Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 1717–1722

DOI: 10.15439/2015F300
ACSIS, Vol. 5

Model Checking of Multi Agent System Architectures Using BigMC

Ahmed Taki Eddine Dib
Faculty of New Information Technologies and Communication
LIRE Laboratory, University of Constantine II.
Nouvelle Ville Ali Mendjeli - BP : 67A,
Constantine – Algeria
Email: dibtaki@gmail.com

Zaidi Sahnoun
Faculty of New Information Technologies and Communication
LIRE Laboratory, University of Constantine II.
Nouvelle Ville Ali Mendjeli - BP : 67A,
Constantine – Algeria
Email: sahnounz@yahoo.fr

***Abstract*— Formal methods offer a great potential for early integration of verification in the design process. These are based on theories and mathematical notations that allow the formal specification of a program and check its implementation. They offer a global vision and a high-level structure and system organization. In addition, the software architecture plays a key role as a pivot point between the requirements of a system and its implementation. In this paper, we present a formal approach based on Bigraphical Reactive Systems for specifying and verifying the main features of the Multi Agent Systems (MAS) architectures based on the Belief-Desire-Intention (BDI) agent model. The proposed approach supports both the static and dynamic aspects of BDI-MAS architectures at different levels of abstraction. Further, we use automatic proof tool BigMc to analyze the specifications and verify system properties.**

Keywords: Multi-Agent Systems, software architecture description language, Bigraphical Reactive System, formal specification, reconfiguration, formal verification, Bigraphical Model Checker.

I. Introduction

The emergence of large-scale IT networks has given rise to numerous distributed applications. These applications require a strong interaction between different entities distributed on the network that may share the same resources and the same goals. Several distributed development models for these applications have been proposed in the literature. However, the importance of the issue, is to be convinced of the legitimacy and trust granted to IT applications. These concerns have led to methods of development and verification. Lately software systems tend to be more distributed, open and concurrent. This evolution of computing has changed the way of thinking but also the design of such systems. Multi-Agent Systems (MAS) are particularly suitable for developing these kinds of systems. However, the diversity and complexity of the basic concepts that characterize multi-agent systems involve a difficulty in understanding and designing of such systems.

Formal methods offer a great potential for early integration of verification in the design process, these are based on theories and mathematical notations allowing both to formally specify the program, to check and prove that its implantation's compliance with all the properties described in the specification. This is called proper implementation with respect to the specification and formally verified program. Formal methods are recognized by standard references, so that the seventh and final confidence level of the Common Criteria [1] is granted to applications built with them. The awareness of the importance of checking more carefully the programs and the maturity of tools dedicated to this task has generated a considerable growth of programs formal verification in the last decade. Offering a global vision and a high-level structure and organization of a system, the software architecture plays a key role as a pivot point between the requirements of a system and its implementation.

The diversity of design concerns in general and particularly in MAS, request support on formal techniques, which offer enough flexibility and expressiveness to rigorously specify MAS architecture at both the static and dynamic level.

In our previous work [2] we proposed a new approach for modeling and analyzing MAS architectures called BDI MAS architecture in which Bigraphical Reactive Systems (BRS) [3] are adopted as a semantic framework to formalize MAS architectures that are based on the Belief-Desire-Intention (BDI) agent model [4] which is the most commonly used approach for representing agent internal state and it also has been used to build a number of significant real-world applications (i.e. web applications,...). Therefore, Milner's BRS are very suitable to formalize MAS fundamental architectural aspects and their reconfiguration.

Thus, in this work we argue that in addition to their graphical aspect and rigorous basis, BRS are capable of representing both locality and connectivity constituting main concepts of MAS architecture then we propose our a bigraph-based model in order to reason about their properties for that we use the automatic proof tool BigMC to analyze the specifications and verify system properties during configuration. The rest of the paper is organized as follows. In section 2, we introduce Bigraphical Reaction Systems (BRS) and the automatic proof tool BigMc . Section 3 and 4 present the related works and then our bigraphical specification of BDI-MAS architecture. The given

formalization approach is verified and validated by the BigMC tool through examples in section IV. Finally, some concluding remarks and ongoing work finish the paper.

