Governance of Cross-Organizational Service Agreements:
A Policy-Based Approach*

Yathiraj B. Udupi
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
ybudupi@ncsu.edu

Munindar P. Singh
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
singh@ncsu.edu

Abstract

Many real-life organizations are hierarchies of largely autonomous, heterogeneous members (individuals or other organizations), often exhibiting rich policies. We restrict our attention to organizations that monitor their environment, collate events, determine compliance of their behaviors with their policies, and potentially act in anticipation of events to ensure the satisfaction of their policies.

This paper models cross-organizational service agreements as resulting in the formation of organizations. This paper emphasizes the importance of proactive policy-based governance in organizations (modeled as multiagent systems) and provides a novel architecture supporting policy monitoring, governance, and enactment.

This paper provides an initial formalization and discusses the compliance and completeness of behaviors produced from specified policies. To demonstrate the practical utility of this approach, it is implemented using an existing policy engine and messaging middleware.

1 Introduction

Hierarchical service organizations, termed *Orgs* here, are structures of entities such as individuals and organizations that exhibit complex behaviors and rich policies. Many real-life organizations can be modeled as Orgs. In particular, Orgs apply wherever distributed resources need to be administered. In business settings, Orgs arise in scenarios such as cross-organizational service agreements, contracts, partnerships, supply chains, and so on. Because of the autonomy of the participants, *governance* is a better metaphor for administrating Orgs than traditional management. Governance involves the processes by which autonomous entities agree to collaborate and share resources. Complex relationships and contracts may bind the members of the Orgs within them. For example, Orgs apply in the area of provisioning of services, where sophisticated service level agreements are required to manage the resources and services controlled by multiple organizations. Business process outsourcing (BPO) relates to the recent expansion of services businesses. BPO involves refactoring an enterprise's processes so that external parties are engaged, yielding improvements such as efficiency and reliability. In general, service computing causes key processes to spread over multiple organizations, but present techniques do not handle them well.

We seek to improve the modeling and governance of complex Orgs through innovations in policy architecture. Traditional policy-based frameworks emphasize reactive behaviors, wherein an external request causes a policy engine to compute a response. Reactive policy frameworks are adequate for applications such as routing packets or controlling access to data items. However, Orgs in business or scientific scenarios inherently require richer policies and proactive behaviors. For example, in scientific computing, an Org representing a "production grid" would deal with resources (e.g., instruments in labs or observatories), representing the real world. In order to ensure the satisfaction of its policies and contracts, such an Org must not only respond to explicit requests, but must also monitor its environment, collate events, determine compliance of its interactions, and potentially act in anticipation of events. For example, a grid facility may assign preemptive priority to hurricane modelers, to react to emergent environmental situations so as to generate more accurate warnings for the public. Preemptive scheduling, though apparently simple, remains a major challenge in grid computing [5].

Modern needs, such as resource sharing in cyberinfrastructures, and accommodating IT and other service engagements, speak to the importance of policy-based governance. This paper emphasizes the importance of proactive policy-

*With partial support from NSF (grant ITR-0081742) and the Center for Advanced Computing and Communication (CACC), NCSU.

based governance in Orgs and provides an enhanced multiagent architecture supporting such governance. This work differs from earlier research on organizations and policies in the following key ways.

Policy formulation and enactment. This paper emphasizes not a policy engine but a multiagent architecture in which policies can be perspicuously formulated. It provides the operational semantics for the propagation, resolution, and enactment of policies.

Event-based monitoring. This paper enables the proactive behavior of Orgs in checking compliance with their policies, by enabling Orgs to communicate relevant events to each other.

Contributions. This paper extends Udupi and Singh's [9] proposal of policy-based multiagent architecture for organizations. It (1) formalizes organizations with a proactive agent-based policy architecture supporting a hierarchical model of policy monitoring, governance, and enactment; and (2) discusses the compliance and completeness of overall behaviors from specified policies. To demonstrate its practical utility, this approach is implemented using an existing policy engine and messaging middleware.

Organization. Section 2 formally describes Orgs and further motivates policy-based governance. Section 3 presents the proposed multiagent policy governance architecture. Section 4 evaluates the proposed approach conceptually and discusses some important consequences. Section 5 offers a comparative evaluation of the present approach with respect to the related work.

