Multi-agent Oriented Policy-based Management System for Virtual Enterprise

Jun Hu
College of Information Science and Engineering, Hunan University, Changsha, China
Email: hujun_111@126.com

Yinghui Song, Ye Sun
College of Information Science and Engineering, Hunan University, Changsha, China
Email: songyinghui2868@163.com, sunyeforever@126.com

***Abstract*—Virtual enterprise is a temporary organization consists of some independent enterprises, aiming to share technology, cost and other resource. It is predicted to be the most important business organizational patterns in the 21st century. To up speed the development of virtual enterprise, this paper presented a policy-based multi-agent management system to simulate the manipulation of virtual enterprise. Firstly, a hierarchical policy specification is proposed to control the behavior of agent. Secondly, the detection algorithms and resolution strategies are described for the conflicts that may be brought about by hierarchical policy. Thirdly, a policy admin tool to simplify the operations of policies is presented. Each enterprise registered in this tool is assigned an agent; policies made by enterprises are executed by agents correspondingly. Subdivision of application level and representation scope of policy leads to that policy-based mechanism not only provides good system control on autonomy agents, but also ensures the flexibility of agents; and gives the solution for policy conflicts at the same time. Finally, the main characteristics of this system are discussed by an example.**

***Index Terms*—virtual enterprise, hierarchical policy specification, multi-agent system, policy conflict**

I. INTRODUCTION

The concept of virtual enterprise (VE) is generated along with the agile manufacturing, which is proposed on the purpose of improving competitive position and cultivating the competitive advantage of manufacturing. With the rapid development of Internet, VE will be one of the most important business organizational patterns in the 21st century because it's unique advantages of optimization and integration of resources and rapid response to market [1]. Members of VE join in or quit a VE freely — they join up to get more benefit for themselves, and once the target achieved, they quit. In this case, the alliance is readily to achieve resource complementary, and amass the best capacity of design, productive and marketing. How to obtain more useful resource becomes the biggest problem.

In view of autonomy, sociability, pro-activeness and other features of agents, some scholars have proposed multi-agent system combining policy specification to simulate the manipulation of VE to help participants to get more resource and improve their competitive ability [2~5]. The behaviors of agents are so identical to VE members, that it couldn't be better to use agents to simulate VE members. But sometimes policies are too strict for agents and they prevented the flexibility of agents. Therefore, some adjustment is necessary for policy specification to fit VE system well.

This paper gives full consideration on the characters of VE, and proposes the concept of "organizer", which are also members of VE but have higher credit than normal VE members. Organizers should make policies on the view of the overall benefit of VE. In this way, the policy mechanism subdivides the representation scope and application level of policy, so it is not only providing VE good control on autonomy agents, but also ensuring flexibility of agents.

In this paper, we firstly present the application environment in part II. Then, a hierarchical policy specification is displayed in part III and methods to detect and resolve policy conflicts which may be brought about by policy are showed in part IV. In part V, a policy admin tool introduced to supervise users, policies, agents, and materials in integrate. In part VI, the related works are introduced. Conclusions and future work are represented in the last part.

II. STRUCTURE OF VIRTUAL ENTERPRISE

VE consists of various enterprises including vendors, customers, partners or even competitors, they ally to get more benefit and spend less cost. The most significant feature of VE is dynamic — individual enterprise joins in the alliance for its own purpose, once its target achieved, it exits the alliance. Besides, VE is a competitive organization. Members of VE share resources in the condition that all members are kindly and resources are adequate. The overall benefit of VE will surely be

Supported by the National Natural Science Foundation of China under Grant No.60773208; the Hunan Provincial Natural Science Foundation of China under Grant No.11JJ3065; the Specialized Research Fund for the Doctoral Program of Higher Education under Grant No.20070532075; Hunan Province Young Core teacher training project.

doi:10.4304/jsw.7.10.2357-2364

improved. But resources are not always adequate, so it is likely that two or more members compete on a same resource. If such competitions are not controlled, the organization may soon paralyses for resource exhaustion. Further more, control force of VE is very weak for lacking of unified management. So, it is very easily to be attacked by unscrupulous members. Therefore, it is essential to improve management ability of VE.

To assure the character of autonomic for VE, this paper takes multi-agent system to simulate its behavior, in which, each member joins in VE is assigned an agent in the system. Member of VE makes policies on its own agent; and the agent follows the instructions that policies indicate it to do or prevent it to do.

