International Journal of u- and e- Service, Science and Technology
Vol.8, No. 11 (2015), pp.113-124
http://dx.doi.org/10.14257/ijunesst.2015.8.11.12

A Formal Verification Method of Obligation Policy in Multi-agent System

Zhang Tao[1], Xie Hong and Huang Shao-bin[3]

[1,3]*College of Computer Science and Technology, Harbin Engineering University, Harbin, China, 150001*
[2]*College of Information and Communication Engineering, Harbin Engineering University, Harbin, China, 150001*
zhangtaohrbeu@163.com

Abstract

In Multi-Agent System, obligations are actions that agents are required to take or some states of affairs which should be maintained, formal modeling and verifying obligation policy which is high-level requirements specifications or communication protocol for constraining agent interaction can enhance the correctness of the system design. In this paper, we present a formal framework for modeling obligation policy, the framework is formalized using concepts of social commitments, it allows reasoning about change in the states of obligations and provides Kripke operational semantics which can help to convert framework of obligation policy to the input model of model checker MCMAS, at the same time ,the properties of framework depending on policy conflicts are represented with CTL, and the violations of properties is detected by using MCMAS which can provide the counterexample and trace it back to the errors in interaction policy. Finally, we present an implementation and report on experimental results of verifying obligation policy using MCMAS model checker.

Keywords*: Multi-Agent System; Model Checking; obligation policy; formal method; MCMAS*

1. Introduction

In the Multi-Agent System (MAS), obligations are generally actions that agents are required to take and are essential for the expression of a large number of requirements [1]. For instance, the social insurance administration regulations specified that enterprises need to go to the local social insurance agencies for social insurance registration in 20 working days from the date of the establishment of the enterprise. Obligations may also define states of affairs which should be maintained [2]. For example, people who pay the endowment insurance must be in the insured state. Since obligation policy constraints the interaction between agents within its domain, it can be regarded as high-level requirements specification [3] or communication protocol [4] in the system design process. Formal obligation policy can help to clarify the description and interpretation of policies, and to correctly design interactions between agents. Eventually, formal verification of obligation policy can also help to ensure the correctness and reliability of the system. Minsky [5] first makes a point of law policy research in the field of computer science, and paper [6] proposed "Law-Governed Interaction" on policy research in distributed systems which inspired a lot of practice in the field of work. The LGI model uses low-level abstract primitives modeling the system so that it cannot meet the increasingly complexity of system design, this abstraction far away from specific fields is a problem common to all such languages, therefore variety of formal languages of obligation policies have been proposed, they

ISSN: 2005-4246 IJUNESST

can be generally classified into two categories. In one hand, there are policy enforcement languages, such as [7], these languages can simply describe and explain policies, however, they lack the formal semantics needed to allow formal policy analysis and the verification of policy properties. On the other hand, there are policy analysis languages such as [8] and [9], these languages allow formal policy analysis and the expression of a large variety of obligations. However, these languages do not provide the operational semantics needed to dynamically enforce and manage obligations in the Multi-Agent System.

View of the above problems, in this paper we firstly proposes a policy framework language which aims at combining the advantages of both policy enforcement and analysis languages, it can enable the specification of large number of practical real life requirements and simple to be converted to the input model of model checker MCMAS based on its operational semantics. Finally, the properties of framework depending on policy conflicts are represented with CTL, and the violations of properties is detected by using MCMAS which can provide the counterexample and trace it back to the errors in interaction policy.

2. Social Commitment

In this paper, we use the concept of social commitment to define the formal semantics of obligation in the interaction policy of MAS. A social commitment SC is a public commitment made by an agent (called the debtor), and directed towards a set of agents (called creditor), indicating that some fact is true or some action will be performed [10]. Nowadays, social commitments have been extensively and effectively used in a variety of areas ranging from developing artificial institutions [11], verifying process communication [12], and modeling multi-agent interaction [13] and so on. The definition of social commitment as follows:

Definition 1 Social Commitment. Social Commitment is a four-tuple $SC = (id, Ag1, Ag2, \varphi)$, where id is unique identifier, $Ag1$ is the debtor, $Ag2$ is the creditor and φ is a well-formed formula (expressed in some logics) representing the commitment content.

