An Automatic Policy Refinement Mechanism for Policy-Driven Grid Service Systems

Conference Paper *in* Lecture Notes in Computer Science · November 2005

DOI: 10.1007/11590354_23 · Source: DBLP

2 authors, including:

Ji Gao
Community College of Philadelphia
134 PUBLICATIONS **818** CITATIONS

An Automatic Policy Refinement Mechanism for Policy-Driven Grid Service Systems*

Bei-shui Liao and Ji Gao

Institute of Artificial Intelligence, Zhejiang University, Hangzhou 310027, China
baiseliao@zju.edu.cn, gaoji@mail.hz.zj.cn

Abstract. Currently, the management of grid services is becoming increasingly complex. To resolve this complexity problem, autonomic computing and policy-based multi-agent technology have been proposed as promising methods. However, there are many challenges to be resolved. Among them, policy refinement is a great problem that hampers the development of policy-based system. To cope with this issue, this paper presents a policy refinement mechanism based on recipes. Recipes define possible refinement alternatives for each abstract policy. And the policy refinement engine automatically refines the policies by choosing the refinement branch in terms of the conditions of each branch.

1 Introduction

In recent years, with the development of grid services [1,2], more and more large-scale and cross-organizational information systems are established. However, the management of these open, dynamic, and heterogeneous grid service systems is becoming increasingly complex. In order to treat this challenging problem, many research efforts have been made. Among them, autonomic computing [3,4,5] and policy-driven agent system [6,7] are promising examples. Due to the dynamic nature of virtual organization (VO), the policy-driven system will play an important role. As we know, from the users' perspectives, there are two desirable functionalities of the grid service systems within VO environment. First, specific virtual organizations should be formed quickly by dynamically organizing distributed partners to resolve the specific problems, according to high-level business requirements. And if the requirements vary at run time, the system is able to take these changes into consideration, and then modify the system's configurations to meet the new requirements. Second, the stakeholders of resource consumers or providers should be enabled to influence the behaviors of the underlying components at run time in terms of their real-time preferences and requirements. To meet these demands, policy-driven multi-agent system is an ideal candidate, as formulated in [6, 7, 8]. However, there are many challenges to be coped with to make the framework proposed in [8] more practical and intelligent. One of them is how to automatically refine high-level abstract policies to low-level concrete policies. In [8], human designers who model the policies are in charge of this part of work, which is a great problem that hampers the practical use of policy-driven system. So, our

* This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2003CB317000.

H. Zhuge and G.C. Fox (Eds.): GCC 2005, LNCS 3795, pp. 166 – 171, 2005.

ultimate objective is to make automated refinement of policies a reality based on policy refinement templates (recipes), and this paper is the initial work.

Until now, only few researchers have devoted their attention to policy refinement, e.g., Bandara [9] and Beigi [10] et al. In paper [9], the authors presented an approach to policy refinement that allows the inference of the low-level actions that satisfy a high-level goal by making use of existing techniques in goal-based requirements elaboration and the Event Calculus. This method is useful to make policy refinement automatic, but it requires that the policy-regulated objects are predetermined and specific, which often can't be met in the VO environment where the participants are dynamic. Paper [10] proposed three types of policy transformation approaches such as transformation using static rules, transformation by policy table lookup, and transformation using case based reasoning. Among them, the static rules based approach is the simplest and most efficient one, and is useful when the application context is static. But when the business requirements and application environment are dynamic, it is infeasible. So, as for dynamic VO environment, in which participating agents may join or leave at run time, we need a more flexible mechanism for policy refinement. This paper proposes a method that extends the rule-based method with dynamic characteristics, called recipe-based mechanism. Recipes defined at design-time consist of all possible refinement schemes under different conditions. Then, under run-time environment, a policy-refinement engine automatically produces specific refined policies according to the real-time conditions.

This paper is organized as follows. Section 2 proposes a recipe-based policy refinement mechanism, in which the policies, the definition of recipes and the principle of policy refinement engine are formulated respectively. Section 3 is conclusions.

2 Policy Refinement Mechanism

As proposed in [8], there are three levels of policies, i.e., abstract policies, concrete policies, and enforceable policies, in which enforceable policies are distributed to specific agents who are obliged to fulfill them, while concrete policies defined the duties and rights of roles will be enforced by role-enacting agents. When an agent enacts a role by negotiation and signing a contract, it will be obliged to fulfill the contract (the details of negotiation and contract-based cooperation are out of the range of this paper and are presented another paper [11]), thus enforce the concrete policies indirectly.

Now, the problem is how to automatically refine the abstract polices to concrete or enforceable ones. Due to the dynamic nature of virtual organization, the policy refinement mechanism should be context-aware. First, the mechanism should be flexible enough to reflect the environment variations. Second, the mechanism should be simple and efficient enough to ensure the process of policy refinement within tolerant time limit. We propose a recipe-based refinement mechanism to treat this problem, as shown in figure 1.