2 Hierarchical Organizations

Our approach is centered on the notion of *agents*. Orgs are multiagent systems, each individual and Org being modeled (recursively) as an agent. An Org in this sense either corresponds to a real-life organization with its existing hierarchical member relationships, or is dynamically created to govern the service agreements formed between two or more peer Orgs. Each agent maintains a set of *policies* and *configuration facts*. Configuration facts describe the structures in which an agent participates. They include information about their existing relationships such as commitments and contracts, organizational structure, power relationships, and so on. Section 3 describes policies at length.

Definition 1 An *Org* O is an agent defined by a tuple $\langle M, P, C \rangle$. Here $M = \{A_1, \ldots, A_n\}$ is a set of *members*; P and C are sets of formulas. The A_i are the *members* of O and P are the *policies* of O. C is a set of *configuration facts* that include relationships such as commitments.

Each agent monitors and records events, which correspond to environmental observations or the actions of this and other agents. Based on these events and their policies, agents modify the configuration and choose their actions.

Commitments. Commitments form the basis for service agreements. Commitments are directed obligations from a *debtor* agent to a *creditor* agent, but arising within the scope of a *context* Org. A *commitment* is defined as $\mathsf{C}(A, B, F, U, Id)$, where A is the *debtor*, B the *creditor*, U the *context*, F the *discharge condition*, and *Id* the identifier. The context Org provides the means to handle any exceptions by revoking or otherwise manipulating the commitment.

We study the following commitment operations (based partly on [8]). Commitments are *created* in the context of an Org. A debtor *discharges* the commitment when the discharge condition holds. A debtor may *cancel* a commitment, or may *delegate* it to another agent. A creditor may *assign* it to another agent. A context Org or a creditor may *release* a commitment, and a debtor or a creditor may *escalate* a commitment to a context Org, indicating any additional requests or complaints.

2.1 An Example Scenario

Consider a (fictitious, but realistic) scenario where the Department of Energy (DOE) and the Department of Commerce (DOC) coexist within an Org Org-DOC-DOE, which is created as a context for the service agreements between DOC and DOE. Figure 1 illustrates this scenario. The dashed edges indicate the hierarchical organizational structure. DOC contains NOAA, which contains the National Hurricane Center (NHC). Likewise, DOE contains National Labs (NL), which contains Argonne (ANL) and Oak Ridge (ORNL). Here the leaf nodes, e.g., NHC and ANL, are individual agents; the other nodes are Orgs. Enactment can occur at multiple level in the hierarchical structure. Hence, other service agreements or relationships can exist between members resulting in the formation of new Orgs. For example, Org-NOAA-NL is created as a context for NOAA and National Labs, and Org-NHC-ANL for NHC and ANL.

ANL, a descendant of DOE, provides grid computing facilities to the NHC, which performs hurricane modeling. ANL is committed to NHC to providing the computing service. NHC commits to ANL to ensure accurate scheduling and supervision of the grid jobs, and also to make timely payments for the services obtained.

2.2 Challenge: Preemptive Scheduling

Previous research has motivated the need for grid applications to be proactive to handle crises, such as hurricanes, earthquakes, and so on [5]. A challenging use case here is