II. BIGRAPHS AND BIGRAPHICAL MODEL CHECKER

A. Bigraphs

Bigraphical Reactive Systems were initially introduced by Milner [3] to provide a completely graphical intuitive formal model capable of representing at the same time connectivity and locality of distributed entities which is very close to MAS concepts. The proposal of BRS provides a model for information systems with mobile placing and mobile linking, in which real-world pervasive and distributed systems can be described and analyzed. Further, it provides the unification of existing process calculi for concurrency and mobility (such as π-calculus, Petri nets, λ calculus, and so on) in a simpler way [5].

Structural Aspects: A bigraph is the combination of two independent structures place and link graphs. The place graph represents system entities geographical distribution. The link graph is a hypergraph representing interconnections between these entities. Within a BRS, system entities are represented by nodes and interactions between them are represented by edges (see Fig. 1.). A node can be dotted with ports representing connexion points to edges or inner/outer names.

Each node has a control, which is an identifier belonging to a set that is called a signature (usually denoted as S). Each control indicates how many ports the node has, whose controls are atomic (node empty), and which of the non-atomic controls are active (node permitting reaction inside) or passive. The inner names and outers names of a bigraph indicate connecters to which other bigraphs or roots (i.e. regions) can be connected. Such interconnection is possible only if the outer name of a bigraph or root is equal to the inner name of another bigraph. Sites represent holes into which a root or node can be nested. They are considered as an abstraction indicating the presence of other elements.

Definition [3]: a bigraph is formally defined by $G = (V, E, ctrl, G^P, G^L) : I \rightarrow J$, $I = \langle m, x \rangle$, $J = \langle n, y \rangle$, where:

- V and E represent finite sets of nodes and edges respectively.
- $ctrl : V \rightarrow K$ a control map that assigns a control to each node. The signature K is a set of controls.
- G^P and G^L are Place and Link graphs respectively.
- I and J represent inner and outer names (interfaces) respectively of the bigraph G.

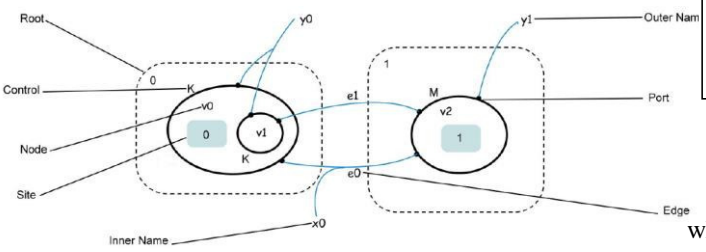

Fig. 1 The anatomy of bigraphs

Bigraph can also be expressed by term language. In [5] Milner axiomatises the structure of bigraphs, to prove that the theory is complete, the algebra of bigraphs structure is surprisingly simple, the primary operations and elements used in this paper are summarized in Table 1.

TABLE 1. TERMS LANGUAGE FOR BIGRAPHS.

Term	Signification
$U \mid\mid V$	Juxtaposition of roots
$U \mid V$	Juxtaposition of nodes
$U \circ V$	Composition
$U . V$	Nesting(U contains V)
$/x . U$	U with outer name x replaced by an edge
x/y	Connection inner names y to outer name x

Dynamical aspects: Bigraphs structural dynamics is expressed through a BRS (Bigraphical Reactive System) consisting of a category of bigraphs and a set of reaction rules; each one defines a redex bigraph to be transformed to a reactum bigraph. Formally, a reaction rule takes the form(R,R',n) where $R : m \rightarrow J$ is a redex, $R' : m' \rightarrow J$ is a reactum and $n : m' \rightarrow m$ is a map of ordinals [3]. The category of all bigraphs and their reaction rules constitute a BRS.