3. Obligation Policy Language Model

An obligation in the policy is a special social commitment in the sense that the debtor must respect and behave in accordance with this commitment, however, the concept of social commitment cannot be a complete representation of the semantic of obligation policy, such as policy contexts, obligation types, obligation state model, etc. Next we define the obligation and obligation policy language by extending the concept of social commitment.

3.1. Obligation Policy Language

Obligation policies are often associated with some condition collection existing in the form of contexts which represent the system states and conditions. Therefore, before the defining the obligation policy language, we first define the context of policy. There are two possible types of contexts, namely state-based contexts and event-based contexts. State-based contexts enable the definition of state condition during which a policy rule applies. On the other hand, event-based contexts specify time moment at which an obligation is activated, violated or fulfilled. State-based contexts is represented as $Cs(Ag1, Ag2, \varphi) \leftarrow p_1, \ldots, p_n$, it states that the context Cs holds for $Ag1$, $Ag2$ and φ while the proposition formula are true. Event-based contexts denote the moments at which state-based contexts begin and cease to hold, and every stated-based context can be

associated with two event-based contexts: an event-based context $start(Cs)$ to denote the moment at which Cs begins to hold, and another event-based context $end(Cs)$ which represents the moment at which the Cs ceases to hold, We used Backus-Naur Form (BNF) to define the context expression of obligation policy as follows:

$$Cs ::= \text{true} \mid \text{false} \mid c \mid Cs \wedge Cs \mid Cs \vee Cs \mid \neg Cs \mid [Cs_1, Cs_2]$$
$$Ce ::= start(Cs) \mid end(Cs) \mid Cs \wedge Ce \mid Ce \wedge Cs$$
$$C ::= Cs \mid Ce$$

Where, C is the context expression of obligation policy, Cs is stated-based context expression, Ce is event-based context expression, C is a user-defined state context identifier. $[Cs_1, Cs_2]$ defines the interval which Cs_1 starts from the moment starts to hold until the moment Cs_2 starts to hold.

Definition 2 obligation. An obligation in the policy is a special social commitment $Obligation=(N, debtor, creditor, \varphi, Ca, Cv)$, where N is unique identifier, $debtor$ is the initiator of obligation, $creditor$ is the recipient of the obligation, φ is a well-formed formula (expressed in some logics) representing the obligation content, Ca and Cv are obligation context expressions, Ca is called the obligation's activation context, Cv is called the obligation's violation context.

Definition 3 Obligation Policy Language (*OPL*). Obligation Policy Language is a six-tuple $OPL = (A, O, C, Action, R, sta)$, where A is the set of agent in the policy, O is the set of obligations, C is the set of policy context, $Action = Act_o \cup Act_c$ is the action set, $Act_o = \{create, active, fulfil, violate, deactivate\}$ is set of actions which operation the obligations, Act_c is the set of action which happened in the obligation content, R is the set of properties of the policy, $sta: O \rightarrow \{\bullet, inactive, active, fulfilled, violated, violated / fulfilled, violated / inactive\}$ is a state distribution function which gives the current state of each obligation.

3.2. Obligation Types

The obligation types which can be expressed using OPL include: continuous obligation and discontinuous obligation.

Continuous obligations specify some state of affairs which should be maintained, they cannot be fulfilled and remain active in the system from the moment their context is started until their context is end. Consider the obligation: $Obligation=(n5, Ag1, Ag2, \varphi_5, insuranced, below\_limit\_payment)$, it specifies that the amount of insurance money which $Ag1$ pays to $Ag2$ cannot be less than the lower limit, where $Ag1$ is insurance participant, $Ag2$ is insurance management agency, and activation context is defined as $insuranced(Ag1, \_, \_) \leftarrow assigned(Ag2, Ag1)$ which specifies context $insuranced$ hold for $Ag1$, any object and any action when $Ag1$ is assigned to $Ag2$, the violation context $below\_limit\_fees$ is defined as $below\_limit\_payment(Ag1, \_, \_) \leftarrow limit\_payment(Ag1, x_1), current\_payment(Ag1, x_2), x_2 \prec x_1$, it specifies the situation that the mount of insurance money which insurance participant pays to insurance management agency is less than the lower limit.

Contrast with Continuous obligations, the discontinuous obligations can be divided into persistent obligations and non-persistent obligations. After being activated persistent obligations are obligations cannot be violated until they are fulfilled by debtor. The activation context of persistent obligations is either an event-based context expression $start(Cs)$ or a state-based context expression of the form $[Cs, false]$.

