Licit: Administering Usage Licenses in Federated Environments

Prashant C. Kediyal and Munindar P. Singh, *Fellow, IEEE*

Abstract—We address the problem of usage license administration in federated settings. This problem arises whenever organizations, such as educational or research groups or institutions, share resources for business and scientific reasons. In such settings, each user's usage of a licensed resource is typically supported by the user's organization. License administration involves satisfying legal requirements while applying organizational strategies for effective resource usage, and carrying out suitable accounting and audit controls.

We propose an approach, Licit, wherein an agent represents each resource sharing site and administers licenses in collaboration with other agents. We show how to represent a variety of usage licenses formally as executable policies and provide a simple information model using which each party can specify both the attributes involved in its licenses and how to resolve them. Our architecture naturally accommodates a variety of site-specific (i.e., custom) strategies for license administration. Licit has been implemented in a popular open source framework for virtual computing, and yields performance results indicating its practical feasibility.

Index Terms—IT administration, grid computing, cloud computing, usage licenses, XACML

1 Introduction

Computing infrastructure is expensive to acquire, maintain, support, and administer. For this reason, recent advances in distributed computing under the rubrics of grid computing and cloud computing [1] promise significant gains in resource usage efficiency and staff productivity. Scientific and educational computing are two important settings where the above-mentioned advances can apply [2]. For example, research groups in different institutions can contribute their otherwise unused resources to a common pool with the understanding that they would benefit from the common pool when they need additional resources. In higher-education settings, university departments and labs are able similarly to share their resources. In K-12 education, school districts are finding that maintaining separate computing infrastructures individually for each school is prohibitively expensive. In all these cases, the goal is to improve resource usage through federation.

Our motivation becomes more apparent in connection with emerging kinds of scientific workflows. Traditionally, workflows have been thought of as limited to individual organizations. However, recent research has begun to show a shift toward federated settings. For example, Blake and Huhns [3] motivate web-scale workflows built on top of distributed services. Likewise, Lu and Zhang [4] and Zhang et al. [5] motivate scientific workflows that not only involve multiple collaborators but also cut across organizational boundaries.

To realize such cross-organizational workflows presumes that resources can be shared effectively. However, existing techniques have no support for ensuring that permissions, especially, resource usage licenses, propagate correctly. Armbrust et al. [6] identify software licenses as a key challenge of realizing cloud computing. We are pursuing a research program of developing approaches for the policy-based administration of resources in federated environments. Specifically, we address the challenge of accounting for and administering software *usage* (and not consider software development or digital media sharing) licenses in federated environments.

To develop the above ideas further, consider a simple, but real, example of a federated computing environment. Some North Carolina institutions—including NC State University (NCSU: our institution) and NC Central University (NCCU)—federate some of their computing resources through the Virtual Computing Lab (VCL), an open source cloud infrastructure [7], [2], [8]. Their motivation is that they need to provide their respective users (students and faculty) virtual access to hosted applications such as Matlab, and federating resources enables handling peak loads better. The challenge here is that each institution's site license covers usage by only its own faculty and students. Yet the applicable license must be accounted for correctly even when requests from one institution users are instantiated at VCL installations across the federation. One might imagine that scenarios such as the above could be avoided if each institution simply obtained computing hardware from the other and handled its licenses independently of the other. However, to accomplish this, the site that is currently hosting a user from another site must be able to (1) confirm that such a (suitably authenticated) guest user is legitimate and that any requirements on the usage session are met, and (2) be able to preempt the guest when necessary. That is, we need a way by which federated parties may collaborate effectively while preserving their autonomy.

• *The authors are with the Department of Computer Science, NC State University, Raleigh, NC 27695. E-mail: {pckediya,singh}@ncsu.edu*

A variation of the above scenario arises where a university or a company obtains a site license but departments within the university or company share the cost and benefits of the license, and administer their own computing resources. Such cases are routine practice today. However, statically allocating licenses among the departments would clearly not be effective, and would obviate most benefits of obtaining a site license.