B. Bigraphical Model Checker (BigMC)

The use of formal methods allows rigorous verification of computer systems. There exist a number of formal verification techniques, model checking [6, 7] is one of many and BigMC (Bigraphical model checker) [8] is one of the few model checking tool devoted to the verification of models encodes as a Bigraphical Reactive System. BigMC ensures that the system's behavior meets the expected properties. This verification is fully automated and consists in exploring all the possible cases. The result of this analysis is to confirm that each property is verified, or not. In the latter case, and this is one of the main interests of this tool, the model checker returns a counter-example. Fig. 2 shows the full BigMC bigraph term:

$$
\begin{aligned}
&M ::= E; M \mid E \\
&E ::= \%passive\ k : arity \\
&E ::= \%active\ k : arity \\
&E ::= \%rule\ n\ T \rightarrow T \\
&E ::= \%property\ n\ P \\
&E ::= T \rightarrow T \mid T \\
&T ::= K.T \mid T \mid T \mid T \mid\mid T \mid \$n \mid K \mid nil \\
&K ::= k[names] \mid k \\
&names ::= n, names \mid n \\
&n ::= [a-zA-Z][a-zA-Z0-9]* \mid - \\
&P ::= matches(T) \mid terminal() \mid !P
\end{aligned}
$$

Fig. 2 BigMC terms language

Using the grammar in Fig. 2, we can specify a model M which may be a composed of another model or an expression E or both. In turn an expression E can be composed of nodes

(being active or passive and assigning an arity to each one of them), reaction rules (whose form is as follows T -> T) and properties denoted by P (which are expressed as a logical formula). In this work, we will use the BigMC grammar to specify our proposed BDI-MAS architecture model in order to check and validate some properties.

III. Multi Agent Systems Bigraph Based Specification

To better understand the multi agent system development, we have reviewed the literature related to Architecture description languages (ADLs) and those of MAS. Therefore, in our previous work [2] we have captured the fundamental concepts to better ensure the specification, evolution and the verification of MAS architecture. This modeling has studied both the structural and dynamic dimension of multi-agent systems architectures. These two dimensions (structural and dynamic) will be developed in this section to show how the proposed framework based on the BRS as a formal notation, express the multi-agent architectures.

At a high level of abstraction, multiagent system is considered as a set of computing entities (a set of agents) that are distributed across multiple sites, and are often referred to as nodes. In Table 2, we summarize fundamental elements intervening in a BDI-MAS architecture.

TABLE 2. CORRESPONDENCE BETWEEN MAS AND BRS CONCEPTS.

MAS architectural element	Bigraph element
Agents, Beliefs module, Desires module, Intention module, plans.	Node
Physical or logical location the agents	Root
Various type of links between the different elements	Edge/Hyper Edge
Abstract elements	Site

A. Structural description of the BDI-MAS model

Our BDI-MAS architecture model structure follows core principles, which we organize in two levels of abstraction: (i) internal (or agent) level; (ii) social (or MAS) level. The former describes the internal structure and state of the agent (i.e. the basic construct elements of the MAS) and the second describes the assembly and interaction among agents that compose the MAS architecture. A multi-agent system does not reduce to a centralized computer system; it consists of a set of interconnected agents. Where each agent can initiate communication, generate messages, and respond to other agent's messages, in order for agents participating in these interactions to achieve overall system goals [9].

a) Agent level:

The Fig. 3 shows our BDI agent and its internal structure. Each agent (denoted by AG) is composed of three principal nodes, which in turn contains other nodes that structure them. In what follows, we will take a closer look on the nodes that compose the agent AG1, for more details see [2].

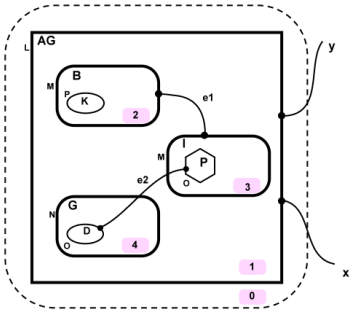

Fig. 3 Bigraphical model of BDI Agent.

The signature associated to a BDI-MAS bigraph is as follows:
K = { L: (2, active), M: (1, active), N :(0, active), O :(1, atomic), P :(0, atomic)}, L, M, N, O and P represent controls associated to different nodes. The different nodes types used in the model and their associated controls are summarized in Table 3.

TABLE 3. NODES TYPES OF BDI-MAS ARCHITECTURE.