To assure security and stability of VE, the concept of "organizer" is proposed in this paper to control the behaviors of volatile members. Organizers are assumed to be honest, integrity, justice, and they make policies from overall interest of VE. The policies made by organizers are called organizational-level policies (OLP). In contrast, normal members of VE are called individuals; and the policies made by individuals are called individual-level policies (ILP). ILP are always on behalf of individuals. So, the structure of application environment for VE is shown in Fig.1. In this system, all behaviors are acted by agents; they receive demands of ILP to get interests for individual on one hand, and they must not violate the instructions of OLP on the other hand.

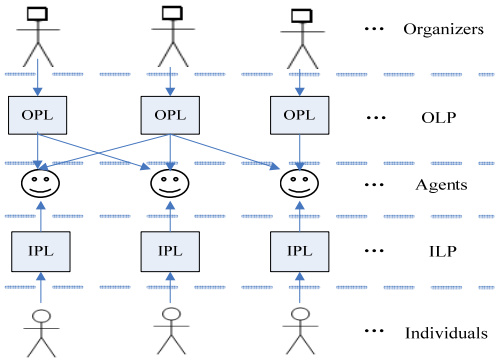

Figure 1. Application environment of hierarchical policy for VE

III. Hierarchical Policy Specification

According to the maker, policies are categorized to ILP and OLP. In order to increase the flexibility of system, this paper classifies the ILP into two layers — low layer and high layer: policies of low layer (LILP) are used to control the concrete behaviors of individual agent, which are enforced to do or not to do. ILP of high layer (HILP) are used to guide agent to generate appropriate plans on low layer policies, and prevent agent from executing actions blindly. Situations of system are always unpredictable, so policies of high layer are very useful in this case.

A Types of Hierarchical Policy

According to the analysis above, policies are classified into 3 categories 8 types (TABLE I).

TABLE I.
Type Of Policies

Application\ Abstract	Low layer policies	High layer policies
Individual-level policies(ILP)	APolicy, OPolicy, PPolicy	UPolicy, TPolicy
Organizational-level policies(OLP)	GPolicy, WPolicy, RPolicy	

LILP include authorization policies (APolicy), obligation policies (OPolicy), and prohibition policies (PPolicy). APolicy specify performance that the subject must do; OPolicy identify actions that are obliged; and PPolicy indicate motions that are prohibited to do. They are all direct order.

There are two types of HILP: target policies (TPolicy) and utility policies (UPolicy). TPolicy specify the goal of an agent, they are made up of obligation policies; UPolicy are used to evaluate the utility of policies. Utility relates to resource that individual enterprise care about, such as price and time. Different enterprises have distinctive demands on various resources, so "weights" on disparate resources are different. On the basis of UPolicy, agents can generate plans to fulfill TPolicy at maximum utility.

ILP often perform the interests of individual, while OLP are designed to restrict the behaviors of individuals.

OLP include role policies (RPolicy), grant policies (GPolicy) and withdraw policies (WPolicy). RPolicy are used to assign roles for VE members. GPolicy are used to grant agents authorizations or obligations, while WPolicy are used to revoke authorizations or obligations from agents.

B Hierarchical Policy Attributes

There are several basic attributes which almost all policies have, such as subject, object, action, cons, time, maker, status, etc. So, before policy specification, some attributes are introduced briefly as follows to help readers to understand.

"Type" can take any value of the eight types above. Usually, the role of policy owner restricted types of policies he owns.

"Subject" indicates the actor of policy, and "object" indicates recipient. Subject and object may be a single enterprise or a role in VE. "Role" is associated with a set of obligation policies and authorization policies. When playing a role, an agent inherits the set of policies related to this role.

"Action" describes operations that policy subject is about to do on the object. It has two elements: name and effects. When a policy is employed, actions that indicated by "name" are trigged and system status are changed responding to "effects".

"Time" is an ordered binary array: $< t_s, t_e >$, where t_s indicates the effective start time of a policy, and t_e indicates the effective end time of a policy. The policy is to be executed only if, it lies in the interval of t_s and t_e.

"Cons" specifies requirements that must be satisfied before a policy is executed; it consists of a series of Boolean expressions. An atomic cons expression has the

form of $x_i \triangleright \tau$, which is remarked as C. x_i are property variables of entity or resource in system, τ are items of predicate logic, they might be constants, variables or even functions of other properties. x_i and τ are connected by $\triangleright$, where $\triangleright \in \{=, \neq, <, \leq, >, \geq\}$. Cons also may be compound expressions of composed atomic expression, which composed of atomic expressions, using $\neg$, $\wedge$ or $\vee$ and braces recursively.