The common problem is that wherever parties federate, we need a way to account for usage licenses appropriately. Doing so is nontrivial because of the following challenges. First, in practice, usage licenses might involve considerations such as the number of concurrent users allowed. Thus the parties must communicate the relevant information or convey appropriate decisions to one another. Second, each site may apply potentially subtle strategies to administering its licenses, even when its users are engaging through one of its collaborators. For example, a concurrent users license merely states the upper bound on the number of concurrent user seats; how to prioritize requests or allocate such seats is entirely up to the licensee. A practically viable approach should support such flexibility. Third, each host must be able to retain control over its resources and consequently be able to express site policies regarding how it collaborates with others. For example, NCSU may decide to share resources with NCCU all day but only after-hours with another institution in the federation.

1.1 Approach in Brief

We propose *Licit* , an approach that addresses the foregoing challenges via license, scheduling, and site policies, respectively. A *license policy* states the terms between the licenser and licensee, for example, a license to use 100 concurrent instances of Matlab. A *scheduling policy* states how the licensee chooses to opportunistically exploit the license, for example, by preempting members of one role in favor of another. Specifically, the licensee might prefer teachers over students so that the oldest (or newest) instance in use by a student is terminated if necessary to fulfill a request made by a teacher. A *site policy* protects the interest of a particular provider by capturing its preferences when it makes its resources available in a federation. For example, NCSU may share its resources with NCCU but limit access by NCCU students during the NCSU exam week—so as to give a higher priority to NCSU students during that time.

Licit applies policy-based agents—computationally representing each site—to collaboratively administer usage licenses. Licit involves agents who play two important roles in an ongoing engagement: *licensee*, who currently owns the license on the desired resource and with whom the current user is affiliated, and *host*, who provides the computational infrastructure over which the user (as a guest) would interact with the resource.

Agents implemented over services are a natural match [9] for the present problem for the following reasons. Each site in a federation is (at least notionally) autonomous and heterogeneous in that it may administer its licenses according to its local strategies and based on its local information model. Potentially, the sites may adopt distinct strategies and informations model. Further, to enforce certain usage constraints based on history and state [10] or potentially in anticipation of demand, each party would need to monitor events and behave proactively—and thus each of the parties must be able to entertain interrupts from others.

We adopt a policy-based approach as the underlying framework for the specification of agents and their interactions. Policies yield greater perspicuity and modularity than procedural representations and have been used in a variety of settings, most commonly access control [11]. We do not propose a new policy language or engine. The Licit approach is implemented over existing technologies. We show how to represent licenses, license administration strategies, and resource allocation considerations as policies. Licit agents can exchange policies with each other. For each agent, we organize the various policies (local as well as those received from peers and express their constraints) in a simple hierarchical structure that yields the desired computations in a natural manner.

1.2 Contributions

We develop a policy-based approach for administering federated computing environments, such as those enabled through the expansion of cloud computing technologies. The novelty of the Licit approach lies in its treatment of policies in a federated setting. Licit is framed in the context of usage licenses for software applications, and can be seen as demonstrating service computing techniques to support further services. However, the basic ideas of both licensing and of Licit apply to resources in general—including databases, data streams, sensors, and so on—that could potentially be subject to a usage license.

1.3 Organization

The rest of this paper is organized as follows. Section 2 formalizes the problem domain by laying out various license types, scheduling strategies, the information model, and the states in the life cycles of a license and of a usage session. Section 3 provides a technical view of the system in detail, including its architecture, the representation of a license as a policy, concepts of a scheduling policy, and the organization of policies at a site. Section 4 presents an evaluation of our implementation's performance demonstrating its feasibility in practical settings. Section 5 discusses the relevant literature and some future directions.