Non-persistent obligations may be deactivated after they are imposed. The condition defining non-persistent obligations is as follows: $end(Ca) \neq false$.

3.2. State Model of Obligation

We define obligation state model to specify the dynamic evolution of obligation policy, the model include some states and operations, and they are depend on obligation type described in the preceding. Obligations $Obligation=(N, debtor, creditor, \varphi, Ca, Cv)$ may have different states as shown in figure 1, and each of these states is formal specified as follows:

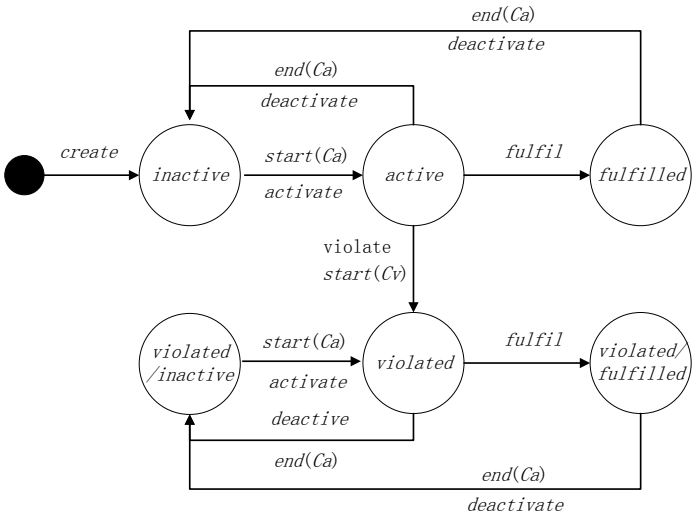

Figure 1. The State Model of Obligation

The initial state • which means the obligation is not created.

The *inactivate* state which can be specified by $inactive(obligation) \leftarrow (create(debtor, obligation) \wedge (\neg start(Ca)) \vee end(Ca)))$. There are two possible situations when obligation in an inactive state, one is obligation is not required to fulfill after being created, that is $create(debtor, obligation) \wedge \neg start(Ca)$, the other is obligation has been activated, but now it is not required to fulfill, in this case, the context $end(Ca)$ has become true.

The *active* state which can be denoted by $active(obligation) \leftarrow active(debtor, obligation) \wedge start(Ca)$. An obligation in the active state when context $start(Ca)$ is hold, that is, the obligation is required to fulfill, and $debtor$ activates the obligation.

The *fulfilled* state which is denoted by $fulfilled(obligation) \leftarrow fulfil(debtor, obligation) \wedge start(Ca)$。An obligation in fulfilled state when the activation context is hold and debtor have fulfilled the obligation.

The *violated* state which can be denoted by $violated(obligation) \leftarrow violate(debtor, obligation) \wedge start(Ca) \wedge start(Cv)$. An obligation in the violated state when activation context and violation context are hold simultaneously, and the debtor has violated the obligation.

The $violated / inactive$ state which can be specified a $violated / inactive(obligation) \leftarrow (violated(obligation) \wedge end(Ca) \wedge deactivate(debtor, obligation)$. The obligation in this state when debtor performs deactivate action in activate context after the obligation is violated.

The $violated / fulfilled$ state which can be denotes by $violated / fulfilled (obligation) \leftarrow (violated(obligation) \wedge start(Ca) \wedge fulfil(debtor, obligation)$. Obligation in this state when debtor fulfilled the obligation in the activated context after the obligation is violated.

3.3. The Formal Semantics of OPL

In order to verify the OPL model used by model checking technique, we firstly define the operational semantics of OLP as a state transition system which is often used in computer science as models to describe the behavior of systems and defined as follows[14]:

Definition 4. Transition System (*TS*). A transition system *TS* is a tuple: $TS = (S, Act, \rightarrow, I, AP, L)$ where:

- S is a set of states and Act is a set of actions;
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation;
- $I \subseteq S$ is a set of initial states;
- AP is a set of atomic propositions;
- $L : S \rightarrow 2^{AP}$ is a labeling function.

The labeling function L relates a set $L : S \rightarrow 2^{AP}$ of atomic propositions to any state s. $L(s)$ intuitively stands for exactly those atomic propositions $a \in AP$ which are satisfied by state s.