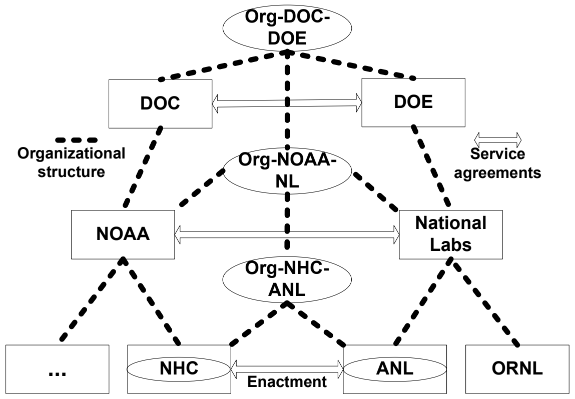


Figure 1. An example hierarchical scenario

that of preemptive scheduling of resources such as in the light of critical events. For example, let's revisit the example of Figure 1. Say, certain events analyzed by NHC indicate an emergency hurricane situation causing a surge in NHC's computational resource requirement beyond what is currently being offered by ANL. In this scenario, NHC makes a request for additional resources in the Org-NHC-ANL context to ANL. In case the request cannot be granted, ANL or NHC may escalate it to Org-NHC-ANL. If NHC escalates, Org-NHC-ANL will check the request before deciding its action. If the request is legitimate and if ANL cannot grant it, then Org-NHC-ANL can inform National Labs of the situation by escalating it further to Org-NOAA-NL. In well-designed organizations, exceptional situations can be handled locally. Thus forwarding of escalations is rare, but can propagate up to the topmost Org. The higher Orgs enforce their policies preemptively. The higher Orgs may preempt some service agreements in favor of others. Here, Org-NOAA-NL would request National Labs to provide the additional resources to NHC. National Labs may cause ANL to give up its other service engagement to support NHC fully instead.

This might appear to be quite easy. Yet, preemptive scheduling features as a challenge problem for production grid applications because current architectures and policy models do not support acquiring the right requirements or modeling and placing the desired policies correctly to produce desired behaviors without hardcoding or human intervention. This paper develops the necessary architecture that would enable such proactive policy-based governance.

3 Policy-Based Governance

The multiagent definition of Orgs supports a policy-based governance architecture. Each agent (individual or Org) that has nonnull policies instantiates a policy engine that controls its communicative and other actions, including those in response to environmental events and the communicative actions of others.

Traditional policy architectures are generally reactive, and act on a per-request basis. For example, XACML, the extensible access control markup language policy framework [7], has an access decision language used to represent a runtime request for a resource. First, a policy associated with a resource is located. Next, the attributes of the request are compared with the rules, ultimately yielding a permit or deny decision. Policy decisions are handled using an architecture that consists of the policy enforcement point (PEP) and the policy decision point (PDP).

The proposed approach goes beyond traditional architectures by emphasizing proactive monitoring and compliance.

3.1 Policy Architecture Requirements

In a dynamic organization, the object of a policy need not be a passive entity but could be an agent—usually one to whom the organization might have delegated some responsibility and granted some authority and visibility for the purposes of a specified family of interactions. A desirable policy architecture should be flexible enough to capture dynamic relationships between entities, and the context in which the entities exist. Policies should kick in automatically leading to the creation of new Orgs, or changing membership in existing Orgs.

The present approach goes beyond XACML in introducing architectural components geared toward monitoring and compliance. It borrows, from traditional grid policy architectures [4], the notions of monitoring and aggregating distributed events, and hierarchical enforcement of policies. However, it extends these notions via (1) event monitoring and propagation, (2) compliance checking aided by monitoring, and (3) policy modeling and enactment in Orgs.

3.2 Agent Policy Architecture

Figure 2 describes our agent architecture. An agent contains a PEP and a PDP as in traditional policy architectures. But the PEP and the PDP provide additional features with the aid of two new components: *Policy Monitoring Point* (PMP), and *Policy Organizational Point* (POP). These are the main modules that constitute our "M-O-D-E" (PMP: **M**, POP: **O**, PDP: **D**, PEP: **E**) architecture. An agent connects to an event bus [2] to send and receive events. The PMP helps to make this agent policy architecture event-based, yet proactive. Whereas a traditional PEP would merely allow or disallow a requested event based on the PDP, here, the PEP can perform an action different from any event that might have been requested. In particular, a domain action may be taken even if there is no explicit external event specifically triggering that action; the triggering condition might be in-

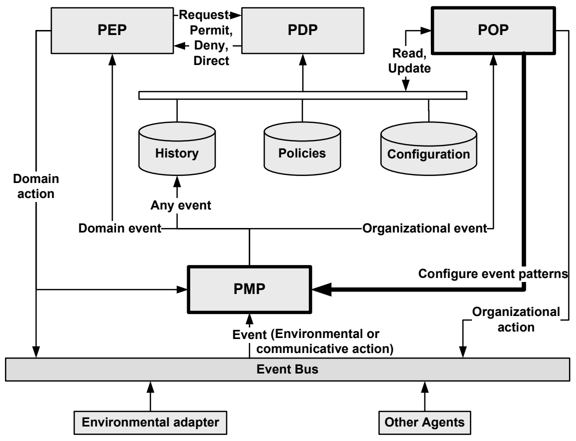


Figure 2. Architecture of policy-driven agent

ternal to the agent, such as based on its prior commitments, events stored in the history, or merely the passage of time.