2 Formalizing Usage Licenses

We describe the structure and elements of a usage license that formalizes a license that a licensee site would

have obtained for the benefit of its user community. We propose a simple information model using which a site can specify a vocabulary needed for *evaluating* its usage licenses, i.e., determining whether a particular user request is valid according to a specific license. We begin with a classification of the license types gathered from interviews with academic IT experts and which arise commonly in modern practice. From this classification, we determine a base vocabulary that can be used to state licenses. Table 1 captures the commonly occurring usage license types. These examples illustrate a nearly comprehensive list of licenses encountered in practice, and hint at the type of licenses supported by Licit.

TABLE 1
Examples of commonly occurring usage licenses.

	This license covers usage ...
⊙	by faculty
⊙	by anyone from Engineering
⊙	by a person physically present within a certain geographic area defined by zip code or by distance from the offices of the licensing institution
⊙	for research; or for a noncommercial purpose; or in an educational activity (thus excluding business purposes of the university)
⊙	if permitted by a specified licensing server
⊙	for one year
⊙	of Matlab
⊙	for a maximum of 100 hours
⊙	by up to 100 students or faculty at any time
⊙	for a maximum of 100 times
⊙	on a specific machine
⊙	on a machine located within a certain geographic area
⊙	on a Linux platform
⊙	on a machine owned by a specific owner
⊙	on a machine with a maximum of four processors

2.1 An Information Model for License Attributes

Figure 1 presents an information model for the attributes that may feature in licenses. The policy representation of a license specifies the predicates that operate on these attributes. Each site would use this information model to specify its custom vocabulary of attributes. The main benefit of developing an information model is to overcome the challenge of heterogeneity, specifically, to enable a site in a federation to interpret the terms in a vocabulary that has been specified by another site.

A license vocabulary comprises *attributes*, each of exactly one data type. The attributes involved in a license describe the subject (prospective user), resource, action, environment, and sometimes the license identifier. An example of an attribute that describes the constraints on a license—identified by a license identifier—is *cumulative usage*. The environment attribute may indicate one or more of the following: start or end time, and the identifier for the node on which resource is to be deployed. An example of an environment type attribute, i.e., an attribute that describes the node, is *number of processors*. We introduce another entity type, *context*, to include attributes that characterize an association of more than

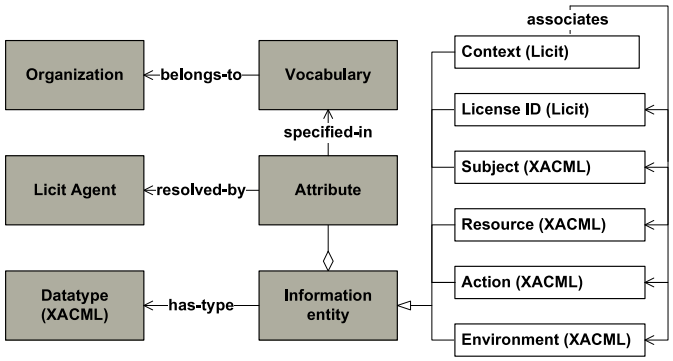


Fig. 1. An information model for license attributes.

one of the subject, resource, action, or environment. An example is the attribute *permitted usage*, which describes permission of a subject to perform an action on a resource in an environment.

In a significant break from current approaches to license administration [12], [13], [14], in our model, an attribute is resolved by an agent (i.e., host, user, or licensee) instead of a service location. This is unavoidable in a federated setting, since the location at which to resolve any attribute associated with the host, such as the *number of processors*, is known only at runtime. This is because the host and thereby the node at which the requested resource is deployed is also known only at runtime. The agent-based solution enables such late-binding of attributes to values because the licensee can simply ask the appropriate agent, in this case the host, to resolve attributes such as *number of processors*. To this end, the licensee would provide values of some attributes to determine the value of a requested attribute. For example, to determine the *number of processors* of a *node*, the licensee must provide the value for the identifier attribute of the node.