Definition 5. The operational semantics of OPL. The operational semantics of OPL is a transition system $TS_{OPL} = (S, Act, \rightarrow, I, AP, L)$, where:

- $S \subseteq (sta(o_1) \times ... \times sta(o_n)) \times C$ is a set of states;
- Act=$Action$ is a set of actions;
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation;
- $I \subseteq (\bullet_1 \times ... \times \bullet_n) \times C$ is a set of initial states;
- AP is a set of atomic propositions;
- $L : S \rightarrow 2^{AP}$ is a labeling function.

In the TS_{OPL}, the state s is composed of the current state and context of every obligations in the system, and the initial state is composed of the initial state and context of each obligation. The action set *Act* is defined the same as Action which defined in OPL, in other words, $Act = Act_o \cup Act_c$, where Act_o is action which operate the obligations, and Act_c is action which is included in the content of obligations. The action in the Act_o change the state of obligations, and the action in the Act_c change the context of obligations, so in the OPL model an obligation can evolve and be transformed as a result of the actions that the debtor performs on it, and its content may also be transformed as a consequence of the actions that debtor or creditors apply to it.

4. Model Checking Interaction Policy

Model Checking [15] is an automatic analysis and verification technology towards finite state concurrent system in the formal verification process, which uses state transition system (*TS*) model System behavior and uses temporal logic formula φ represent modal/temporal property, and verify whether the system meets the property by calculate $TS \models \varphi$?. For verifing the interaction policy of MAS, we first convert the OPL model of policy into the ISPL model which is the input model of model checker MCMAS.

4.1. OPL Model to ISPL Model

MCMAS [16] is a model checker for multi-agent systems with ISPL (Interpreted Systems Programming Language), which allows us to model MAS. In the ISPL model, the MAS is distinguished into environment agent and standard agents. Environment agent is used to describe boundary conditions, infrastructures and observation variables shared by standard agents. The agents are modeled as non-deterministic automaton in the form of a set of local states, a set of actions, protocol functions and evolution functions. The main step in our verification workflow is to translate OPL model into ISPL program. We start by translating the set of interacting agents directly into standard agents in Agent section and the obligations into local variables in Vars section. Such variables are enumeration type including all possible obligation states. Actions of agents are directly expressed in Actions section in which these actions work as constraints to trigger or stop transitions between obligation states.

4.2. The Properties of OPL Model

In our approach, we use computational tree logic (CTL) [15] to represent the properties of OPL model which need to be verified by MCMAS. CTL is built up from a finite set of propositional variables AP, the logical operators $\neg$ and $\vee$, and the temporal modal operators X (next) ,U(until), F(eventually), and G(always). Formally, the set of CTL formulas over AP is inductively defined as follows: $\phi ::= true \mid fasle \mid (\neg \phi) \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid \mathrm{A}\,\mathrm{X}\,\phi \mid \mathrm{E}\,\mathrm{X}\,\phi \mid \mathrm{A}\,\mathrm{F}\,\phi \mid \mathrm{E}\,\mathrm{F}\,\phi \mid \mathrm{A}\,\mathrm{G}\,\phi \mid \mathrm{E}\,\mathrm{G}\,\phi \mid \mathrm{A}[\phi\,\mathrm{U}\,\phi] \mid \mathrm{E}[\phi\,\mathrm{U}\,\phi]$. The properties of OPL model which need to be verified can be classified into: Safety and Liveness. The property Safety means that “some thing bad never happens” and it can be represented as $\mathrm{A}\,\mathrm{G}\neg\phi$. The property Liveness means that“something good will eventually happen” it can be represented as $\mathrm{A}\,\mathrm{G}(\phi \rightarrow \mathrm{A}\,\mathrm{F}\,\psi)$.

5. A Case Study

5.1. Model Obligation Policy

Taking the newly enacted law regarding to the transfer continuity of basic endowment insurance (TCoBEI) [17] as an example, the paper analyses the background and the key-points of the new policy and points out the interest game relation behind the policy based on OPL and model checking.

TCoBEI specifies the scopes of its application and the perform approach of basic endowment insurance. Limited space, we describe only the main content of TCoBEI as follows: The second term specifies the work type of insured person; the third term specifies the perform approach of basic endowment insurance and individual account in the migration process of insured person, and stress that the surrender procedure cannot be handled; the IV term specifies the calculation of transfer of funds, the assistance funds must be transferred in accordance with 12% of wage payment; the VI term specifies the insured person whose insurance relationship leaves census register seat can enjoy the benefits of basic endowment insurance unless the accumulative number of years of endowment insurance payment must be at least 10 years.