"Maker" is the editor of the policy. The role of makers limited the types of policies. In other words, individual enterprise managers could only make policies of ILP, and OLP could only be made by organizers. Usually, the role of a maker determines the priority of a policy. When conflicts happen, the results always depend on the maker who has high priority.

"Status" indicates the status of a policy. At any time, "status" may take one of the enumeration values: {INITIALIZED, READY, EXECUTING, DONE}. When a policy inserted into the system justly, the status is "INITIALIZED", when the current time is equal or greater than t_s, the status is triggered to "READY", when the current time is equal or greater than t_e, the status is triggered to "DONE". A policy is to be executed only in the status "READY". The detailed procedure of policy execution will be described later.

C Hierarchical Policy Specification

APolicy, OPolicy and PPolicy are LILP. They have the same attributes like: subject, object, acts, time, cons, and status. If a department of assembling cell phones called Assembler1, is obligated to buy charger4 and if the stock count of charger4 is less than 50, then he can make a OPolicy to his agent. The policy made in object-oriented language is shown in Policy1. According to that, we can get that Assembler1 makes this policy to be effective from *2011-12-12 00:00:00* to *2012-12-12 00:00:00*, and it is going to buy charger4 from a producer, if and only if, the stock count of charger4 is less than 50, the price of charger4 which belongs to producer is less than ￥10, and the credit of producer is no less than 2.

```
OPolicy1{
        Subject: Assembler1
        Object: producer
        Acts: buy charger4
        Time: <2011-12-12 00:00:00, 2012-12-12 00:00:00>
        Cons: this.charger4.count<50 &&
                producer.charger4.price<10 &&
                !（producer.credit<2）
        Maker: Assembler1
}
```

Policy1. Example of OPolicy

```
APolicy1{
        Subject: Assembler1
        Object: vendor
        Acts: sell Phone1
        Time: <null, null>
        Cons: this.Phone1.count>1000
                && this.Phone1.price>=800
        Maker: Assembler1
}
```

Policy 2. Example of APolicy

Policy2 describes an APolicy of Assembler1 to sell Phone1 on internet from with the condition that his store count is bigger than 1000, and price is more than ￥800.

The main attributes of UPolicy is a valuation function. The evaluation function is composed of pairs of parameters and values. The parameters may take the value of price, time and other resources that the individuals care about. For different enterprises, they have different demands for different resource. So, different individuals will assign different values on different resources. But there is a rule that all users should comply with: the sum of all values equal 1. The following formula is a valuation function made by Assembler1.

Fun1 = 0.5*price+0.2*size + producer.credit*0.3

TPolicy has same attributes as OPolicy, but the execution of TPolicy and OPolicy are different. In the policy-based multi-agent system, OPolicy are bounded to be executed only if the conditions are satisfied; APolicy are not bounded to be executed. Whether an APolicy is to be executed depends on TPolicy and UPolicy, that is to say, according to TPolicy, only when the value of an APolciy is high that it is to be executed.

GPolicy and WPolicy have the same form as LILP, but the value of action can only be "authorize", "obligate" or "prohibit". For GPolicy, system not only stores it to database, but also generates a corresponding LILP for the related subject. Similarly, system stores WPolicy to database, and checks if there are policies that satisfy the conditions. If such policies exist, the system will call appropriate executions.

RPolicy are used to assign roles for VE members. Its representation form is: P= (type, subject, role, cons, maker). When cons are satisfied, subject is given the "role". This brings convince for the system to separate different types of users.

IV. Conflict Detection And Resolution

Enterprises in VE may be interested in a same resource. So, policies made by different enterprise are readily to conflict. How to detect and resolve conflicts has become a most important criterion for evaluation of policy-based system. This part we will begin with policy conflict definition, and then conflict detection algorithms and conflict resolution methods are introduced.

A Definition Of Conflicts

When an agent is running in the system and executing a policy, it will violate one or more policies in the system, and then conflicts happen.

Definition 1: policy *p1* = (*type1*, *subject1*, *object1*, *action1*, *cons1*, *maker1*, *time1*) and *p2* = (*type2*, *subject2*, *object2*, *action2*, *cons2*, *maker2*, *time2*) conflict if the following conditions are true:

a) *subject1= subject2* and *object1= object2*;
b) *action1 = action2*;
c) *type1* =F and *type2* ∈ {O,A} or *type2*=F and *type1* ∈ {O,A};

d) overlap(*time1*, *time2*);

e) $cons1 \wedge cons2 \neq \varnothing$

Usually, it is easy to detect conditions (a) ~ (d), but cons consists of complex Boolean expressions, so optimization is necessary before (e) is detected.

