Using GDT4MAS as a Formal Support for Engineering Multi-Agents Systems

Bruno Mermet and Gaële Simon
GREYC-UMR 6072 and University of Le Havre, Université de Caen Basse-Normandie,
Campus Côte de Nacre, Boulevard du Maréchal Juin, 14032, CAEN cedex 5, France

Keywords: Agent-oriented Software Engineering, Formal Specification, Formal Verification.

Abstract: This paper focuses on multi-agent systems engineering process. An assessment of current needs in this domain, based on the analysis of systems already developed, is performed. This assessment shows that the formal verification of MAS is one of these needs. It is then shown how the formal approach GDT4MAS provides answer to many of the other needs. This approach is based on a MAS formal specification associated to a proof process allowing to establish the correctness of properties of the system. The main purpose of this paper is to show that, unlike most other formal approaches for MAS, GDT4MAS can at the same time propose formal aspects making a proof possible and contribute to different general aspects of agent-oriented software engineering, even when formal verification is not a concern.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Most failures in software development are a consequence of miss-adapted methodologies or of the weaknesses of tools that are used. In the domain of standard software, recent years led to the design of new methodologies (relying on the agile concepts) and to the development of new tools to reduce the ratio of unsucessful developments (Beck et al., 2001; Martin, 2009).

However, as it has been explained in other works (Ahmad and Rahimi, 2009), Multi-Agent Systems are specific softwares and require their own methods and tools. Moreover, we are mainly interested in formal specification and formal verification, and these techniques, which are also very specific, require their own methods and tools.

This led to the design of the GDT4MAS method and model, presented in (Mermet and Simon, 2009). GDT4MAS proposes its own language to specify, design and implement MAS. But, as this model relies on concepts that are shared by most MAS architectures, it provides characteristics that may be useful for anyone interested in the development of MAS, even for people that are not interested in formal methods.

The goal of this article is then to show how GDT4MAS tackles many requirements that are exhibited by several works dealing with Agent Oriented Software Engineering and why using GDT4MAS may help MAS designers.

In the next section, we present the concepts of GDT4MAS that must be known to understand this article. In section 3, we explain why it may be interesting to verify Multi-Agent Systems, and we show why GDT4MAS is a good candidate to perform such a task. Then, we explain how this model helps in the development process where several kinds of stakeholders take place. Section 5 presents how incremental development can be instanciated thanks to GDT4MAS. In the next section, it is shown how several kinds of MAS architectures can be modeled thanks to the expressiveness of GDT4MAS.

2 GDT4MAS

2.1 Main Concepts

We only summarize here the essential parts of GDT4MAS.More details can be found in (Mermet and Simon, 2009; Mermet and Simon, 2011; Mermet and Simon, 2013).

When specifying MAS with GDT4MAS, 3 parts have to be specified: the environment, the types of agents and the agents themselves, that are instances of each type of agent, with specific initialisation values. In the sequel, we briefly present these different parts.

The environment is specificied by a set of typed variables and an invariant property $i_{\mathcal{E}}$.

The type of an agent is specified by a set of typed variables, an invariant and a behaviour.

The behaviour of an agent is mainly defined by a *Goal Decomposition Tree (GDT)*. The GDT is a tree of goals, whose root corresponds to the main goal of the agent (in the standard version of GDT4MAS, agents have only one main goal). A plan is associated to each goal. Such a plan, when executed with success, must achieve the goal and is expressed either by a single action of by a set of subgoals linked together by a *decomposition operator*.

A goal G is mainly described by a name n_G, a satisfaction condition sc_G and a guaranted property in case of failure gpf_G.

The satisfaction condition (SC) of a goal is specified formally by a formula that is satisfied when the execution of the goal succeeds. On the other hand, the GPF of a goal specifies what happens when a goal execution fails (of course, it is meaningless for an NS goal, that is to say a goal that always succeed).

SC and GPF are *state transition formulae* (STF), because they express a relation between two states, called initial state and final state. In the sequel, we will use the term *non-deterministic state transition formula* when for a given initial state, several final states satisfy the STF. For example, formula $x' > x$ is a non-deterministic STF because, for a given initial state ($x = 0$ for instance), several final states ($x' = 2$, $x' = 10$) satisfy the formula.