Node	Control	Attribute	Arity	Meaning
AG	L	Active	2	Agent
B	M	Active	1	Beliefs Module
G	N	Active	0	Goal Module
I	M	Active	1	Intention Module
P	O	Atomic	1	Plan
D	O	Atomic	1	Desire
K	P	Atomic	0	Knowledge

b) Social level:

The model presented provides notations for describing the structure of MAS in terms of hierarchical configurations of interacting components. It provides an explicit and common basis for describing MAS architectural configurations (see Fig. 4).

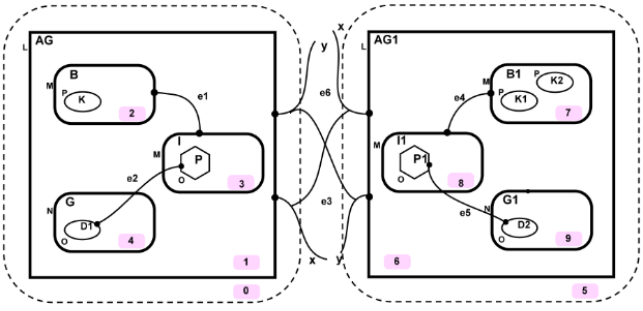

Fig. 4. Bigraphical model of BDI-MAS configuration.

B. Modeling BDI-MAS Architectural Reconfiguration

As defined in our previous work [2] the BDI-MAS architecture dynamics is formalized using reaction rules expressing changes of form in terms of shape shifting while preserving architectural constraints. In this subsection, we give some reaction rules samples defined to model BDI-MAS internal and external behavior and reconfiguration. Table 4 depicts how we defined the behavioral model, based on reaction rules.

TABLE 4. MODELLING MULTI AGENT SYSTEM DYNAMICS.

Multi Agent System	BRS
Configuration *MAS*.	Bigraph : $G_{MAS}= (V_{MAS}, E_{MAS}, ctrl_{MAS}, G_{MAS}\,p, G_{MAS}\,L)$
Reconfiguration from *MAS* to *MAS'*	Meta reaction rule: $RL = (MAS, MAS', m' \rightarrow m)$

Example RL1: Resolution of an internal goal reaction rule

$$AG_{x\,y}\ .(B_{e1}.(K\ |d2)\ |G.(D1\ |d4)\ |I_{e1}\ .(P\ |d3)\ |d1)\ |d0$$
$$\rightarrow$$
$$AG_{x\,y}\ .(B_{e1}.(K\ |K1\ |d2)\ |G.(D1_{e2}\ |d4)|\ I_{e1}\ .(P_{e2}\ |d3)\ |d1)\ |d0$$

Example RL2: Resolution of an external goal (collaboration) reaction rule

$$AG_{xy}.(B_{e1}.(K|K1|d2)|G.(D1|d4)|I.(d3)|d1)|AG1_{xy}.(B1_{e2}.(K2|d7)|G1.(D3|d9)|I1_{e2}.(d8)\ |d6)$$
$$\rightarrow$$
$$AG_{xy}\ .(B_{e1}.(K\ |K1\ |K3\ |d2)\ |G.(D1_{e6}\ |d4)\ |I.(P2_{e6}\ |d3)\ |d1)\ |$$
$$AG1_{xy}.(B1_{e2}.(K2\ |K3\ |d7)\ |G1.(D3\ |d9)|I1_{e2}.(d8)\ |d6)$$

IV. FORMAL ANALYSIS OF PROPERTIES

Software verification becomes essential to the development of computer systems. Indeed, the use of formal methods allows to prove that a system satisfies a given specification. In fact, these methods appear one of the main solutions for the development of high quality and safe systems at a reasonable costs and time span. Further, the use of these methods in the development process allows the verification and validation of the specification and facilitates the passage to the implementation. These techniques are accompanied by powerful tools that can be used to automate various stages of verification. The use of rather conventional design methods (composition, aggregation, etc.) in the development cycle has paved the way for the smooth introduction of techniques such as model checking.

In our case, we use BigMC a Bigraphical Model Checker to check properties such as deadlock and some violations that the model should not allow to happen during its execution. First, we specify the structural aspect (i.e., nodes and their signature and outer and inner interfaces …) then the dynamic aspect (i.e., reaction rules ex internal resolution of goal …) using the BigMC syntax term language. Then we formulate the properties that we would like to verify on each example. Finally, we will analyze and validate the resulted output given by the BigMC tool.