B Optimization For Cons

Cons consist of complex Boolean expressions, with $\neg, \wedge, \vee$ in between. Cons are of much importance in policy specification. A policy is to be taken only if the cons are true. Further more, when comparing two policies, the most difficult part is to compare the cons. So, before conflicts detection, optimization is necessary. The optimization algorithm described in the main 5 steps:

Step 1: Eliminate operation "$\neg$";

If C_i is a atomic formula, then $\triangleright: =, \neq, <, \leq, >, \geq$ is been transformed to $\neq, =, \geq, >, \leq, <$;

If C_i is a compound expression, then use Morgan Law to process the expression recursively, as formula (1) and (2):

$$\neg(X \vee Y) \Rightarrow \neg X \wedge \neg Y \qquad (1)$$

$$\neg(X \wedge Y) \Rightarrow \neg X \vee \neg Y \qquad (2)$$

Step2: Remove inequality operator "$\geq$", "$\leq$" and "$\neq$";

Expressions $S_i \geq \tau$, $S_i \leq \tau$ and $S_i \neq \tau$ was transformed as formula (3) ~ (5)

$$S_i \geq \tau \Rightarrow (S_i > \tau) \vee (S_i = \tau) \qquad (3)$$

$$S_i \leq \tau \Rightarrow (S_i < \tau) \vee (S_i = \tau) \qquad (4)$$

$$S_i \neq \tau \Rightarrow (S_i < \tau) \vee (S_i > \tau) \qquad (5)$$

Step3: Transform the whole expression into disjunctive normal forms (DNF);

If there is an expression:

$C = ((P \wedge Q) \vee R) \wedge (S \wedge (R \vee T))$, in which *P*, *Q*, *R*, *S*, *T* are all atomic formulas, the steps of transformation are as follows:

$$((P \wedge Q) \vee R) \wedge ((S \wedge R) \vee (S \wedge T)) \qquad (6)$$

$$((P \wedge Q \wedge S \wedge R) \vee (P \wedge Q \wedge S \wedge T) \vee (R \wedge S \wedge R) \vee (R \wedge S \wedge T)) \qquad (7)$$

Step4: Eliminate the redundant items.

There may be the same item in a condition expression, so when it was transformed to a DFN, there might be more than one copy of a variable in the same conjunctive formula. So, elimination of the redundant items is necessary for avoiding duplicate computation in future work. The elimination of (7) is as follows:

$$((P \wedge Q \wedge S \wedge R) \vee (P \wedge Q \wedge S \wedge T) \vee (R \wedge S) \vee (R \wedge S \wedge T)) \qquad (8)$$

Step5: Sort each CNF in alphabetical order, and then sort the whole DNF.

The expression above is already in alphabetical order, so there is nothing more to do with it.

Through these optimizations, the cons are transformed to DNF, whose sub expressions are conjunctives of atomic formulas. There are several reasons for transforming cons to this form. Firstly, comparing the cons of two policies is much more readily than comparing the original cons; secondly, the DNF are easily for checking whether the cons of a policy is satisfied—as long as one conjunctive of the DNF is true, the cons is satisfied; Last but not least, to judge the value of a conjunctive is actually to judge whether the conjunctives of atomic formulas have solutions, as a matter of fact, the atomic formulas are inequalities or equalities, so the problem is finally translated to solvable mathematical questions.

C Conflicts Detection Algorithm

As shown in definition 1, conflicts arise in two policies when their attributes are same or overlap. Conflicts are classified to "static conflicts" and "dynamic conflicts" in line with inevitability of them.

If two policies have contrast type, and all other conditions are the same, then conflicts are bounded to happen, we call these conflicts static conflicts; But when there are only interactions on each condition, conflicts may not happen, we call these conflicts dynamic conflicts.

As for static conflicts, detection will be taken before a policy is inserted into system. Presuming that the P is the policy that will be inserted, and P_i (i =1, 2,…, n) are policies which are already exist in a system. Then, check each P_i with P to make sure whether there are conflicts. If conflicts exist, the system will do treatment according to conflicts resolution strategy; otherwise insert P into the system. The algorithm of detecting P and P_i are presented in Algorithm 1, in which C and C_i are optimized cons expressions of P and P_i.

```
isStaticConflict(P,Pi){
    if(either the subject, object, action or time of P and Pi are not the
        same) return false;
    if(C.length!=Ci.length) return false;
    else{
        while(!C.end()){
            A=C.next();
            While(!C.end()){
                B=C.next();
                if(A.equals(B)){C.reset();break}//out the loop
            }
            If (C.end()) return false;
        }
        If (C.end()) return true;
        Else return false;
    }
}
```

Algorithm 1. The algorithm of detecting static conflicts

Detection of dynamic conflict is more difficult, this paper creates a running time monitoring thread to solve this problem. Right before a policy is executed, the thread checks if conflicts will happen. As a matter of fact, values of all policy variables are mostly assumed before executed. So, the algorithm of detection is the same with static conflict detection.