POP. The POP manages "organizational" relationships, which are reflected mainly in the configuration store and the policies. Thus the POP potentially changes the behavior of the PDP by modifying the policies that the PDP applies. In essence, the POP "reconfigures" an Org by performing operations on the commitments that apply to the parties involved, or requesting such operations from other agents, such as a higher-level Org.

PMP. The PMP is crucial in enabling proactive event-driven behavior. It observes or monitors an event stream that can include *communicative actions* (described in Section 3.4) and environmental events. PMP captures specified event patterns from the stream(s), as instructed by the POP. Some communicative actions are *organizational* events that add or manipulate the contents of the various stores of the agent. Other events are *domain* events that include requests and responses. The PMP dispatches the organizational events to the POP, which may apply its policies and take the appropriate actions on the data stores. The PMP sends events to the PEP for further processing. All events are stored in the history.

The PEP's actions are relayed back to the event bus, and thus visible to its own PMP. If an action performed by the PEP involves any changes in its stores (e.g., events such as the creation or manipulation of a commitment), the PMP sends the action as an organizational event to the POP, which may perform an appropriate action on the corresponding store.

3.3 Policy Enforcement

The hierarchical structure of an Org, coupled with the distributed nature of its members, has interesting consequences on policy enforcement. Specifically relevant policies on an Org can be pushed to its members, ensuring that a member not only complies with its local policies, but also with the policies of its ancestor Orgs. The policy architecture also supports events being propagated between an Org and its members.

Figure 3 represents a segment of a hierarchy with an Org and two of its children (To improve legibility the event bus is shown as a separate module (but connected) for the top Org and its two children). The event flows indicate that agents can send events to their parents, siblings, or children.

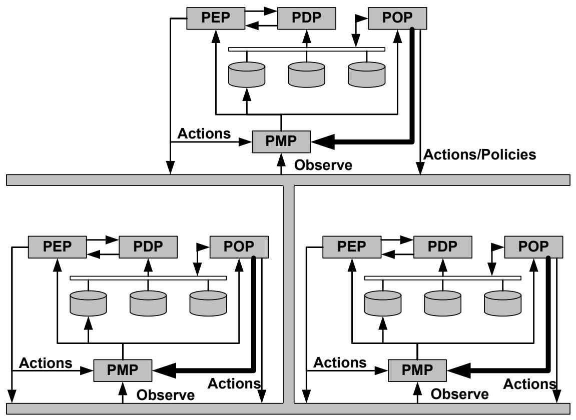


Figure 3. Hierarchical enforcement schematically

Policy Types. We define three kinds of policies: *local* (P_L), *hierarchical* (P_H), and *forwarded* (P_F). The *local policies* of agent A ($P_L(A)$) are policies that are set autonomously by the agent. The *forwarded policies* of agent A ($P_F(A)$) are the agent's policies that it propagates to its descendants. The *hierarchical policies* of agent A ($P_H(A)$) are the forwarded policies derived from A's ancestors.

For example, in our DOE–DOC scenario, DOE may have a local policy to offer the "best" graded QoS when dealing with DOC, or with DOC's descendants. DOC may have a local policy to contact only DOE, or DOE's descendants for any high-performance requirements. The local policies of DOE and DOC mentioned above are also their corresponding forwarded policies, and hence are propagated down to their respective descendants, including ANL and NHC. The hierarchical policies of NHC and ANL include the forwarded policies of their ancestors NOAA and DOC (for NHC), and National Labs and DOE (for ANL).

A parent Org propagates its *forwarded policies* to its children via communicative actions. Here, we note that $P_F(A) \subseteq P_L(A) \cup P_H(A)$. The PMP of a child receiving the policies may trigger an organizational event that is sent to the POP. The child's POP would (if the source is confirmed as a genuine ancestor) add the new policies to its policy store. The hierarchical policies of an agent override its local policies, in order to allow the higher-level organizations to enforce their policies on their descendants.

Definition 2 A policy p_i overrides a policy p_j ($p_i \succ p_j$), if an agent considers p_i over p_j to make a policy decision.

Property 1 $\forall p_H \in P_H(A), p_L \in P_L(A), p_H \succ p_L$: The hierarchical policies $p_H \in P_H(A)$ of agent A override its local policies $p_L \in P_L(A)$.

