down$. *So it must just be verified for* $status_2 = up$, *which means that* $status_1$ *is up also (as* $status_2 = status_1$*).*
- *If* $status_1 = up$, *then* $T_1 < 3$ *and, as* $T_2 = T_1$, $T_2 < 3$. *So,* $T_2 < 10$.
- $G_2(x_{R2}, y_{R2}) = G_1(x_{R2}, y_{R2}) = clean$ *by hypotheses.*

In the formal semantics of the parOR operator, the starting condition of the sub-agents is the same as for the parAND operator. However, the satisfaction condition of the parent goal must be satisfied as soon as one sub-agent ends with success. Hence the proof schema presented in table 6.

$$C_g \rightarrow \bigwedge_{i=1}^{n} (TC_{a_i}) \tag{32}$$

$$\forall i \in [1..n].(At^0(C_g) \wedge T_0^1(SC_i) \rightarrow T_0^1(SC_g)) \tag{33}$$

Table 6 Proof schema for the parOR PDO

The formal semantics of the parCondOR operator specifies that the success of one sub-agent establishes the achievement of the parent goal. This however requires that at least one sub-agent must be started, hence the proof schema presented in table 7, where $cond_i$ is the condition associated to sub-agent a_i.

$$C_g \rightarrow \bigwedge_{i=1}^{n} (cond_i \rightarrow TC_{a_i}) \tag{34}$$

$$\forall i \in [1..n].(AT^0(C_g) \wedge T_0^1(SC_i) \rightarrow T_0^1(SC_g)) \tag{35}$$

$$C_g \rightarrow \bigvee_{i=1}^{n} (cond_i) \tag{36}$$

Table 7 Proof schema for the parCondOR PDO

Finally, the proof schema associated to the parCondAND operator is presented in table 8. In this proof schema, $agents(i)$ is the set of agents attached to condition $cond_i$ and SC_a is the satisfaction condition of the main goal of agent a.

$$\forall i.\,(\forall a \in agents(i).\,(C_g \wedge cond_i \rightarrow TC_a)) \tag{37}$$

$$C_g \rightarrow \bigvee_{i=1}^{n} (cond_i) \tag{38}$$

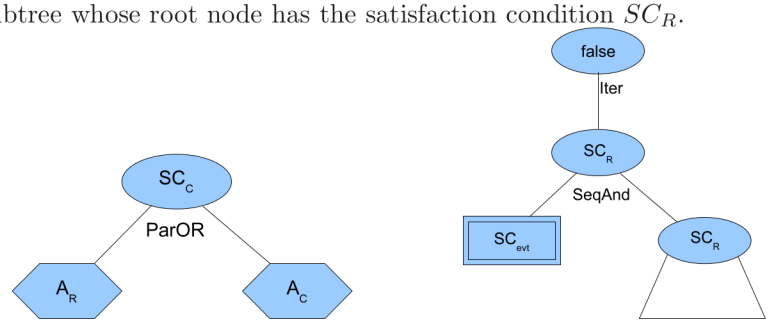
$$\forall i.\left(At^0(C_g) \wedge cond_i \rightarrow \left(\bigwedge_{a \in cond(i)} \left(T_0^i(SC_a) \wedge ISV_i^{(n+1)}\right) \rightarrow T_0^{(n+1)}(SC_g)\right)\right) \tag{39}$$

Table 8 Proof schema for the parCondAND PDO

4 Design patterns

Several design patterns that help to use PDOs have been defined [Simon and Mermet (2009)]. It has also been shown in the cited article that PDOs give a solution to the management of communications. These patterns also help the designer to determine when to use nested agents. These design patterns are briefly summarized here:

- **Several actuators** and **several achievement goals**: The architecture of the solution to these two problems consists in using a parAND operator with several sub-agents: one for each actuator or each achievement goal.
- **Roles composition**: In goal-based methods like gaia [Wooldridge et al. (2000)], roles are formally specified. Moreover, the agent model specifies which roles an agent implements. But the way roles are combined is not specified. Using a parAND operator between sub-agents, each one being dedicated to the implementation of one role, is a way to formally combine roles and to verify that these roles can be combined.
- **Task interruptible by the occurrence of an event**: In that case, a parOR operator with two agents is required. The first sub-agent tries to execute the task. The GDT of the second sub-agent consists in a main goal decomposed by a seqAND into two subgoals: the first one is an external goal waiting for the event to occur and the second one handles the event.
- **Combining cognitive and reactive behaviours**: In that case, the agent has to perform a cognitive task but, during its execution, it may have to react to external events. This can be achieved by the structure presented in figure 6, as demonstrated in [Simon and Mermet (2009)]. The agents A_C and A_R are respectively in charge of the cognitive and reactive parts. Occuring events, represented by a satisfaction condition SC_{evt} are managed by the subtree whose root node has the satisfaction condition SC_R.