We formalize the policy TCoBEI according to the OPL model, there are four agents in the OPL model of TCoBEI: the agent GR describes the insured individual in the TCoBEI, the Agent PB represents social insurance management agency, the agent out_place indicates the location of insurance relationship of GR before GR changed the insured location, and the agent in_place model the location of insurance relationship of GR after GR changed the insured location. Due to space limitations, we only introduce the obligations of agent GR and PB which related to the properties to be verified, these obligations are shown as follows:

$Obligation = (n1, GR, PB, \varphi_1, want\_insured, over\_age\_limit)$, obligation $n1$ represent that people younger than 50 years old can participate in the insurance if he wants to join the insurance. $\varphi_1 = participating(GR, endowment\_insurance)$ represents the insurance activities of Agent GR, and the Boolean type context $want\_insured$ means whether GR wants to join the insurance, the context $over\_age\_limit$ indicates whether GR 's older than 50 years old.

$Obligation = (n2, GR, PB, \varphi_2, insured, not\_alive)$, obligation $n2$ indicates that GR should be registered in the social insurance management agency if he has participated in insurance. Where, $\varphi_2 = register(PB, GR)$ represents GR register in the social insurance management agency, the context $insured(GR, \_, \_) \leftarrow participating(GR, endowment\_insurance)$ represents GR in insured state and it is abbreviated as $insured$, the context not_alive means whether GR is alive.

$Obligation = (n3, GR, PB, \varphi_3, insuranced, retired)$, obligation $n3$ indicates that GR should pay pensions to PB before retirement, where $\varphi_3 = pay\_fees(GR, PB)$ represents GR pays pensions to PB and the context $retired$ indicates whether GR has retired.

$Obligation = (n4, GR, PB, \varphi_4, insuranced \wedge want\_transfer, retired)$, obligation $n4$ indicates that GR can apply to PB to migrate the insurance relationship before retirement. Where $\varphi_4 = apply\_transfer(GR, PB)$ represents GR applies to PB to migrate the insurance relationship and the context $want\_transfer$ indicates whether GR wants to migrate the insurance relationship.

$Obligation = (n5, GR, PB, \varphi_5, insuranced \wedge retired \wedge acc\_payment\_year, not\_alive)$, obligation $n5$ indicates that retired GR can get money from PB if GR is still alive and the cumulative contribution year of GR has reached standard requirements. Where $\varphi_5 = get\_money(GR, PB)$ represents GR gets money from PB and the context $acc\_payment\_year$ indicates whether the cumulative contribution year of GR has reached standard requirements

$Obligation = (n6, GR, PB, \varphi_6, insuranced \wedge \neg retired, not\_alive)$, obligation $n6$ means that GR who has not retired wants to quit the insurance. Where $\varphi_6 = apply\_quit(GR, endowment\_insurance)$ represents GR applies to PB to quit the insurance.

$Obligation = (n7, PB, GR, \varphi_7, insuranced \wedge want\_transfer \wedge \neg retired, not\_alive)$, obligation $n7$ represents PB agree to GR who has not retired to migrate the insurance relationship and $\varphi 7 = agree\_transfer(PB, GR)$.

$Obligation = (n8, PB, GR, \varphi_8, insuranced \wedge retired \wedge achieve\_payment\_year, not\_alive)$, obligation $n8$ represents PB pay the benefits of insurance to GR who has retired and is still alive. Where $\varphi_8 = pay(PB, GR)$ and the context $achieve\_payment\_year$ indicates whether GR can get the benefits of insurance.

$Obligation = (n9, PB, GR, \varphi_9, insuranced \wedge achieve\_payment\_year, retired)$, obligation $n9$ represents PB agree to GR who has not retired to quit the insurance.

5.2. Implementation of the Case Study

To show the feasibility of our approach, we implemented this case study by model checker MCMAS. Firstly the OPL model is converted into input model of MCMAS by the method described in Section 4.1, and the part of code of ISPL corresponding with OPL model of TCoBEI as shown in the figure 2 and figure 3. In particular, the features of Agent

are represented by context, such as the age of GR, the survival status of GR, and soon on. These contexts are converted into variables in the ISPL model.