In the DOE–DOC scenario, consider a situation when ANL has a local policy that allows offering only lower grade QoS during a particular time period. However, one of the DOE policies overrides this local policy ensuring the "best" QoS being offered to NHC at all times.

3.4 Policy-Based Control of Agent Actions

Enacting cross-organizational service agreements involves agents performing several actions to manage their commitments and their configuration. The policies of an agent control its actions by specifying rules that force the agent to take certain actions or respond to the actions of other agents. We consider two kinds of actions.

Domain Actions are certain the "functional" or application operations performed by the agent. Examples include running a simulation, allocating a light path in an optical network, a grid service provisioning, and so on. Domain actions may enable the manipulations of the configuration facts. For example, discharging a commitment may involve performing one or more domain actions.

Communicative Actions. In an organizational setting, commitments formed among agents may be manipulated. A communicative action may specify administrative operations on such configuration facts. The execution of a communicative action might result in one or more agent domain actions. For example, the creation of a commitment may result in multiple domain actions to discharge the commitment.

Definition 3 A *communicative action* is expressed as $\mathit{operationName}(A, B, \Theta)$, where A is its sender, B its recipient, and Θ its subject.

The relevant operations are domain actions, commitment operations, and administrative operations such as *request*, *accept*, *deny* and so on. For the commitment operations, Θ would be the commitment on which the operation performed, or may be the policy being propagated. For example, a commitment C_1 can be created by A_1 and communicated to A_2 using the communicative action of $\mathit{create}(A_1, A_2, \mathsf{C}_1)$.

4 Evaluation

Our policy architecture emphasizes the proactive aspects of policy management. Monitoring events is crucial for proactive policy-based governance. The Org architecture addresses challenges such as preemptive scheduling. This section also motivates a formal analysis of how agents' actions comply with policy. Hierarchical policy enforcement may look simple, but is expressive because we generate hierarchical Orgs flexibly and with potential overlaps. Our policy architecture handles hierarchical relationships within an Org such as a real-life organization, as well as peer to peer relationships resulting from cross-organizational service agreements.

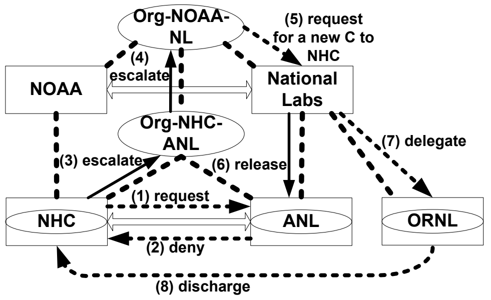
4.1 Challenges: Preemptive Scheduling

Figure 4 (left) revisits the scenario of Section 2.2. When an ancestor Org receives an escalate of a commitment from a descendant, the ancestor Org can check if it is legitimate, and perform any compensatory action. If it is unable to satisfy the escalate, it can forward the escalate up in the hierarchy. In the present case, National Labs preempts ORNL from any of its other service engagements (not shown in the figure), and delegates the NHC commitment to ORNL.

Figure 4 describes how our agent policy architecture handles this scenario. It shows a messaging sequence diagram indicating the communicative actions and their processing by the different components of the architecture.

Handling Conflicts and Escalations An agent can potentially form multiple commitments with different agents. An agent's commitments may potentially conflict because of timing or resource constraints. For example, Figure 5 shows a scenario where ANL is simultaneously committed to both NHC and the data mining department (DM) of Visa. ANL denies NHC's request for additional resources because of the two conflicting commitments.

Our policy architecture resolves such conflicts with the help of escalations and the dynamically created Org structure. An agent involved in multiple simultaneous commitments may assign *preemptive priorities* to selected commitments. In the above example, the conflict between the two commitments created by ANL can be detected via a logic engine. ANL might have a policy to handle such conflicts by canceling one of its commitments on the basis of the specified priorities. More importantly, such priorities would be specified by a higher-level Org when the service agreements are initially made. Priority assignments may depend on the order in which the agreements are created. For example, if NOAA-NL agreement is created first, NOAA may receive the *highest* priority for hurricane and earthquake modeling, while Visa may receive the *best possible* priority for data mining.