2.2 GDT Example

Figure 1 shows an example of GDT. The goal of this behaviour is to light a given room n (n is a parameter of the GDT). In order to do that, the agent tries to enter into the room. As a cellular eye detects when someone enters into the room and switches the light on, this looks like a suitable plan. However, if the cellular eye does not work as expected (this is why the goal *Entering into the room* is NNS, i.e. not NS), the agent will have to use the switch. More details can be found in (Mermet and Simon, 2013)

2.3 Agents

Agents are specified as instances of types of agents, with effective values for the agent type parameters.

2.4 Proof Principles

The proof mechanism provided by GDT4MAS aims at proving the following properties: agents preserve invariant properties (Mermet and Simon, 2013), agents behaviour are sound, that is to say, plans associated to goals are correct and agents achieve liveness properties that may be associated to their agent type.

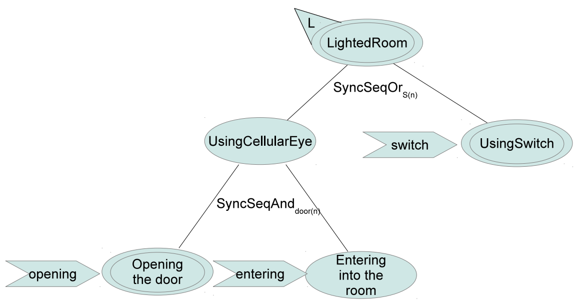

Figure 1: Example of a GDT.

Moreover, this proof mechanism relies on a few important principles: proof obligations (properties to be proven) can be generated automatically from a GDT4MAS specification, proof obligations are expressed in first-order logic and can be verified by any adequate automatic theorem prover and finally, the proof system is compositional: the proof of the correctness of an agent is decomposed into several small independent proof obligations.

2.5 Holonic GDT4MAS

An extension of GDT4MAS has been presented in (Mermet and Simon, 2011), where it has been shown that the execution of the goal of an agent can be performed thanks to several sub-agents (a holonic agent is itself made of sub-agents).

Figure 2 illustrates the usage of the *ParAnd* operator. The root goal in this figure corresponds to one of the goals of a holonic agent. To achieve this goal, two sub-agents, A_1 and A_2, represented by hexagons, are started by the holonic agent. A_1 (whose behaviour is described by a GDT that is not presented here) tries to achieve its main goal, whose satisfaction condition is SC_1 and A_2 tries to achieve a goal whose satisfaction condition is SC_2. If either A_1 or A_2 fails, the execution of the other sub-agent is stopped. But if both sub-agents achieve their main goal, then the parent goal is achieved.

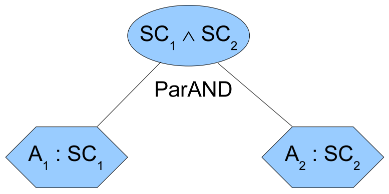

Figure 2: Example of a holonic agent.

3 VERIFICATION OF AGENT BEHAVIOURS

For more than ten years, many papers have focused on the necessity to bring guarantees on the correctness of multi-agents systems. Indeed, it appears that it is one of the main reasons explaining the small acceptance of MAS in industry (Ahmad and Rahimi, 2009). Moreover, it is also established that formally verifying MAS is certainly the adequate solution (Demazeau, 2004).

However, papers dealing with formal specification of MAS are scarce, for several reasons:

- formally verifying MAS is a very hard problem. Indeed, formal verification of standard software is already a complex problem, and solutions provided by formal methods cannot be applied to whole huge systems. So, as MAS are more complex systems, trying to formally verifying them seem vain for many people.
- Most MAS formal verification systems consider verifying the whole system behaviour (as METATEM for instance (Fisher, 2006)), which is certainly a too complex task on real-size systems. On Standard softwares, formal verification generally concern only the safety core of the sofware (Abrial, 1996); So, trying to apply formal verification techniques to whole MAS is likely to lead to failure.
- Many formal verification works use a dedicated specification language that is only used for the specification and for the verification and then, the system must be implemented in a traditional programming language. As a consequence, they may be few links between the verified specification and the implemented program (Ahmad and Rahimi, 2009; Abrial, 1996).
- Trying to reuse standard formal verification methods to MAS leads to defeat. In particular, the autonomy of agents is not consistent with the main concepts of these tools: indeed, standard formal methods rely on the fact that the whole system is known when the verification is performed (Abrial, 1996). This is not suitable to MAS, where autonomous agents can be part of the system.
- most work so far dealing with formal verification of MAS rely on model-checking (Raimondi and Lomuscio, 2004; Kacprzak et al., 2004), which is not the most suitable choice in most cases. Indeed, model-checking is a verification mechanism that is mainly dedicated to finite-state systems, as it consists in an exhaustive test of the system. But in multi-agent systems, as agents actions are interlaced, studying the whole set of possible traces is most of the time not feasible.