Central to this mechanism is the recipe-based approach. The idea of this approach is inspired by the styles of human treating complex problems. As we know, when a person deals with a complex problem, he should first have the knowledge about how to achieve it. This knowledge is not static, specific program to be executed, but a template

that defines possible plan steps. Which branch of plan steps is chosen at run time depends on the real time conditions. In this paper, we take recipes as templates of policy refinement. Each recipe defines all possible refinement plan steps of an abstract policy. Conditions of the plan steps that reflect the state of application context and environment are the basis for refinement engine's decision-making. As to a specific abstract policy, the refinement engine reasons about the abstract policy according to related recipes and the real-time context of the VO, and brings about concrete policies or enforceable policies. In this way, the outcomes of the refinement not only meet the goal of abstract policy, but also reflect the status of the underlying environment.

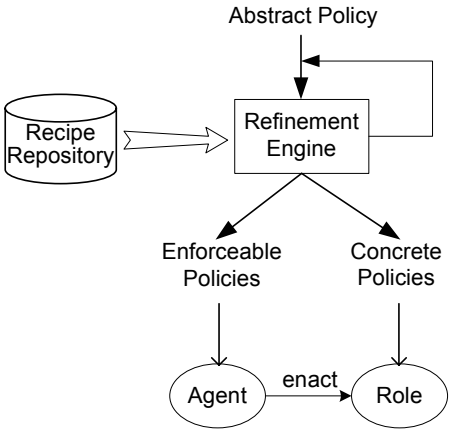

Fig. 1. Recipe-Based Policy Refinement Mechanism

Policy Definition. In this system, we extend the policy definition presented in our previous papers. The *Behavior* of a policy subject [6] is extended to *Processing*. There are three types of *Processing,* including abstract goals (prefixed by "$"), enforceable goals, and activities. Abstract goal can't be realized directly, and should be mapped to enforceable goals. We distinguish three levels of policy, including abstract policy (also prefixed by "$"), concrete policy, and enforceable policy. When a policy has one or more abstract goals, we call it abstract policy. Abstract policy is directly derived from business requirements. Concrete policy is refined from abstract policy. Its subject and target are roles of a VO. Enforceable policy is specific to individual agents and can be enforced by related agents. Now, we need an approach to refine an abstract policy to concrete policies or enforceable ones. In the following, we first give the definition of policy, and then formulate the policy refinement mechanism. The abstract policies, concrete policies and enforceable policies are specified in a unified form, as defined in the following in the form of EBNF[1].

[1] In EBNF, terminals are quoted; non-terminals are bold and not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([…]); components that can occur any number of times(including zero) are enclosed in braces ({…}). Whitespace is ignored in the productions here.

```
Policy  ::= 'Policy(' ['$'] PolicyID Modality  [Trigger]
            Subject {Processing} [Target]
            [Constraint]')'
Modality::= 'A+' | 'A-'| 'O+' | 'O-'
Trigger ::= 'Trigger(' AgentMentalState| TimeEvent ')'
Subject ::= 'Subject(' Role | AgentID ')'
Processing::= 'Processing(' ['$']Goal| Activity')'
Target    ::= 'Target(' Role | AgentID ')'
Constraint ::= 'Constraint(' LogicalExpression ')'
```

According to this definition, there are four kinds of policy (expressed by *modality*): positive authorization (A^+), negative authorization (A^-), positive obligation (O^+) and negative obligation (O^-). When a policy is abstract, it is prefixed with a symbol '$'. In each policy, there is a *PolicyID* that is globally unique to identify a policy, a *Trigger* that denotes the triggering conditions of a policy, and a *Constraint* that expresses the applicable conditions of a policy. In addition, the *Subject* and *target* of a policy can be a role or an agent, expressed by *Role* and *AgentID* respectively. Finally, *Processing* is the means taken by a policy subject. A *Processing* can be an abstract goal, an enforceable goal, or an activity, in which abstract goal is prefixed with a symbol '$'.

Recipe Definition. The refinement of abstract policies to concrete or enforceable ones is carried out by means of recipes, which are designed offline by system designer. A recipe is defined **in the form of EBNF** as follows.

```
Recipe   ::='Recipe('RecipeID AbstractPolicyID
           {PlanStep |'Loop(' PlanStep ')'}')'
PlanStep::= '(←' PolicySet [Condition] ')'|
            {'(←'PolicySet [Condition]')'}|
            '(←' 'Return(error)' Condition ')'')'
PolicySet::= Policy |'(' ('sequence'|'Concurrence') '('
            {'←'Policy [Condition] }|{Policy}')''')'
Condition::= 'Condition(' LogicalExpression ')'
```

In this definition, a recipe is composed of a *RecipeID* that identifies a recipe, an *AbstractPolicyID* that specifies the abstract policy to be refined, and some plan steps that defines possible refinement schemes of an abstract policy. In each plan step, there are a *PolicySet* and a *Condition*. When a *Condition* is satisfied, the related *PolicySet* is selected.