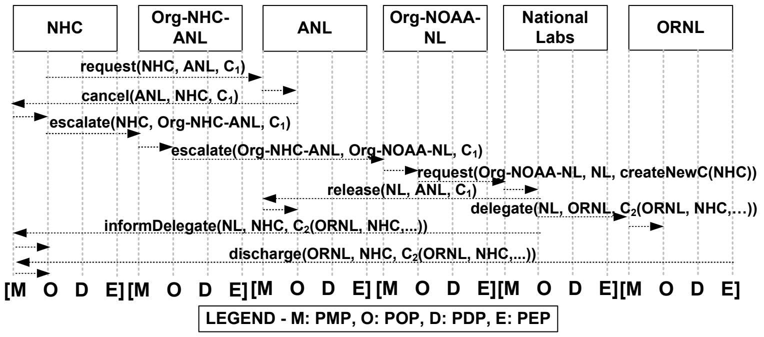

NHC
Org-NHC-ANL
ANL
Org-NOAA-NL
National Labs
ORNL
request(NHC, ANL, C1)
cancel(ANL, NHC, C1)
escalate(NHC, Org-NHC-ANL, C1)
escalate(Org-NHC-ANL, Org-NOAA-NL, C1)
request(Org-NOAA-NL, NL, createNewC(NHC))
release(NL, ANL, C1)
delegate(NL, ORNL, C2(ORNL, NHC,...))
informDelegate(NL, NHC, C2(ORNL, NHC,...))
discharge(ORNL, NHC, C2(ORNL, NHC,...))
[M O D E][M O D E][M O D E][M O D E][M O D E][M O D E]
LEGEND - M: PMP, O: POP, D: PDP, E: PEP

Figure 4. A messaging sequence diagram for the preemptive scheduling scenario

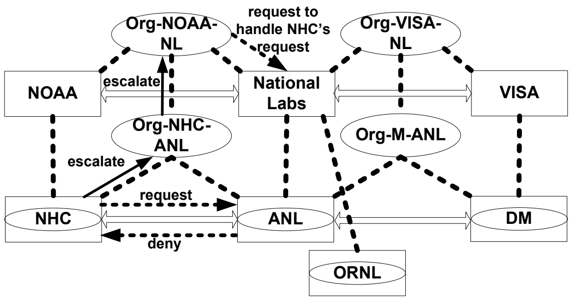


Figure 5. A conflicting commitments scenario

In the above example, depending on the priorities specified for the service agreements (NL-NOAA and NL-VISA), when NL receives a request to handle an escalate from one of its higher Orgs (Org-NOAA-NL or Org-VISA-NL), NL can either release ANL from NHC or DM and delegate to ORNL, or can release ANL from DM, and put it on hold.

Privacy constraints sometimes limit the handling of escalates directly by the lower Orgs. For example, Org-ANL-NHC cannot resolve NHC's escalate, because ANL may not share any information about its Visa contract while denying NHC's request. Hence escalates may be forwarded up the hierarchy until one of the Orgs can resolve it.

Org Creation Path versus Escalation Path. We observe that the path of the escalations reverses the path in which the Orgs are created. However, the creation path could be longer than the escalation path. As a case in point, once a commitment is assigned, the assigner may not want to hear if it does not work out. For example, when a bank B_1 sells a loan to B_2, and B_2 cannot collect on it, then B_2 has to deal with the borrower on its own, not through B_1.

4.2 Compliance and Completion

The compliance and completion of an action is determined based on the policies in effect and the events observed. Below P is a policy, a is an action, and $\models$ means entailment.

Definition 4 $P \models a$ means that logical rules P require action a given the facts. $P, a \not\models \mathsf{false}$ means that action a is consistent with (permissible based on) the set of rules P given the facts. $P, a \models \mathsf{false}$ means that action a is prohibited by the set of rules P, given the facts.

An action $a(A)$ performed by agent A is policy compliant if $a(A)$ *complies* with the policies, events, and facts stored by A ($\langle P_H, P_L, E_h, F_c \rangle$). F_c is a set of configuration facts, and E_h is the set of events in the history. Below F_C and E_h are fixed and are not included in the definitions and postulates.

Postulate 1 states that an action must be done if it is required by the hierarchical policies of an agent. Agents in Orgs are not completely autonomous. Hierarchical policies have a higher precedence than local policies. Using Property 1, we see that (when P_H requires action a), even if action a is not permissible with the local policies of the agent, a must be done, because the hierarchical policies override the local policies.

Postulate 1 If $P_H \models a$, then A must do a

Postulate 2 states that an action must not be performed if it is not permissible by the hierarchical policies.