GDT4MAS clearly proposes a solution to the first problem, as it is a method designed to perform formal verification.

Moreover, when using GDT4MAS, it is easy to formally specify and verify safety parts, while not verifying less essential parts. Of course, in this latter case, the correctness is established under the assumption that non-formalized parts are correct. This clearly provides a solution to the second problem.

An other important characteristics of GDT4MAS is that a generalized translation system of a GDT4MAS specification in any imperative language using automata has been designed (and formally verified), and a version for Java as been implemented. So, the verified specification can be automatically translated into Java to be executed, or to interact with other Java programs. As a consequence, the formal specification need not to be transcripted by hand in a programming language.

In addition, as GDT4MAS has been designed to verify MAS, the autonomy of agents is well taken into account. Verification of agents are performed independently of the other agents, and only proofs depending on agents interactions (based on external goals) take into account the instances of agent types.

Finally, GDT4MAS relies on theorem proving, that seems, for us, to be a quite better solution than model-checking. Indeed, model-checking is well suited for finite-state systems, which is not true for MAS. When considering the potential huge number of agents in a MAS, and the massive distributed aspect that it implies, it greatly reduces the potential applications of techniques relying on model-checking.

To conclude, GDT4MAS proposes a wide-purpose method, going from the early specification to the executable code via formal verification.

4 SEPARATION OF ENGINEERING TASKS

During the whole development process of a software, several tasks have to be performed, and these tasks may be devoted to different persons, according to their skills. In standard software development processes, the following tasks are often distinguished: requirement engineering, design (preliminary and detailed), implementation and test. When dealing with formal verification, an other task must be added: the verification task. However, the way the result of each task is used by the others is often fuzzy.

Distinguishing tasks and roles is a very important point in a software development process, for two main reasons:

- it helps in scheduling the development process and in parallelizing work;
- it helps in attributing to each stakeholder the most suitable tasks.

When using GDT4MAS, the development process is also decomposed into several steps:

- the first step consists in determining agent types (with their capabilities) and environment variables;
- the second step consists in specifying the root goals and the variables of the agent types;
- the third step consists in progressively decomposing goals of agents into sub-goals.

These steps matched more or less the requirement engineering, preliminary design and detailed design tasks. The implementation task is missing because it may be performed automatically at the end of the third step.

The formal aspects can be handled in parallel during each step:

- during the first step, it consists in formally specifying the environment invariant and the agent actions;
- during the second step, it consists in formally specifying the main goal (satisfaction condition and gpf) of each agent type and the invariant of each agent type;
- during the third step, it consists in formally specifying each subgoal, and in verifying invariants and goal decompositions.

The fact that formal aspects may be isolated from the rest is very important. Indeed, a developer is seldom at ease with formal methods, and a method that clearly separates the formal aspects from the other tasks has greater chances to please. Indeed, an industrial firm may continue to use the skills of its employees, and only a few new ones should be hired. Moreover, when specification and verification roles are distinguished, the hardest part (the verification) can be devoted to an expert of the prover.

This segregation of formal aspects is seldom present in formal verification systems that are proposed for MAS, and this is certainly one of the major aspects that explain they are seldom used (Ahmad and Rahimi, 2009).

However, tests are not tackled by GDT4MAS. Of course, when formal methods are used, tests are less necessary. But they remain useful, as they consider also parameters that are not taken into account by the formal verification (the environment of execution, the system or the compiler for instance). So, it is the responsibilty of the development team to manage tests. But we aim at improving GDT4MAS in order to use it to help in writing test cases.