A. Reachability checking

In this section, we would like to verify the soundness of our model. For that purpose, we decide to start with the building blocks of our model's dynamic, which are no other than the reaction rules for the resolution of an internal, external goal resolution and the reconfiguration example of adding a new agent to the system specified in the section 4.

Fig. 5 describes, how our BDI agent is able to solve an internal goal, as it may be seen in the redex, the presence in the node G of a desire D1 to satisfy, one can also notice the presence into the node B of a knowledge node denoted by K, which is necessary for triggering the rule. In the reactum one can first see the appearance of a node P, which is the best plan among plans that can satisfy the desire D1 (choosing the best possible plan remains tied to heuristics that cannot appear in the architectural level). Secondly, the creation of a link e2 between the node D1 and P specify that the desire D1 could be satisfied by the execution of the plan P. Finally, the execution of P induces two possible cases either: (1) adding / removing knowledge at the node B (2) beliefs of the agent do not change [2].

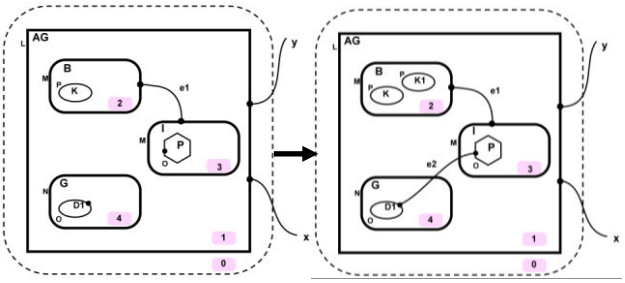

Fig. 5. Internal goal resolution reaction rule.

Fig. 6 below represents the structural bigraphical specification of the resolution of an internal goal in BigMC term language.

```
# BDI Agent Internal Nodes                    #Hyper-edges
                                              %name in;
%active Agent : 2;                            %name out;
%active Intensions : 1;                       %name e1;
%active Beliefs : 1;                          %name e2;
%active Goals : 0;                            # ------
%passive K : 0;
%passive K1 : 0;
%passive Plan1 : 1;
%passive Desire1 : 1;
%passive Plan2 : 1;
%passive Desire2 : 1;

# ---Initial configuration---

Agent[in,out].(Beliefs[e1].(K)|Goals.($0) |Intensions[e1]);
```

Fig. 6 Internal goal resolution in BigMC term language

Fig. 7 shows the dynamics of our example, through the presentation of a sequence of meta-reaction rules written in BigMC, we can also see that there are two types of reactions of rules linking and placing reaction rules. The former is responsible for creating or deleting links between nodes while the second type is responsible for creating, moving or deleting nodes in our BDI-MAS Agent.

```
# RL 1
Agent[in,out].(Beliefs[e1].($5)|Goals.($0)|Intensions[e1])->
Agent[in,out].(Beliefs[e1].($5)|Goals.(Desire1[-]) |Intensions[e1].($1));

# RL 2
Agent[in,out].(Beliefs[e1].($5)|Goals.(Desire1[-])|Intensions[e1].($1))->
Agent[in,out].(Beliefs[e1].($5)|Goals.(Desire1[-]) |Intensions[e1].(Plan1[-]));

# RL 3
Agent[in,out].(Beliefs[e1].($5)|Goals.(Desire1[-])|Intensions[e1].(Plan1[-]))->
Agent[in,out].(Beliefs[e1].($5)|Goals.(Desire1[e2]) |Intensions[e1].(Plan1[e2]));

# RL 4
Agent[in,out].(Beliefs[e1].($5)|Goals.(Desire1[e2])|Intensions[e1].(Plan1[e2]))->
Agent[in,out].(Beliefs[e1].(K |$5)|Goals.($0) |Intensions[e1]);
```

Fig. 7. Internal goal resolution dynamics

Now that we have specified our model structural and dynamical aspects in BigMC the penultimate stage before the checking is to specify or formulate the property to check on our example. For this purpose, BigMC provides a set of predefined predicates using the syntax showing in Fig. 2, in the following example, we will use two of them, the first property that we would like to verify named violation_free uses the predicate !match(T) which states that we must not find a match to the expression between brackets during our system execution. The second property is deadlock_free which uses the predicate !terminal() the predicate as transcribed here states that there will be a possible future state reachable by a step of reaction rule from the current one. For more elaborated properties the common boolean operator such as AND, OR and NOT are used.