(a) GDT of the main agent (b) GDT of the A_R

Figure 6 Architecture for combining cognitive and reactive behaviours

Example 13 *[Design patterns applied to our case study] The CERoM problem can be analysed in terms of design patterns given in the previous section:*

- *R2 type of agent is made of two actuators that act concurrently. So, the design pattern* Several actuators *is used to decompose the main goal of robot R2 with a* parAND *operator into two sub-agents:* arm *and* incineration system*;*
- *While the incineration system conveys wastes (its basic behaviour), it must detect abnormal beaviours of the incinerator. If this occurs, it must interrupt its basic behaviour. This typically corresponds to the design pattern* Task interruptible by the occurrence of an event*.*
- *While a robot R1 is bringing a waste to the nearest* up *robot R2, it must detect if this robot breaks down. If this occurs, it must react by modifying the target. This corresponds to the design pattern* Combining cognitive and reactive behaviours*, with two sub-agents:* motor *and* tracker*. The behaviour of the tracker given in figure 4(c) can now be explained: the satisfaction condition of its main goal is* false*, the satisfaction condition of goal t2 is the same as the one of goal t4 (this goal, that consists in finding the nearest robot of type R2 that is up, has the same SC as goal* 12 *of robot R1), and the satisfaction condition of external goal t3 expresses that the status of the target agent is* down*.*

5 Related Work

As mentioned in [Gerber et al. (1999); Schillo and Fischer (2004)] for example, the term "holon" was originally introduced in 1967 by Arthur Koestler in order to name recursive and self-similar structures in biological and sociological entities (as a human individual for example). In [Fischer et al. (2003)] "an holonic agent is defined as an agent consisting of sub-agents with the same inherent structure". From this point of view, GDT agents can be seen as specific holons because they have the recursive structure of a holon but these holons do not result from the decision of agents to merge as suggested by Fischer. The same authors define formally a holon by a triplet Head, SubHolons and Commitments. Head is a subset of subholons that represents the holon to the environment and that is responsible for coordinating the actions inside the holon. For a GDT agent, the head is, among others, in charge of the management of the end of the different sub-agents according to the PDO which has been used to activate them. But it does not represent the holon for its environment. Indeed, for other agents in the environment, the holon is an agent like the others with which they can communicate. The communication can then be assumed by a specific member of the holon. The commitments mentioned by Fischer "define the relationship inside the holon" and can be compared to the semantics of the PDO used. Nevertheless, sub-agents are not allowed to engage in several holons at the same time as mentioned by Fischer *et al.*. As a consequence, GDT agents composed of sub-agents can be compared to holons implemented by moderated associations which is one of the possibilities described in [Gerber et al. (1999); Fischer et al. (2003)]. In such holons,"agents give only part of their autonomy to the superholon". Indeed, sub-agents of GDT agents can be stopped by the super-agent (the head of the holon) but they keep their autonomy for the rest of their behaviour. Moreover, "the members of a superholon are always represented as agents and, hence, we do not lose the capability to solve problems in a distributed fashion" which is exactly what we expect from sub-agents.

In [Gerber et al. (1999)], the authors list a set of properties for agenthood applied to holonic agents:

- An agent group forming a holon acts as a single entity in its environment. This property is verified by GDT agents, whether they are composed of sub-agents or not.
- "A super-holon may have actions as its disposal that none of its sub-agents could perform alone" : GDT agents composed of sub-agents do not really have new capacities but they are able to solve problems by combination of the capacities of their sub-agents.
- "Holons have some representation of their environment. This knowledge might be represented explicitely within the super-holon". In GDT agents, the knowledge about the environment is represented by variables for instance, which are observable by sub-agents.