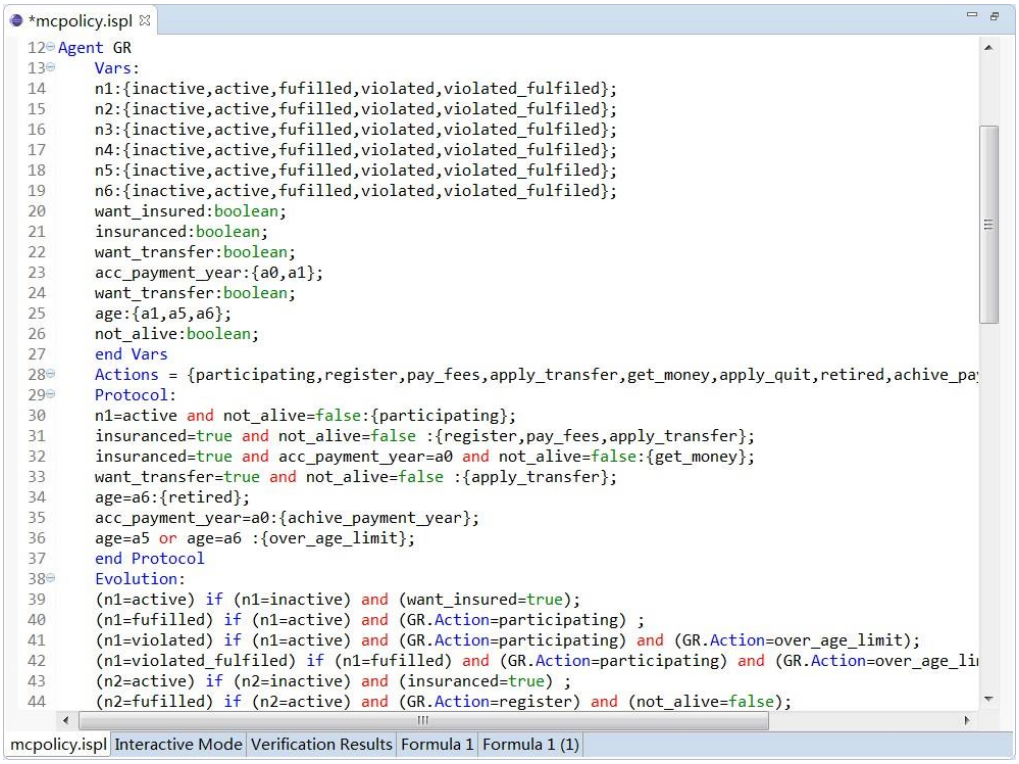

Figure 2. The ISPL Model of Agent GR

```
Agent PB
    Vars:
    n7:{inactive,active,fufilled,violated,violated_fulfiled};
    n8:{inactive,active,fufilled,violated,violated_fulfiled};
    n9:{inactive,active,fufilled,violated,violated_fulfiled};
    end Vars
    Actions = {agree_transfer,agree_quit,pay};
    Protocol:
    n7=active:{agree_transfer};
    n8=active :{pay};
    n9=active :{agree_quit};
    end Protocol
    Evolution:
    (n7=active) if (n7=inactive) and (GR.Action=apply_transfer);
    (n7=fufilled) if (n7=active) and (PB.Action=agree_transfer);
    (n7=violated) if (n7=active) and (PB.Action=agree_transfer) and (GR.Action=retired);
    (n7=violated_fulfiled) if (n7=fufilled) and (PB.Action=agree_transfer) and (GR.Action=retired) ;
    (n8=active) if (n8=inactive) and (GR.Action=achive_payment_year);
    (n8=fufilled) if (n8=active) and (PB.Action=pay) and  (GR.Action=retired) ;
    (n8=violated) if (n8=active) and (PB.Action=pay) ;
    (n8=violated_fulfiled) if (n8=fufilled) and (PB.Action=pay) ;
    (n9=active) if (n9=inactive) and (GR.Action=apply_quit);
    (n9=fufilled) if (n9=active) and (PB.Action=agree_quit) ;
    (n9=violated) if (n9=active) and (PB.Action=agree_quit) and (GR.Action=retired);
    (n9=violated_fulfiled) if (n9=fufilled) and (PB.Action=agree_quit) and (GR.Action=retired);
    end Evolution
end Agent
Evaluation
    GRinsured if GR.n1=fufilled;
    GRgetM if GR.n5=fufilled;
end Evaluation
InitStates
    GR.not_alive=true and GR.want_insured=true and  (GR.age=a5) or (GR.age=a1) or (GR.age=a6) and (GR.acc_payment_year=a1);
end InitStates
Formulae
    AG (GRinsured ->GRgetM) ;
end Formulae
```