D Conflict Resolution

Various policies result in various conflicts, there are three types: conflicts between OLP and OLP (OOC), conflicts between ILP and ILP (IIC), and conflicts between OLP and ILP (OIC). To ensure the flexibility of multi-agent systems, different approaches are taken to deal with different types of conflicts.

Because all organizers considered from the overall interest of the system, so this paper take harmonization algorithm, which is provided in [6], to resolve OOC. The algorithm changes the policy of low priority to one or more policies to avoid collision. For space limitation, details of the algorithm were not described here.

According to the simulation system we built, organizers are used to guarantee the safety of the system that given an OLP a higher priority. When OIC happens, the policy of ILP is deleted and the policy of OLP is kept.

Because individual enterprises of VE usually have interest conflict, IIC are more than OOC and OIC. One way to resolve these conflicts is to use harmonization algorithm; but the simplest one is to delete low priority policy. Because there are so many conflicts of IIC, sometimes it is more preferable to use the latter approach. Which options to choose depends on the requirements of the system users.

V. POLICY ADMIN TOOL

This policy admin tool bases on multi-agent system to simulate the behaviors of virtual enterprises. All members of VE are assigned an agent. This system takes production-demand-sales integration of phones as an application scenario to make and manage policies.

A Architecture

The architecture of policy admin system is divided into four parts: User Supervise Module, Policy Management Module, Agent Management Module, and Material Management Module. The architecture is shown in Fig.2.

According to the division of system logic structure, the user is divided into two classes: ordinary enterprise users (called individuals in the article) and administrators. The policy management tools have two different clients. This module implemented four sub-modules: register module, log in module, information management module and log out module. Every logged in users are of authorization to administrate policies, supervise agents and manage materials information. But for different users, the operations that they can perform are not the same which depends on the role of user.

Policy Management Module contains Policy Specification, Policy Info Management, Policy Execution, Conflict Detection, and Conflict Resolution. Most of these modules have been represented in part III and IV.

Agent Management Module is used to manage agents' information and control behaviors of agents. It includes several parts: agent naming, agent life cycling management, agent behaviors management, and agent role management, etc. The most important parts are agent assignment, agent start/stop, and agent revoke. Only when the agent is started, he could execute policies.

Material Management Module is used for users to store the materials of his company or department. It is only a subsidiary module in this system. The only thing we should know is that, it stores some attributes of a material such as material name, material code, color, price, and quantity. Some of these attributes are useful when executing a policy.

B Key Technology

The key technology of this paper mainly includes policy specification, policy conflict detection and resolution, policy execution, and management of multi-agent.

This paper takes object-oriented language as policy specification language. That is, any object-language is able to implement policy discussed in this paper.

The policy conflict detection and resolution are described in part IV.

An existing plug-in – Java Agent Development Framework (JADE) – is taken in this paper as work bench. JADE is a software development framework aimed for developing multi-agent systems and applications conforming to FIPA [7] standards for intelligent agents.

As we know in part III, the status value of a policy indicates the status of a policy. In Fig. 3, we'll see how "value" changes in the system.

When a policy is inserted into a database, the status is defined INITIALIZED. When the current time is bigger than start time of a policy, the database will trigger an event to change the policy status to READY. When the current time is bigger than end time of a policy, the database will trigger an event to change the policy status to DONE. The figure simplified the change of status from READY to EXECUTING, because the changing process involves behaviors of agents.