Some recent works deal with the problem of the design and implementation of holonic multi-agent systems together with organisational aspects. In [Gaud (2007)], the author proposes a software development process with associated tools for the analysis, design, implementation and simulation of holonic multiagent systems. This thesis introduces the CRIO metamodel allowing the analysis and the design of complex systems from a holonic and organisational point of view. The author

points out the fact that, in order for multiagent systems to be really used in industrial applications, the gap between the design and implementation phases must be as short as possible. That is why for example, we have proposed an automatic process to implement GDT agents by automata [Simon and Flouret (2006)]. The metamodel CRIO is organized in three different domains : the problem domain, the agency domain and the solution domain [Cossentino et al. (2007)]. The agency domain defines the notions of organizations which are different contexts of interaction of roles. A role, to which a goal and a behaviour are associated, requires a set of capacities to be executed. A capacity is a description of a "know-how" without any hypothesis on the agent that will perform it. In this context, a holon can play a set of roles inside different organisations which implies that it provides a set of capacity implementations allowing the roles to be performed indeed. The authors have chosen to propose an implementation of (composed) holons which implies that they must contain at least a single instance of a holonic organisation and a set of production organisations. The holonic organisation specifies how members manage the super holon. A production organisation describes "how members interact and coordinate their actions to fulfill the super-holon tasks and objectives" [Cossentino et al. (2007)]. In GDT agents, it is exactly what is specified by the semantics of the PDO which introduces the main goals of the sub-agents i.e. the members of the holon. Moreover, the organisation of the holon is also specified by the PDO : agents associated to subgoals are members of the holon to which is added a specific agent (the head) which has to control the life of the holon.

In [Gaud et al. (2008)], the same authors propose a verification by abstraction framework for such multiagent systems. The verification approach "is based upon the abstraction of capacities of roles" in the CRIO metamodel. To perform proofs, the authors propose a specification of the main concepts of CRIO using OZS [Hilaire et al. (2000)] which is a formal language combining Object Z and statecharts. These specifications have then applied to the contract net protocol. Then proofs of general properties of the protocol have been performed using bounded model checking and theorem proving by induction. The authors show that abstractions provided by capacities make the proofs more tractable. Unfortunately the specification of groups and holons are not given in the paper which prevents to evaluate if the approach is also well suited to holonic systems. The given proofs are performed only for roles which does not take into account the problem of multiple roles played by the same agent or holon. Moreover, even if general specifications of several CRIO concepts are given which can be instanciated on various systems, the approach does not allow to generate what has to be proven (what is called proof obligations for GDT agents) which is an entire part of the proof process.

Anemona [Botti and Giret (2008)] is the first multiagent methodology adapted to the analysis and design of holonic systems. However, it is specialised in holonic manufacturing systems. But the authors propose a metamodel for the design of holons based on the notion of "abstract agent". This architecture is very close to the architecture of a system based on composed GDT agents. The main difference is that, for GDT agents, an abstract agent (a composed agent) can be a member of only one multiagent system (a super-agent).

Adam et al. [Adam et al. (2008)] propose a framework allowing the design of holonic MAS adapted to the management of information in human organisations.

Holons are in particular used to dynamically add or remove roles to an agent. These additions and deletions are managed by rules corresponding to required properties. These rules can be compared to PDOs, which also specify when sub-agents are introduced, and could be certainly translated into design patterns for PDOs. These would give the possibility to verify that the composition of roles is meaningful.

In [Occello (2000)], a recursive specification of MAS is proposed, leading to MAS which are not far from holonic MAS because of the recursive structure of agents. However, in this approach, even the environment is specified recursively because it is considered as an entire part of the specified MAS.

6 Conclusion

This paper describes how the GDT4MAS model can be modified to allow the specification and the verification of multi-agent systems with compound agents.

First of all, it has been shown that such MAS can be seen as particular kinds of holonic multi-agent systems. Indeed, as holons, compound agents are perceived as classical ones by other agents. However, compound agents are particular holons which are their own representative. Moreover, in such systems, sub-holons can not be part of several super-holons at the same time. Last but not least, these holons are not created by agents which have decided to merge but by agents which sometimes need sub-holons to perform their behaviour.