Figure 3. The ISPL Model of Agent PB

In this case study, the property assertion is a CTL formula AG ((g.n1=fufilled)->(g.n5=fufilled)) which assert that the GR can enjoy the insurance benefits after retirement if GR participated in insurance. We use MCMAS to verify whether OPL model of TCoBEI meets the property and the verification results as shown in the Figure 4.

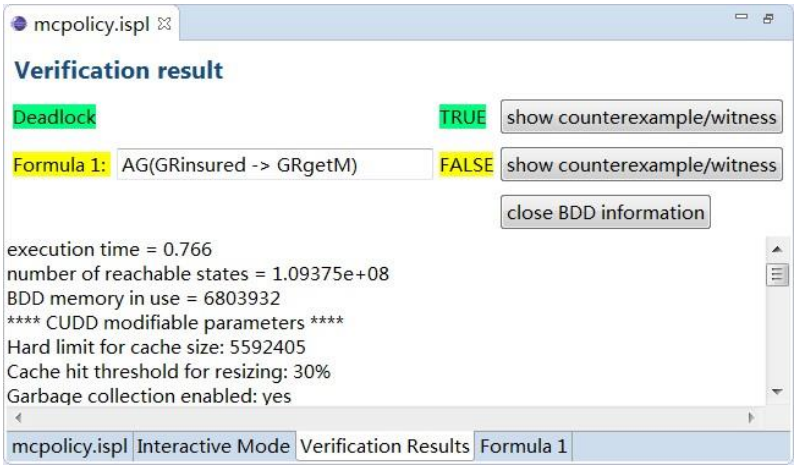

Figure 4. The Verification Results of TCoBEI

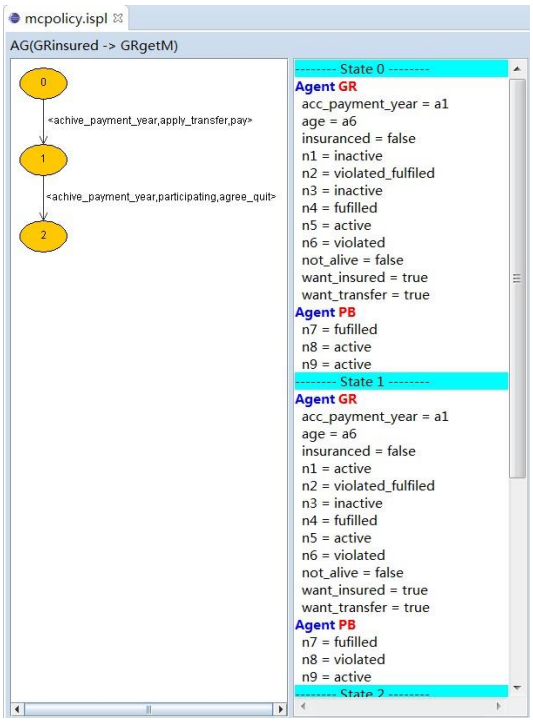

Figure 5. The Counter Example of Verification

In Figure 4, the MCMAS show that the policy model violates the constrains of property which specified by CTL formula and give a counterexample as shown in the Figure 5. Analyzing running track provided by counterexample, we found a flaw in the policy： According to the term VI of TCoBEI, insured person can enjoy the benefits unless they have to pay endowment insurance for ten years, this means that the insured person whose age over fifty and the years they pay for endowment insurance less than one for various reasons finally cannot enjoy the benefits of endowment insurance, they also cannot quit the endowment insurance according to

the third term of TCoBEI, this embarrassing situation will bring insured person huge economic losses.