%property secure !matches(Agent[in,out].(Beliefs[e1].(K | K1 | $2)|Goals.(Desire1[-] | $0) |Intensions[e1].(Plan1[-] | $1)));

%property deadlock_free !terminal();

The result of the model checking is shown in the figure after 20 steps the model checker reached successfully the intended state and does not report any property violation (due to the lake of space intermediate rewriting steps are omitted) .

```
> C:\Progra~1\BigMC/bin/bigmc -m 20 -r 50 -p
C:\DOCUME~1\dhte\LOCALS~1\Temp\bigmc_model3872130719824471653.bgm
1: Agent[in,out].(Beliefs[e1].K.nil | Intensions[e1].nil | Goals.$0)
2: Agent[in,out].(Beliefs[e1].K.nil | Goals.Desire1[-].nil | Inten-
sions[e1].nil)
3: Agent[in,out].(Beliefs[e1].K.nil | Goals.Desire1[-].nil | Inten-
sions[e1].Plan1[-].nil)
4: Agent[in,out].(Beliefs[e1].K.nil | Intensions[e1].Plan1[e2].nil |
Goals.Desire1[e2].nil)
.........
18: Agent[in,out].(Beliefs[e1].(K.nil | K.nil | K.nil | K.nil | K.nil)
| Goals.Desire1[-].nil | Intensions[e1].nil)
19: Agent[in,out].(Beliefs[e1].(K.nil | K.nil | K.nil | K.nil | K.nil)
| Intensions[e1].Plan1[-].nil | Goals.Desire1[-].nil)
20: Agent[in,out].(Beliefs[e1].(K.nil | K.nil | K.nil | K.nil | K.nil)
| Goals.Desire1[e2].nil | Intensions[e1].Plan1[e2].nil)
[mc::step] Interrupted!  Reached maximum steps: 20
 [mc::report] [q: 1 / g: 21] @ 20
```

Fig. 8. Internal goal resolution checking result.

Example 2 the external goal resolution implies at least two agents as shown in the Fig. 4 of the section 4. The Fig. 9 represents the example transcribed in BigMC term language.

```
# RL 1
Agent[-,-].(Beliefs[e1].(K)|Goals.(Desire1[-
])|Intensions[e1])||Agent1[-,-].(Beliefs[e2].(K1)|Goals
|Intensions[e2]) ->
Agent[in,-].(Beliefs[e1].(K)|Goals.(Desire1[-
])|Intensions[e1])||Agent1[in,-].(Beliefs[e2].(K1)|Goals.(Desire1[-])
|Intensions[e2].($0));
.........
# RL 10
Agent[-,-].(Beliefs[e1].(K | K2 |$2)|Goals.(Desire1[e4])
|Intensions[e1].(Plan2[e4])) ->
Agent[-,-].(Beliefs[e1].(K | K2 | K3 |$10)|Goals.(Desire1[e4])
|Intensions[e1].(Plan2[e4]));

# ---Initial configuration---
Agent[-,-].(Beliefs[e1].(K)|Goals.(Desire1[-])|Intensions[e1])|Agent1[-
,-].(Beliefs[e2].(K1)|Goals |Intensions[e2]);

%property secure !matches(Agent[in,out].(Beliefs[e1].(K | K2 | K3
|$6)|Goals.(Desire1[-]) |Intensions[e1].(Plan2[-])));

%property deadlock_free !terminal();

%check
```

Fig. 9 External goal resolution written in BigMC.