Postulate 2 If $P_H, a \models \mathsf{false}$ (P_H prohibits a), then A must not do a

Postulate 3 states that when P_H permits a, a must be performed if required by the local policies.

Postulate 3 If $P_H, a \not\models \mathsf{false}$ (P_H permits a) and if $P_L \models a$, then A must do a

Postulate 4 states that action a must not be performed when not permissible by the local policies, and not required by P_H.

Postulate 4 If $P_H \not\models a$ (P_H does not require a) and if $P_L, a \models \textsf{false}$ (P_L prohibits a), then A must not do a

Postulate 5 states that when P_H and P_L both permit a, but not require a, then a may be performed.

Postulate 5 If P_H and P_L do not require a and if $P_H, P_L, a \not\models \textsf{false}$ (P_H and P_L both permit a) A may do a

The above postulates group the hierarchical policies of an agent which is a collection of the forwarded policies of its ancestors ($P_H = P_F(A_{\text{root}}) \cup \ldots \cup P_F(A_{\text{parent}})$). In P_H, the forwarded policies of the root agent A_{root} override those of its child and so on.

Definition 5 An action a of an agent A is said to be *policy compliant* if it is not prohibited by A's policies when it occurs.

Action Path. An action path in an Org is a sequence of operations that occur within its scope, i.e., across its descendants, in enacting a service agreement. The configuration administered by the Org agent evolves due to these operations, as the relationships among the administered parties change. An action path for a commitment is a sequence of operation beginning from its *create* and ending with its *discharge*, *cancel*, or *release*. Action paths for other kinds of configuration facts are similar.

Definition 6 A *commitment action path CAP* is a finite sequence of tuples $\langle a_i, A_i, C_i \rangle$ (i ranges from 0 to n), each representing a manipulation of the commitment C_i. Here a_0 is the *create* of C_0 by agent A_0, C_{i+1} results from action a_i applied on C_i, and a_n is either discharge, cancel, or release.

For example, in the NHC–ANL scenario, if ANL discharges the commitment by itself, then the action path is ⟨create(ANL, NHC, C), ANL, C⟩ and ⟨discharge(ANL, NHC, C), ANL, C⟩.

Definition 7 An action path is *compliant* if each of its actions is policy compliant when it occurs.

The following theorem establishes that all action paths in the proposed architecture are compliant.

Theorem 1 An action path formed in an Org is compliant.

Proof: In an Org, the hierarchical policies are aggregated with the local policies of an agent. We know from Postulates 1 and 3 that any action required by the hierarchical policies of an agent, or required by the local policies of an agent and permissible by the hierarchical policies, must be performed by the agent. At other times, when P_H or P_L prohibit an action, then that action must not be performed (Postulates 2 and 4). An agent may perform an action otherwise when both P_H and P_L permit it (Postulate 5). Hence all actions taken by the agents in an Org are permissible both by their local and hierarchical policies and are policy compliant by Definition 5. Thus the theorem follows from Definition 7.

Definition 8 An Org is *complete for a configuration fact* when an action path for that fact exists within that Org. An Org is *complete* if it is complete for all relevant configuration facts.

Potentially, an Org could be incomplete if no compliant actions were to exist under some circumstances. However, our definition of compliant action paths includes those that end in discharge, cancel, or release. By thus expanding the possibilities for the final action of an action path, we create more opportunities for an action path to exist. Further, our architecture prioritizes upper-level policies over lower-level policies. But an architecture cannot by itself ensure completeness.

For example, it is possible for the descendants of an Org to interfere with each other. In general, it is possible for two descendants of an Org (neither of which is an ancestor of the other) to have mutually conflicting policies. Barring such interference, an Org can complete for a configuration fact although the configuration may be violated externally. Theorem 2 captures this intuition.

Definition 9 Two actions are *independent* if either can occur and if both occur, they can occur in either order with the same resulting state.

Theorem 2 An Org is complete if all pairs of actions of all pairs of its descendants (neither of which is an ancestor of the other) are independent.

The above notion of completeness does not entail a "happy" ending. For example, NL may complete the commitments it delegates internally by releasing all of them, although that might cause NL to cancel its commitment to NOAA. Ultimately, completeness resulting in all commitments being discharged depends upon the various parties having the right policies for the circumstances in question. In particular, the cancel or release of a commitment may cause the creation of other commitments. The design of such policies is an important topic beyond the scope of this paper.