5 INCREMENTAL DEVELOPMENT

For several years, it has been established that most software development projects fail because they are badly managed (Zeller, 2009). Thus, recent development methods rely on incremental technique. This is for instance the case of the "incremental development", but this is also the basis of the Test-Driven-Development technique (Beck, 2002).

Incremental development can be considered in two opposite directions. In standard development, the incremental aspect is mainly implemented in a bottom-up way: a small compoment is first designed and developed to obtain a first prototype, and then, new components are progressively designed, developed and merged with the previous propotype.

When developing agents, it is not easy to use a bottom-up incremental development because the goal-oriented specification of agents and the notion of plan look more like a top-down specification. And this is exactly one of the main principles of GDT4MAS. Specifying the resolution of a goal by a set of subgoals and a decomposition operator directly implements a top-down method. Moreover, in GDT4MAS, two principles are essential to make this process efficient.

Firstly, decompositions of sibling nodes are independant. This facilitates the progressive design of the resolution process of different subgoals of a given goal, and this is a very important notion to facilitate incremental development.

Secondly, decompositions rely on the refinement principle. The refinement notion is a key notion in the formal specification domain. There are several definitions of refinement, but all the definitions of the *action refinement* share a common aspect that we can informally summarize by the following sentence: an action a_2 refines an action a_1 if and only if all the state changes allowed by a_2 are also allowed by a_1 (Back and Sere, 1991). For instance, $x' > x$ is refined by $x' > x+5$, which is refined by $x' = x+10$.

This implies that thanks to refinement, we can reduce indeterminism of goals, and as a consequence, it is completely possible to use indeterminism in goal specifications. In GDT4MAS, indeterminism is allowed in goals specifications via non-deterministic STF, and so, we have extended the notion of action refinement to the notion of *goal refinement*.

Refinement is a key notion to make incremental top-down development. Indeed, a top level goal can be abstract and non deterministic. It is then possible to *refine* it by decomposing it into more deterministic goals, that can then be also refined and so on. Moreover, refinement makes possible the introduction of new variables. Indeed, as the STF $x' > x$ does not specify anything on variable y, it can be refined by the following STF: $y' = 4 \wedge x' = x + y'$. It is so possible to start with very high level goals, and incrementally detail their implementation by reducing the indeterminism thanks to goal refinement/goal decomposition.

6 MODELING DIFFERENT AGENT MODELS

As explained in section 4, when using GDT4MAS, during the third development step, agent behaviours must be designed. Depending on the agent type, different models of agents may be used. GDT4MAS gives to the designer a large choice of models. The goal of this section is to show how some of them can be specified with GDT4MAS.

6.1 Specifying Agents with Goals

Whatever the model of agents is used, agents are most of the time described using the notion of goal. This is for instance the case in models relying on BDI agents, like Agentspeak (Rao, 1996). This is also the case of GDT4MAS. Moreover, goals are often divided into two categories: achievement goals and maintain goals. Both types are present in GDT4MAS.

Achievement goals are represented by standard goals of GDT4MAS. However, two types of achievement goals are distinguished: progress goals (the resulting state is expressed as a modification of the initial state, for instance $x' = x + 1$) and state-reaching goal (the resulting state is expressed in an absolute way, for instance, $x' = 5$).

The semantics given to “maintain goals” varies from one article to another. A maintain goal consists either in guaranteeing that a property remains always true or in re-establishing a property each time it becomes false. The first case can be directly expressed in GDT4MAS by invariants. In the second case, a progress goal can be expressed thanks to an agent whose main goal expresses the property to establish, and whose triggering context is the negation of this property. So, as soon as the property becomes false, the execution of the agent begins in order to re-establish the desired property.

In many agent models or agent-oriented languages, when an agent must achieve a goal, it tries a plan then perhaps another, and so on. But it is not clear to understand why the agent tries an other plan or not, according to the goal. Is it because the goal has not been achieved ? Or because the plan could not be completed ? And when a goal is removed from the goal base, is it because it has been achieved ? Or because it seems it is impossible to achieve ? Or because it is obsolete ? In GDT4MAS, we determine which goals are non-necessarily satisfiable, and the operational semantics of the GDT operators relies on the fact that a goal has been achieved or not. This criterion is seldom considered in agent specification models.