The model checker rewriting steps is limited to 50, the result is without call the model is free of violation. As a result, the BigMC tool does not give a counter example see Fig. 10.

```
> C:\Progra~1\BigMC/bin/bigmc -m 20 -r 50 -p
C:\DOCUME~1\dhte\LOCALS~1\Temp\bigmc_model5536451907465996104.bgm
1: (Agent[-,-].(Goals.Desire1[-].nil | Beliefs[e1].K.nil | Inten-
sions[e1].nil) | Agent1[-,-].(Beliefs[e2].K1.nil | Goals.nil | Inten-
sions[e2].nil))
2: (Agent[in,-].(Beliefs[e1].K.nil | Goals.Desire1[-].nil | Inten-
sions[e1].nil) | Agent1[in,-].(Beliefs[e2].K1.nil | Goals.Desire1[-
].nil | Intensions[e2].nil))
3: (Agent[in,-].(Beliefs[e1].K.nil | Goals.Desire1[-].nil | Inten-
sions[e1].nil) | Agent1[in,-].(Beliefs[e2].K1.nil | Goals.Desire1[-
].nil | Intensions[e2].Plan1[-].nil))
.........
9: (Agent1[-,-].(Goals.nil | Intensions[e2].nil | Beliefs[e2].nil) |
Agent[-,-].(Goals.Desire1[-].nil | Intensions[e1].Plan2[-].nil | Be-
liefs[e1].(K.nil | K2.nil)))
10: (Agent[-,-].(Intensions[e1].Plan2[e4].nil | Goals.Desire1[e4].nil |
Beliefs[e1].(K2.nil | K.nil)) | Agent1[-,-].(Beliefs[e2].nil | Inten-
sions[e2].nil | Goals.nil))
11: (Agent1[-,-].(Beliefs[e2].nil | Goals.nil | Intensions[e2].nil) |
Agent[-,-].(Beliefs[e1].(K3.nil | K.nil | K2.nil) |
Goals.Desire1[e4].nil | Intensions[e1].Plan2[e4].nil))
[mc::step] Complete!
 [mc::report] [q: 0 / g: 11] @ 12
```

Fig. 10. External goal resolution checking result.

V. Related work

There is an important core of work regarding to the design and development at the architectural level as mentioned in [9] and [10], several works propose different languages, formal and semi-formal, Architecture Description Language (ADL). Such as Darwin, Rapide, Dynamic-Wright [11] and π-ADL[12] for representing and analyzing software architectures in order to predict architectural qualities before the implementation, and guiding the design and coding process. Nonetheless, these works are labeled by a lack of coverage of concepts related to the definition of a multi-agent system, for example, the representation of the agent is

generally limited by a single object devoid of the necessary concepts to express its autonomy and cognitive aspects (such as beliefs, knowledge and competences). As cited in [13] there exist various analysis techniques among the existing ADLs for testing, model checking, and evaluating performance based on architectural models. Bordini in [15, 16] has presented an approach for verifying multi-agent programs. In this approach, the system is written with the logic-based agent-oriented programming language AgentSpeak and automatically translated into either Promela or Java. This is an important work; however, the verification of MAS focuses on the program rather on the architecture. In [17] Walton address the verification of communication between agents participating in multi-agent web service systems, the approach is based on the application of model checking techniques. This approach is too specific, it is used to verify lightweight protocol language and it cannot be applied to a wide range of multi agent systems and neither at the architectural level. In [18] and [19] are abstract formal models for developing formal specifications of multi-agent systems these approachs uses the Z notation as formal foundation. However, the Z language cannot model in an effective way the interaction, distribution and the concurrence in a MAS. Fisher in [20] describes the first steps towards a formal specification and verification of multi-agent systems using Concurrent METATEM and the temporal belief logics. This approach suffers from a low level of abstraction and does not take into account the reconfiguration of the system at the architectural level.

VI. Conclusion

In this paper, we have described our proposed formal modeling approach of the BDI-MAS architecture. The system has been specified at both individual (agent) and social (MAS) levels. The BDI-MAS bigraph simplifies considerably the MAS architectures readability. A MAS architecture is seen as a hierarchical configuration of interacting nodes. The model emphasizes on both locality and connectivity that can be used to represent the location and interconnection of MAS architectures. On the other hand, reaction rules allow developers to correctly analyze the BDI-MAS architecture features, including modeling the behavior of the BDI agents and describing reconfigurations that could be added to the architecture. Further, the use of bigraphs as formal basis in the development process allows the verification and validation of the specification. Using the BigMC tool we have shown that our BDI-MAS architecture model through its Meta reaction rules are free of violations and deadlock. Our aim is to have a graphical intuitive solid formal foundation for modeling MAS architecture in order to handle the complexity of the systems in general, adopting a high level of abstraction that removes unnecessary details regarding all the expected properties and facilitates the passage to the implementation.