5 Discussion

The foregoing presented a multiagent policy architecture to govern cross-organizational service agreements. The key differentiating features of this approach can be summarized as follows:

Proactive Policy Modeling. It enables us to go beyond the traditional emphasis on reactive policies. A production grid not only must react to explicit requests (reactive) but also must monitor its environment, collate events, and determine how to act (proactive).

Distributed Policy-Based Governance. It recognizes that Orgs are distributed. It supports two complementary perspectives. One is that there is a single locus of policy enforcement. The other is that a distributed Org must have parts that collaborate to enforce a given policy, which is achieved by hierarchical policy aggregation going downwards and escalates going upwards.

5.1 Implemented Prototype

We have implemented a prototype based on the proposed architecture using a policy engine based on Jess and conventional messaging middleware. This prototype demonstrates the policy-based enactment of commitments including the scenarios discussed above. A simulation has been set up with multiple agents each with its own rule engine communicating with others via messaging middleware.

Future work. This paper opens up important future directions in the field of Services computing. We consider handling specific cases of conflicts among service agreements, commitments, and other social relationships among agents as future work. As an important real-world scenario, we are also working on a case study in the domain of resource sharing in multiorganizational IT infrastructures, focusing on a governance model for sharing of IT services. We are developing design patterns and templates for specifying Orgs and their policies. An important future exercise is to identify strategies for conflict resolution and policy enforcement.

5.2 Related Work

Compared to our previous work [9], we have simplified the model and developed the "M-O-D-E" architecture, accommodating hierarchies more precisely.

Policy Languages We consider two such policy languages. Rei [6] supports constructs such as rights, prohibitions, obligations, and so on. The architecture underlying Rei has a resource manager that functions like a PEP. It consults the Rei policy engine (functioning like a PDP), which produces a certificate specifying permissions, their validity period, and such. By contrast, our approach considers organizational architecture explicitly and could be realized using Rei.

Ponder [3] supports obligation policies as event triggered condition-action rules. Our approach offers a more general proactive architecture, and a high level vocabulary based on commitments.

Policy Architectures Grid services research has considered policy architectures. Grid policies focus on resource usage, access control, membership, and resource management. Dumitrescu *et al.* propose scientific data grids based on usage policies [4]. The usage policy enforcement happens both at the VO level (for grid-wide policies) (VO-level V-PEP), and at the site level (S-PEP for site policies). V-PEPs interact with S-PEPs and schedulers to enforce VO-level policies. Each PEP is supported by a monitoring distribution point (MDP), which gathers information about resource usage and policy restrictions. MDPs are distributed but can interact with other MDPs. Our approach handles general cross-organizational interactions and can support grid-like VO architectures.

Normative Systems Boella *et al.* propose a conceptual model of virtual organizations as normative multiagent systems [1]. They demonstrate the dynamic aspects of organizations using different types of interactions between the normative systems and the agents playing specific roles. Our architecture goes beyond the above in providing an expressive organizational structure, and policy-based governance.

References

[1] G. Boella, J. Hulstijn, and L. van der Torre. Virtual organizations as normative multiagent systems. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 192–201, 2005.

[2] D. E. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[3] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of International IEEE Workshop of Policies for Distributed Systems and Networks (POLICY)*, pages 18–38, 2001.

[4] C. L. Dumitrescu, M. Wilde, and I. Foster. A model for usage policy-based resource allocation in grids. In *Proceedings of 6th International IEEE Workshop of Policies for Distributed Systems and Networks (POLICY)*, pages 191–200, 2005.

[5] N. Group. NGG2 Expert Group Report: Requirements and Options for European Grids Research 2005–2010 and Beyond. July 2004. Available at URL: http://www.semanticgrid.org/docs/ngg2_eg_final.pdf.

[6] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *Proceedings of 4th International IEEE Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 63–74, June 2003.

[7] OASIS. eXtensible access control markup language (XACML) version 2.0 specification document. *OASIS Standard*, Feb. 2005.

[8] M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

[9] Y. B. Udupi and M. P. Singh. Multiagent policy architecture for virtual business organizations. In *Proceedings of the IEEE International Conference on Services Computing (SCC)*, pages 44–51, Sept. 2006.