6. Conclusions

This paper propose a formal method based social commitment to model and verify agent interactions specified by obligation policy in Multi-Agent System. The obligation policy language can specify large number of practical real life requirements and simple to allow non-specialists to understand it and use it, this is because the OPL model is associated with an obligation state-based model which clarifies the semantics of obligation by identifying the different obligation states and state transitions, and the OPL model use the context concept to represent the conditions of agent interaction. Furthermore, we defines the operational semantis of OPL model which can help to convert framework of obligation policy into the input model of model checker MCMAS. Finally, we use MCMAS to verify whether the OPL model meets the property represented by CTL formula, and we give a real case study to demonstrate the effectiveness and applicability of our approach. In the future work, we plan to extend the logic and its model checking to consider other obligation actions such as assign and delegate.

Acknowledgments

This work is sponsored by the National Science & Technology Pillar Program under grant number 2012BAH08B02, the Fundamental Research Funds for the Central Universities of China under grant number HEUCF100603 and HEUCF041204, and Postdoctoral Assistance Fund of HeiLongJiang under grant number 3236310148.

References

[1] M. Loizos, P. David and C. P. Avi, "Specifying and monitoring economic environments using rights and obligations", Autonomous Agents and Multi- Agent Systems, vol. 20, no. 2, **(2010)**, pp. 158-197.
[2] D. X. Xu, M. Sanford and Z. L. Liu, "Testing Access Control and Obligation Policies", International Conference on Computing, Networking and Communications, **(2013)**; San Diego, CA.
[3] J. Young, D. Antón and I. Annie, "A method for identifying software requirements based on policy commitments", Proceedings of the 18th IEEE International Requirements Engineering Conference, **(2010)**; Sydney, NSW, Australia.
[4] M. Baldoni and C. Baroglio, "Constitutive and Regulative Specifications of Commitment Protocols: A Decoupled Approach", ACM Transactions on Intelligent Systems and Technology, vol. 4, no. 2, **(2013)**, pp. 1-25.
[5] N. H. Minsky and D. Rozenshtein, "A law-based approach to object-oriented programming", Proceedings on Object-Oriented Programming Systems, Languages and Applications, **(1987)**; New York, NY, USA.
[6] N. H. Minsky and V. Ungureanu, "Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems", ACM Transactions on Software Engineering Methodology, vol. 9, no. 3, **(2000)**, pp. 273–305.
[7] W. L. Chi and G. H. Hwang, "A Framework and Language Support for Dynamic Security Policy in Service-Oriented Architecture", Journal of Information Science and Engineering, vol. 30, no. 6, **(2014)**, pp. 1887-1903.
[8] R. Craven, J. Lobo, J. Ma and A. Russo, "Expressive policy analysis with enhanced system dynamicity", Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, **(2009)**; New York, USA.
[9] Y. Elrakaiby, F. Cuppens and N. C. Boulahia, "Formal enforcement and management of obligation policies", Data & Knowledge Engineering, vol. 71, no. 1, **(2012)**, pp. 127-147.
[10] J. Bentahar and M. E. Menshawy, "Communicative commitments: Model checking and complexity analysis", Knowledge-Based Systems, vol. 35, no. 11, **(2012)**, pp. 21-34.
[11] N. Fornara and M. Colombetti, "Specifying and enforcing norms in artificial institutions: A retrospective review", Proceedings of the 9th International Workshop on Declarative Agent Languages and Technologies, **(2012)**; Taipei, Taiwan.
[12] M. E. Menshawy and J. Bentahar, "Reducing model checking commitments for agent communication to model checking ARCTL and GCTL*", Autonomous Agents and Multi-Agent Systems, vol. 27, no. 3, **(2013)**, pp. 375-418.

[13] A. Gunay and P. Yolum, "Constraint satisfaction as a tool for modeling and checking feasibility of multiagent commitments", Applied Intelligence, vol. 39, no. 3, **(2013)**, pp. 489-509.
[14] C. Baier, J. P. Katoen and K. G. Larsen, "Principles of Model Checking", The MIT Press, **(2008)**.
[15] E. M. Clarke, O. Grumberg and D. A. Peled, "Model Checking, Cambridge, MA: MIT Press, **(1999)**.
[16] E. Kholy, W. Bentahar, J. E. Menshawy, M. Qu, H. Y. R. Dssouli, "Conditional Commitments: Reasoning and Model Checking", ACM Transcations on Software Engineering and Methodology, vol. 24, no. 2, **(2014)**, pp. 1-43.
[17] http://baike.baidu.com/view/3104112.htm.