Policy Refinement Engine. As shown in figure 1, policy refinement engine (PRE) transforms abstract policies to concrete policies or enforceable policies with the support of recipes. When an abstract policy is put into a PRE, the PRE will select a recipe whose abstract policy ID (AbstractPolicyID) matches that of this abstract policy. And then, a policy refinement scheme will be produced according to the recipe and the conditions within each plan step. The PRE makes decisions by checking the conditions of each plan step, and forms a policy refinement scheme. In PRE, there is an algorithm for policy refinement. Let RCP be a set of available recipes; AP be a set

of abstract policies to be refined; A and NA be lists for storing on-going refinement policies and non-abstract policies respectively; T be a tree to record the refinement scheme of an abstract policy; p be an abstract policy to be refined. Formally, the algorithm is defined as follows.

Step1 Let A={p}, NA=∅, T.ROOT=p;
Step2 Take p out of the list A, and let p_0=p;
Step3 If p_0 is not an abstract policy, push it into the list NA and GOTO Step7;
Step4 If $\exists r_i \in$ RCP, such that r_i.AbstractPolicyID=p_0.PolicyID, then Input(r_i); else Return error and GOTO Step9;
Step5 If $\exists c_k \in r_i$.PlanStep.Condition = TRUE, then Output(c_k, PolicySet); else Return error and GOTO Step9;
Step6 Let the policies of PolicySet from Step5 be p_0's Children, adding to the tree T, and meanwhile push them into the list A;
Step7 Take the first element (denoted as f) out of A, if A≠∅ then Let p_0=f, GOTO Step3, else GOTO Step8;
Step8 Output T;
Step9 Stop.

In this algorithm, we adopt a width-first approach. At the first, an abstract policy (p) is refined to several sub-policies. Then, if there are some abstract policies among these sub-policies, the abstract policies are further refined to sub-sub-policies respectively. And then, if there are some abstract policies among these sub-sub-policies, the refinement will continue, until all policies are concrete or enforceable.

3 Conclusions

Policy-based management (PBM) is an important technology for simplifying the management of grid service systems. As a key part of PBM, policy refinement is still a challenging issue to be resolved. In light of the recipe-based idea, this paper presented a mechanism for automatic policy refinement. The contributions of this paper are two-fold. First, we proposed a notion of policy refinement recipe, which can not only express all possible refinement alternatives of a specific high-level policy, but also reflect the runtime context by conditions within plan steps. Second, a novel policy refinement mechanism and its reasoning algorithm are presented. By this mechanism, an abstract policy can be refined to different sets of policies under various real-time contexts, thus human administrator can only focus on defining high-level policies, without worrying about the underlying details. Compared to the mechanisms proposed by other works [9,10] mentioned in section 1, our method have the following advantages. On the one hand, in our policy definition, the subject and target of policy can be roles, which are mapped to specific agents at run-time by a mechanism proposed in the paper [8], so the specification and refinement of policies can meet the demands of VO environment where the participants are dynamic. On the other hand, the conditions within a recipe reflect the context of dynamic environment, thus our approach is more flexible and powerful than traditional static rule based methods [10].

References

1. I.Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.
2. I. Foster, C. Kesselman, J. Nick, S. Tuecke.The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
3. Jeffrey O. Kephart and David M. Chess.The Vision of Autonomic Computing. IEEE Computer, 36(1):41--52, 2003
4. Bei-shui Liao, Ji Gao, Jun Hu, Jiujun Chen. A Model of Agent-Enabling Autonomic Grid Service System. In proceedings of GCC 2004, LNCS 3251, pp.839-842, 2004
5. Bei-shui Liao, Ji Gao, Jun Hu, Jiujun Chen. A federated multi-agent system: Autonomic Control of Web Services.In Proceedings of the Third International Conference on Machine Learning and Cybernetics, pp.1-6 vol.1, 2004
6. Bei-shui Liao, et al. A Policy-Driven Multi-Agent System for OGSA-Compliant Grid Control. In the proceedings of 2004 IEEE International Conference on System, Man and Cybernetics: 5525-5530
7. Bei-shui Liao, Ji Gao. Dynamic Self-Organizing System Supported by PDC-Agents. Journal of Computer-Aided Design & Computer Graphics, to appear
8. Bei-shui Liao, et al. Ontology-Based Conceptual Modeling of Policy-Driven Control Framework: Oriented to Multi-agent System for Web Services Management. In: C.-H. Chi and K.-Y. Lam(Eds): AWCC2004, LNCS 3309, pp. 346-356, 2004
9. A. K. Bandara, et al. A Goal-based Approach to Policy Refinement. In Proceedings of 5th IEEE Workshop on Policies for distributed Systems and Networks (Policy 2004) 229-239
10. Beigi, M., Calo, S., Verma, D., "Policy Transformation Techniques in Policy-based Systems Management", in proc of 5th IEEE International Workshop on Policies and Distributed Systems and Networks, IEEE, 2004, pp 13-22
11. Beishui Liao, Ji Gao.A Model of Multi-agent System Based on Policies and Contracts. Proceedings of 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05), 15-17 September 2005, Budapest, Hungary, LNAI, Springer Verlag.