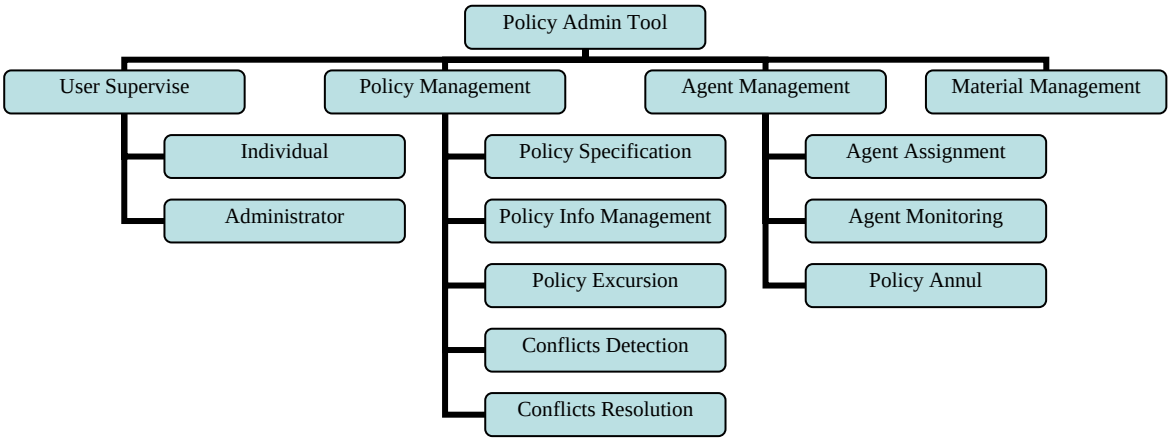

Figure 2. The architecture of policy admin tool

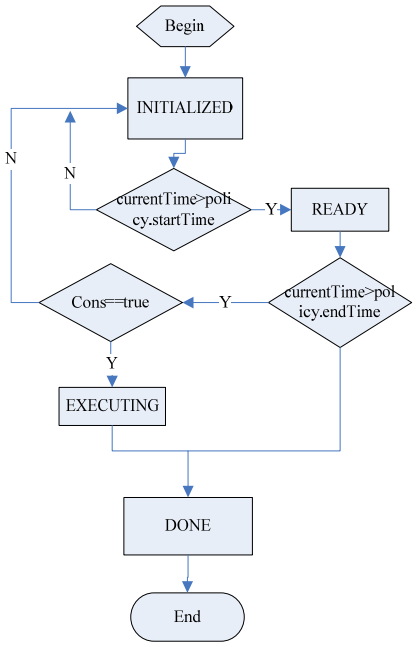

Figure 3. The changing procedure of policy status

As a matter of fact, the changing process of policy status is played by agent; and the execution procedure of an activated agent is shown in the next pseudo-code. Further more, other actions acted by agents are showed in this figure.

A term called ReadyPolicyQueue is used in this pseudo-code. It is a global queue that stores policies which have the status READY. When the status of a policy is READY, it is added to the ReadyPolicyQueue. An agent stops executing this cycle until it is terminated.

Agent executing procedure:

Procedure1: agent view ReadyPolicyQueue to ensure that whether the policy is READY. If the queue is not null, agent choose a policy, go to Procedure 2; else wait a given interval and go to procedure1.

Procedure2: check the cons of the policy; if cons are not satisfied, go to procedure3; else go to procedure4.

Procedure3: insert this policy to the end of ReadyPolicyQueue, wait a given interval and go to procedure1

Procedure4: change the policy status to EXECUTING, and execute the policy, go to procedure5.

Procedure5: if the policy is executed successfully, change the policy status to DONE; go to procedure1.

C Case Study

There are 3 types of roles in production-demand-sales integration of phones scenario: mobile phone components manufacturer (Manufacturer), mobile phone assembler (Assembler) and mobile vendors (Vendor). In the system, users make policies to his agent; observe the interactions of agent in time; and make new policies to agent timely to change the behavior of agent.

As we all know, mobile phones consist of many components, such as IC card, battery, charger, Bluetooth adapter, data cable, LCD screen, etc. Assemblers buy different components from different manufacturers and assemble them to various available mobile phones that are in different type, color, and price; and sell these phones to separate vendors.

Recently, Assembler1 has assembled excess quantity of Phone1, and his production team has consumed so many chargers and he want to buy some charger4 on internet, so he made an OPolicy by the admin tool. The implementation to insert this policy is shown in Fig.4.

Figure 4. The portal of policy admin tool

In order to ensure the interests of all manufactures in VE, the administrator is going to make a GPolicy that allows all manufactures to sell charger4 at price that more than 12. Then, this GPolicy conflicts with OPolicy1 which is made in the system (see Policy1). When the administrator clicked the button “Conflict ?”, the admin tool system starts conflict detection algorithm to detect whether conflicts will happen; button “Submit !” is used to starts the algorithm of inserting a policy. If conflicts exist, a warning message will be sent to the user, but whether a policy is inserted into the system, it depends on the priority of user and conflict type. If a potential conflict is exists, a warning message is sent to the user using FIPA with the ACLMessage.INFORM (To get more information, please refer to [7]).