In the perspectives of this work, we plan to:

- Formally analyze and verify some non-functional properties such as security of the BDI-MAS architectures model.
- Provide a tool that generates executable implementation from our BDI-MAS architecture model,
- Develop a methodology around the model in order to guide the development of MAS.

References

[1] Commoncriteriaportal.org, 'Common Criteria : New CC Portal', 2015. [Online]. Available: http://www.commoncriteriaportal.org/. [Accessed: 16- Feb- 2015].

[2] A. Dib and Z. Sahnoun, 'Formal Specification of Multi-Agent System Architecture', in International Conference on Advanced Aspects of Software Engineering, Constantine, 2014, pp. 65-72.

[3] R. Milner, 'Bigraphs and Their Algebra', Electronic Notes in Theoretical Computer Science, vol. 209, pp. 5-19, 2008.

[4] A. Rao and M. Georgeff, 'Modeling rational agents within a BDI-architecture', Australian Artificial Intelligence Institute, Victoria, Australia, 1991.

[5] R. MILNER, 'Axioms for bigraphical structure', Math. Struct. in Comp. Science, vol. 15, no. 06, p. 1005, 2005.

[6] S. Merz, 'Model Checking: A Tutorial Overview', in Modeling and Verification of Parallel Processes, 2001, pp. 3-38.

[7] S. Merz, 'Model Checking Techniqes for the Analysis of Reactive Systems', Synthese, vol. 133, no. 12, pp. 173-201, 2002

[8] G. Perrone, S. Debois and T. Hildebrandt, 'A model checker for Bigraphs', Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12, 2012.

[9] R. Allen, 'A Formal Approach to Software Architecture', Phd, Carnegie Mellon University, 1997.

[10] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione and A. Tang, 'What Industry Needs from Architectural Languages: A Survey', IIEEE Trans. Software Eng., vol. 39, no. 6, pp. 869-891, 2013.

[11] N. Medvidovic and R. Taylor, 'A classification and comparison framework for software architecture description languages', IIEEE Trans. Software Eng., vol. 26, no. 1, pp. 70-93, 2000.

[12] R. Allen, R. Douence and D. Garlan, 'Specifying and analyzing dynamic software architectures', Fundamental Approaches to Software Engineering, pp. 21-37, 1998.

[13] F. Oquendo, pi-ADL', SIGSOFT Softw. Eng. Notes, vol. 29, no. 3, p. 1, 2004.

[14] P. Zhang, H. Muccini and B. Li, 'A classification and comparison of model checking software architecture techniques', Journal of Systems and Software, vol. 83, no. 5, pp. 723-744, 2010.

[15] R. Bordini, M. Fisher, W. Visser and M. Wooldridge, 'Verifying Multi-agent Programs by Model Checking', Autonomous Agents and Multi-Agent Systems, vol. 12, no. 2, pp. 239-256, 2006.

[16] R. Bordini, M. Fisher, C. Pardavila, W. Visser and M. Wooldridge, 'Model Checking Multi-Agent Programs with CASP', Computer Aided Verification, pp. 110-113, 2003.

[17] W. Wan, J. Bentahar and A. Ben Hamza, 'Modeling and Verifying Agent-Based Communities of Web Services', Trends in Applied Intelligent Systems, pp. 418-427, 2010.

[18] D'Inverno, M., Luck, M., Georgeff, M., Kinny, D. and Wooldridge, M. (2004). The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System. Autonomous Agents and Multi-Agent Systems, 9(1/2), pp.5-53.

[19] Luck, M. and d'Inverno, M. (2006). Formal Methods and Agent-Based Systems. NASA Monographs in Systems and Software Engineering, pp.65-96.

[20] M. Fisher and M. Wooldridge, 'On the Formal Specification and Verification of Multi-Agent Systems', International Journal of Cooperative Information Systems, vol. 06, no. 01, pp. 37-65, 1997.