We can now express a base vocabulary that includes the attributes that feature in the license types of Table 1. Table 2 classifies these attributes based on the entity to which each applies. Read the first row of Table 2 as follows: The attribute *role* describes the subject and is resolved by the licensee; example values for *role* are "faculty" and "student." Read the subsequent rows following the same pattern.

Some attributes may be particular to the licensee and must be resolved by it. Some licenses may be based on information that can only be provided by the user, e.g., a user's *acceptance of the terms* regarding the purpose of use. We include these with the licensee.

Licit is not limited to the above attributes, and a site may introduce additional attributes as needed. Whether a site uses the above or its site-specific attributes, it needs to specify the agent each attribute is *resolved by* so that a licensee would contact that agent to request values for those attributes.

Let us consider how an attribute vocabulary may be used. Suppose Alice, a computer science student at

TABLE 2
Classification of attributes that feature in common licenses.

Attribute	Describes	Possible values	Resolved by
Role	Subject	faculty; student	Licensee
Department	Subject	computer science	Licensee
User's IP address	Subject's context	192.168.2.1	Licensee
User location	Subject's context	on campus; zip code	Licensee or User
Purpose	Subject's intention	research; educational	Licensee or User
Permitted usage	Request's context	true or false	Licensee
Period	License's context	for one year	Licensee
Concurrent usage	License's context	by up to 100 users	Licensee
Cumulative usage	License's context	up to 100 hours by a group	Licensee
Counted usage	License's context	five times	Licensee
Node name	Compute resource	on a specific machine	Host
Node location	Compute resource	zip code	Host
Node operating system	Compute resource	on Linux platforms	Host
Node ownership	Compute resource	on a departmental machine	Host
Node type	Compute resource	with no more than four processors	Host

NCCU (licensee), requests a usage session with Matlab at NCSU (host). The host finds that a node is available so it forwards the request to the licensee. The licensee finds that its license for Matlab involves checking the subject's attribute *department* and the *number of processors* of the node on which Matlab will be deployed. Therefore, the licensee requests the value of this attribute for Alice from its own (NCCU's) attribute service, which returns "computer science." Next, because the vocabulary states that this attribute is resolved by the host, the licensee requests the value of *number of processors* from the host (NCSU) instead of its own attribute service. Based on the above information, the licensee finds that it can provide access to Alice.

It is to be noted that the challenge of determining a value for an attribute rests with the attribute service. A site may introduce any apparently idiosyncratic attributes that make sense for its local needs. All it needs to do is ensure that the attribute service can resolve such attributes for the entity (e.g., licensee or host) involved. For example, a site may introduce an ad hoc attribute "social-influence" with values "high" and "low" and use this attribute within a policy to decide which users are to be accorded special treatment in accessing certain resources. The attribute service on its part may calculate a value for social influence by querying a social network website. This ad hoc example is meant to illustrate that Licit can work in a wide range of settings.

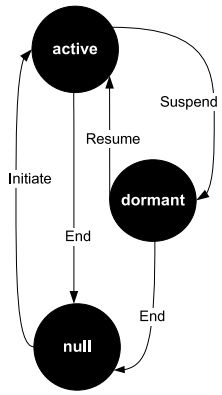


Fig. 2. A usage session involving a licensed resource.

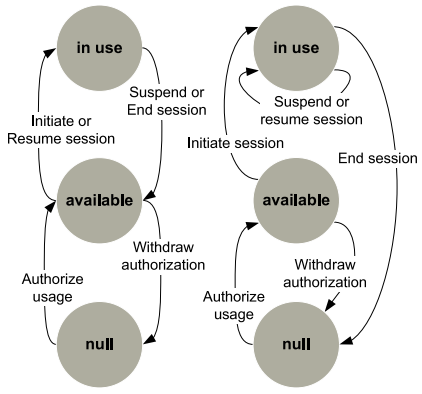


Fig. 3. Life cycles of a concurrent-usage (left) and a single-usage license (right).