GDT4MAS takes into account that a goal may be achieved even if the execution of the plan associated to this goal (ie. its decomposition) has failed. Indeed, it has been shown that, because of side effects, this may happen (Mermet and Simon, 2009).Forgetting this fact may lead to specify incorrect behaviours.

6.2 Expressiveness of GDT4MAS

Through several articles, the expressiveness of GDT4MAS seems adapted to model most problems tackled by multi-agent systems.

For instance, it has been shown in (Mermet and Simon, 2011) that agents mixing cognitive and reactive behaviours can be easily specified thanks to the holonic extension of GDT4MAS.

The exception management is also a key issue in MAS (Klein et al., 2003), But it is rarely handled by MAS models and languages (Platon et al., 2008) like AgentSpeak for instance. However, once again, thanks to the holonic extension of GDT4MAS, this can be clearly specified with GDT4MAS, as explained in (Mermet and Simon, 2011).

Finally, many communication styles can ben specified with GDT4MAS. To our knowledge, this is the only formal model dedicated to MAS that manage this aspect of MAS. Communications are indeed ignored in other formal systems such as METATEM for instance.

7 CONCLUSION

Throughout this article, we have shown that GDT4MAS presents very interesting characteristics to help in engineering multi-agent systems. Although this model has been designed in order to formally verify MAS, its characteristics can be re-used in a wider context of MAS engineering. It is moreover interesting to notice that, most of the time, these characteristics were not preconceptions, but they are the consequence of the will of the designers of the model to have a usable formal verification system.

Of course, an important requirement to use a model to support engineering is that CASE-tools exist for this model. We have chosen not to detail this aspect in this article to focus on the characteristics of the model. But it is important to notice that GDT4MAS has been designed from the beginning to be supported by tools. For instance, a platform has been developed to execute specifications either automatically or step by step. This platform also generates proof obligations in a format compatible with the automatic prover PVS. Morover, a web application is being developed to help in editing GDT4MAS specifications.

In order to increase the use of Multi-Agent Systems, it is crucial to provide models that help in designing and implementing such complex systems, and it is also important to provide techniques that increase the confidence in such systems. GDT4MAS is certainly a good candidate for that.

REFERENCES

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

Ahmad, R. and Rahimi, S. (2009). Motivation for a new formal framework for agent-oriented software engineering. *IJAOSE*, 3(2/3):252–276.

Back, R.-J. and Sere, K. (1991). Stepwise refinement of action systems. *Structured Programming*, 12(1):17–30.

Beck (2002). *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Beck, K. et al. (2001). Manifesto for agile software development.

Demazeau, Y., editor (2004). *Systmes Multi-Agents*, volume 29 of *ARAGO*. OFTA.

Fisher, M. (2006). Metatem: The story so far. In *Proceedings of the Third International Conference on Programming Multi-Agent Systems*, ProMAS'05, pages 3–22, Berlin, Heidelberg. Springer-Verlag.

Kacprzak, M., Lomuscio, A., and Penczek, W. (2004). Verification of multiagent systems via unbounded model checking. In *AAMAS'04*.

Klein, M. et al. (2003). Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):179–189.

Martin, R. C. (2009). *Clean Code: A handbook of agile software craftsmanship*. Prentice Hall.

Mermet, B. and Simon, G. (2009). GDT4MAS: an extension of the GDT model to specify and to verify Multi-Agent Systems. In *et al.*, D., editor, *Proc. of AAMAS 2009*, pages 505–512.

Mermet, B. and Simon, G. (2011). Specifying recursive agents with gdts. *Autonomous Agents and Multi-Agent Systems*, 23(2):273–301.

Mermet, B. and Simon, G. (2013). A new proof system to verify gdt agents. In Zavoral, F., Jung, J. J., and Badica, C., editors, *IDC*, volume 511 of *Studies in Computational Intelligence*, pages 181–187. Springer.

Platon, E., Sabouret, N., and Honiden, S. (2008). An architecture for exception management in multiagent systems. *IJAOSE*, 2(3):267–289.

Raimondi, F. and Lomuscio, A. (2004). Verification of multiagent systems via orderd binary decision diagrams: an algorithm and its implementation. In *AAMAS'04*.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In van Hoe, R., editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands.

Zeller, A. (2009). *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann, Burlington, MA, 2 edition.

SCIENCE AND TECHNOLOGY PUBLICATIONS