VI. Related Work

Most of substantial systems have taken policy mechanisms to control the behavior of agents; but only a few of them have proposed conflict detection methods and resolutions means to resolve the side effect of policies mechanism. These systems are categories into two groups: policy-driven systems and norm-governed systems.

KAoS, Rei, and Ponder are three typical and popular policy languages of policy-driven systems [8]. KAoS uses OWL as its basic language; but OWL is an incomprehensible machine language, so the implementation of KAoS is hard to be promoted. The advantage of Rei is its clear expression; it departed the specification of ontology and policy [9]. Ponder is an object oriented policy language for the management of distributed systems and networks [10~12]. yLopez and Grossi[13~14] provide norm-governed languages for

MAS. yLopez thought that the ability of a single agent is limited, they join in the society to obtain his own profit. yLopez takes an abstract language like human laws to specify policies, but it doesn't describe the policy language in detail, so it is impracticable. Grossi views norms from two implementing aspects: regimentation and enforcement [15], but it doesn't provide policy specification.

KAoS proposed a method called "harmonization algorithm" to solve policy conflicts at a minimum cost. But the algorithm to detect conflict pays no attention to reasoning of conflict condition, so it brings certain miscalculation. Rei takes an algorithm similar with KAoS to detect conflicts and has same defaults as KAoS. Ponder defines conflicts as user type defined by users. It did not modify conflicted policies, but execute them at different time or different space. But it still changes the attributes of policies. Up to now, there are more and more researchers done in conflict detection and resolution methods, such as researchers in article [16-17].

There is little research on policy graphic user interface. A most successful graphic user interface is designed by team of Ponder researchers. Ponder supplied a tool of graphic user interface to manage policy. It has brought much convenience to policy users, but there is a little problem: to make policies using this platform, users have to know specialized knowledge of Ponder. For this reason, that Ponder is hard to be a popular policy admin tool in real world.

To simulate the manipulation of VE, intelligent agents must be adopted to help make decisions for system participants. Many articles have introduced multi-agent to simulate the actions of VE. But almost all of them take VE system as a harmonious distributed system and do not notice that there might be wicked members in VE. Therefore, this paper takes "hierarchical" policy to restrict the actions of agent. The idea of "hierarchical" origin from our former research: PRAL (Policy Representation and Assignment Language) and LPRF (Layered Policy Representation Framework) [18~19]. PRAL defines policy as a concept of embedded ontology; it has three types of policies: Authorization policies, Obligation policies, and Negotiation policies, the former two policies are basic policies for agent to execute, and the last are used to direct the agent's behavior during negotiations. LPRF promoted the concept of "layer" formally. It divides policies into three layers: constrained behavior layer, strategy layer and autonomy layer. Constrained behavior policies display the external behavior of subject of policy from the view of macro, and strategy layer policies that allow agent to take decision according to internal conditions or states of itself, it is micro policies. Autonomy layer policies are built on the other two types of policies, which are used to guide the behaviors of agents more intelligently.

VII. Conclusions

To get more service and resource from internet, policy mechanism combined with multi-agent system is a good choice. But most policy mechanisms lack of detailed formalized description and don't take the specific application environment into account. Especially, little research takes these parts as integration.

This paper provided a multi-agent oriented policy-based management system for virtual enterprise. By giving full consideration to the characters of virtual enterprises, hierarchical policy describes the policies in two application levels and two abstract layers, and it not only keeps the flexibility of agents, but also controlled their behaviors well. For the conflicts which may be brought about by policy specification, conflicts detection and resolution methods are applied. Also, a policy admin tool that using multi-agent system to simulate the behaviors of enterprises has been presented.

In conclusion, there are still many theoretical studies need to be improved in spite of the valuable or useful part in the paper. For example, trustworthy evaluation in policy-based system will be future work.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant No.60773208; the Hunan Provincial Natural Science Foundation of China under Grant No.11JJ3065 the Specialized Research Fund for the Doctoral Program of Higher Education under Grant No.20070532075"; Hunan Province Young Core teacher training project. Also, we would like to give special thanks to Lei Deng for the useful comments on the manuscript.