The initial GDT4MAS model, where agents behaviour are described by GDTs, is presented. It is then shown how the introduction of compound agents (holons) in this model makes it more expressive by allowing the specification of parallel behaviours. These specifications are based on specific goal decomposition operators called PDOs which can be used in the GDT of the compound agent. It is also shown that this extension increases the compositionality of the proof system associated to GDT4MAS. Indeed, proving the correctness of the behaviour of a compound agent is based on proofs of the correctness of the behaviour of sub-agents. But, if sub-agents are not themselves compound agents, these last proofs are the same as the proof of the correctness of the behaviour of a classical agent. Indeed, in the model, a sub-agent is an agent whose environment is the compound agent it belongs to. So proof schemas proposed with GDT4MAS can be used again. However, new proof schemas have been defined for PDOs in order to prove the correctness of this kind of decompositions.

These propositions have been used in a case study which is an extension of the RoM problem. The specification of this problem with GDT4MAS and a part of the proof of its correctness are detailed in the paper.

As a consequence, the new GDT4MAS model allows the specification of particular holonic MAS and the verification of their correctness which is an important aspect for the holonic MAS community.

Our model lacks however of CASE tools to support it. The development of such tools is a work in progress and it should help us apply our method to more complex case studies. Another perspective consists in specifying more design patterns, in particular to model various kinds of communications.

References

Abrial, J.-R. (1996). *The B-Book*. Cambridge Univ. Press.

Adam, E., Strugeon, E. G.-L., and Mandiau, R. (2008). Flexible hierarchical organisation of role based agents. In *2nd IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshops*, pages 186–191.

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

Botti, V. and Giret, A. (2008). *ANEMONA: A multi-agent methodology for Holonic Manufacturing Systems*. Springer.

Cossentino, M., Gaud, N., Galland, S., Hilaire, V., and Koukam, A. (2007). A metamodel and implementation platform fo holonic multi-agent systems. In *EUMAS*.

Dennis, L., Fisher, M., and Hepple, A. (2008). Language constructs for multi-agent programming. In *Computational Logic in Multi-Agent Systems: 8th International Workshop, CLIMA VIII, Porto, Portugal, September 10-11, 2007. Revised Selected and Invited Papers*, pages 137–156.

Digilog (1997). Atelier B, Guide de l'utilisateur v3.0. Technical report, Digilog.

Fischer, K., Schillo, M., and Sieckmann, J. (2003). Holonic Multiagent Systems: A Foundation for the Organisation of Multiagent Systems. In Springer, editor, *HoloMAS*, volume 2744 of *LNCS*, pages 71–80.

Gaud, N. (2007). *Systèmes multi-agents holoniques : de l'analyse à l'implantation*. PhD thesis, Université de Technologie de Belfort-Montbéliard.

Gaud, N., Hilaire, V., Galland, S., Koukam, A., and Cossentino, M. (2008). A verification by abstraction framework for organizational multi-agent systems. In *AT2AI: Froma Agent Theory to Agent Implementation*.

Gerber, C., Siekmann, J., and Vierke, G. (1999). Flexible autonomy in holonic agent systems. In Musliner, D. and Pell, B., editors, *AAAI Spring Symposium on Agents with Adjustable Autonomy*, pages 52–58.

Hilaire, V., Koukam, A., Gruer, P., and Müller, J.-P. (2000). Formal specification and prototyping of multi-agent systems. In Omicini, A., Tolksdorf, R., and Zambonelli, F., editors, *Engineering Societies in the Agents World*, volume 1972.

Mermet, B. and Simon, G. (2009). GDT4MAS: an extension of the GDT model to specify and to verify MultiAgent Systems. In *AAMAS*, pages 505–512.

Mermet, B. and Simon, G. (2010). The CERoM case study. http://users.info.unicaen.fr/~bmermet/GDT/publications/gdt/etudeDeCasIjaose09.pdf.

Mermet, B., Simon, G., Saval, A., and Zanuttini, B. (2007). Specifying, verifying and implementing a MAS: A case study. In Dastani, M., Segrouchni, A. E. F., Ricci, A., and Winikoff, M., editors, *Post-Proc. 5th International Workshop on Programming Multi-Agent Systems (ProMAS'07)*, number 4908 in LNAI, pages 172–189. Springer.

Occello, M. (2000). Towards a generic recursive agent model. In *Int. Conf. on Artificial Intelligence*, pages 649–654.

Schillo, M. and Fischer, K. (2004). Holonic multiagent systems. *Zeitschrift fr Künstliche Intelligenz*, 3.

Simon, G. and Flouret, M. (2006). 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.

Simon, G. and Mermet, B. (2009). Spécifier des agents composés d'agents avec les GDT. In Guessoum, Z. and Hassas, S., editors, *JFSMA*.

Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312.