References

[1] Heping Zhong. C.: Research on Incentive Contracts for Partners within a Virtual Enterprise[C]. //In: 2010 International Conference on Networking and Digital Society. 2010:384--387

[2] Basile C, Cappadonia A, Lioy A. C.: Geometric Interpretation of Policy Specification[C]. //In: Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks. Washington, DC: IEEE Computer Society. 2008:78--81

[3] Jingfan Tang, Ming Xu, Ming Jiang. C.: Towards Workflow Oriented Virtual Enterprise Based on Policy Driven and Multi-agent Service Composition[C]. //In: Fifth International Conference on Fuzzy Systems and Knowledge Discovery. 2008:495-499

[4] Jun Hu. Research on Autonomic Computing Oriented Policy-based Multi-agent Cooperation System[D] Hangzhou, China, 2006

[5] Yathiraj B. Udupi, Munindar P. Singh. C.: Multiagent Policy Architecture for Virtual Business Organizations. In: IEEE International Conference on Services Computing Piscataway, NJ: IEEE ,2006

[6] Uszok A, Bradshaw J, Jeffers R, et al. C.: KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, deconfliction and enforcement[C]. //In: Proceedings of IEEE Workshop on Policy. Washington, DC: IEEE Computer Society. 2003:93--96

[7] The Foundation for Intelligent Physical Agents[G/OL]. Available at http://www.fipa.org . 2010-1-15

[8] Tonti G, Bradshaw J M, Renia J, et al. J.: Semantic Web Languages for Policy Representation and Reasoning: A

Comparison of KAos, Rei, and Ponder [G]. Lecture Notes in Computer Science. Berlin: Springer-Verlag. 2003:419--437
[9] Kagal L, Finin T, Joshi A. A Policy Language for a Pervasive Computing Environment [A]. In: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks[C]. Washington, DC: IEEE Computer Society. 2003:63-75
[10] Lymberopoulos L, Lupu E, Sloman M. PONDER Policy Implementation and Validation in a CIM and Differentiated Services Framework [A]. In: Network Operations and Management Symposium[C]. South Korea. 2004:31--44
[11] Dulay N, Lupu E, Sloman M, et al. A Policy Deployment Model for the Ponder Language [A]. In: Network Management Proceedings[C]. Seattle, WA. 2001:529-543
[12] Dulay N, Lupu E, Sloman M, et al. C.: A Policy Deployment Model for the Ponder Language. In: Network Management Proceedings. Seattle, WA. 529--543(2001)
[13] Lopez F L, Marquez A A. An Architecture for Autonomous Normative Agents[C]. //In: Proceedings of the Fifth Mexican International Conference in Computer Science. Washington, DC: IEEE Computer Society, 2004: 96-103
[14] Lopez F L, Luck M, Dinverno M. A Normative Framework for Agent-Based Systems [J]. Computational & Mathematical Organization Theory, 2006, 12(2): 227-250
[15] Grossi D, Aldewereld H, Dignum F. Ubi Lex Ibi Poena: Designing Norm Enforcement in Electronic Institutions [A]. In: Lecture Notes In Artificial Intelligence[C]. Berlin: Springer-Verlag. 2007:101--114
[16] Yi Ren, Fangquan Cheng, Zhiyong Peng, Xiaoting Huang, Wei Song. A privacy policy conflict detection method for multi-owner privacy data protection. Electronic Commerce Research. 2011,11(1):103-121
[17] Apurva Mohan, Douglas M. Blough. An attribute-based authorization policy framework with dynamic conflict resolution[C]. //Proceedings of the 9th Symposium on Identity and Trust on the Internet. New York: ACM, 2010:37-50
[18] Jun Hu, Wang Baiyun. C.: LPRF: A Layered Policy Representation Framework[C]. //In: 2009 World Congress on Computer Science and Information Engineering. Washington, DC: IEEE Computer Society. 402-406(2009)
[19] Jun Hu, Ye Sun, Baiyun Wang. Research on Policy Conflict Based on Layered Policy Representation Framework[C]. //Proceedings of the Fifth International Conference on Semantics, Knowledge and Grid. Los Alamitos, CA: IEEE Computer Society Press. 2009:144-151

Jun Hu born in 1971 and received M.Sc. in Computer Application from Kunming University of Science and Technology, Kunming, China, and Ph.D. in Computer Science and Technology from Zhejiang University, Hangzhou, China. In 2010, he was an academic visitor at University of Southampton working on multi-agent system. Currently, he is an associate professor of Hunan University, Changsha, China. His research interests are in multi-agent system, distributed artificial intelligence and software engineering.

Yinghui Song born in 1985 and is a Master of Computer Application from Hunan University. Her main research interests include distributed artificial intelligence, multi-agent systems, and machine learning.

Ye Sun born in 1983 and received M.Sc. in Computer Application from Hunan University, Changsha, China. His main research interests include artificial intelligence, multi-agent systems and Virtual reality. He holds a post in SuperMap from July, 2007 until now, and mainly works on developing software for three-dimensional effects and Virtual reality.