2.2 License and Usage Session Life Cycles

We model each license as an object with a life cycle. Not surprisingly, different types of licenses exhibit different life cycles and constrain possible enactments in different ways. Additionally, we model a usage session as an object with a life cycle of its own.

In Figure 2, the usage session's life cycle starts when a guest user's request is accepted so that the provisioning of resources may begin. The usage session at this point becomes *Active*. A usage session may be suspended, thereby making it *Dormant*, and a suspended session resumed, thereby making it *Active* once again. The user can end the session from either *Dormant* or *Active* state. Note that in Figures 2 and 3 the license life cycle is interlinked with the usage session so that it is affected by all events that affect the usage session.

Let us now consider a *concurrent users* license. In Figure 3, *Null* means the license does not exist at the host. Once a license is provided to the host it becomes *Available*. Once a guest user's request is accepted so that

the provisioning of resources may begin the license is *In Use*. When the user suspends a usage session, the corresponding license becomes *Available* (for use). When the suspended session is resumed, the license is *In Use*. Finally, the license may be withdrawn.

The above life cycles are described in terms of state diagrams. In general, different license types support different state diagrams. For example, as Figure 3 shows, a *Single Usage* license remains *In Use* when the session is suspended and becomes *Null* when the associated usage session ends. A suspended session may be resumed, making the session *Active*

In Licit, the state of a license is maintained by the licensee even as the usage-sessions are instantiated in a distributed manner at the various hosts. Doing so enables us to support the use case where a host denies access to a user because of the host's local policies. In this case, the user can try elsewhere.

2.3 Scheduling Strategies

Often an organization must optimize its license usage according to its local needs. Usage licenses often provide a lot of flexibility in this regard. For example, a typical license would specify the maximum number of concurrent users. It would not specify which users, if any, are to be accorded priority over others. Suppose NCSU has obtained a license for 100 concurrent users. NCSU could arbitrarily reserve 10 of the licensed seats for faculty and only allow up to 90 student users. (Such a static allocation is not necessarily optimal but may well be sufficiently effective for the needs of a licensee: In any case, it is the licensee's prerogative to allocate license seats according to its local strategies.) Alternatively, suppose all 100 seats are taken. Now, if a request arrives from another prospective user, NCSU has many choices, including these:

- *Deny* the request.
- *Preempt* one of the current users to free up a license seat, using a variety of strategies to choose which of the current users would be preempted: For example,
  - The one who has used the resources the most.
  - The one who has the longest running session (for fairness).
  - The one who has the shortest running session (to waste the least amount of work).
  - The one with the lowest organizational rank.
- *Suspend* one of the current users to temporarily free up a license seat using a variety of strategies such as the above; when a license seat becomes available, resume one of the (potentially multiple) suspended users according to a suitable strategy.
- *Delay* the request, holding it in a queue, and serve the request when a seat frees up; apply any of numerous possible disciplines to maintain the queue.

The above examples seek to illustrate the rich variety of strategies that an organization may apply while it respects a usage license. In current practice, these strategies are hardwired or handled in an ad hoc manual manner, which is expensive and would not scale to large federations. In this context, Li et al. [15] point out the need for a scheduling mechanism that accounts for requests distributed beyond the local domain. We address the above need by formalizing a scheduling strategy for a license as a policy, thereby abstracting it out of the code and making it reusable. The licensee checks the applicable scheduling strategy in deciding upon a request.

When a new request arrives that results in a need for scheduling, we need to carry out a comparison of the request with all the other requests authorized by the same license. In a federated setting, however, sessions associated with various requests may be distributed over various hosts. In such a scenario, performing a federation-wide comparison or prioritization to determine the session associated with the lowest priority request (according to whatever is the stated scheduling strategy) requires the agents to cooperatively maintain an annotated index of distributed instances. Section 3.3 describes a simple scheme to address this challenge.

3 Technical Approach: Agents and Policies

The demands of the autonomy and heterogeneity of the sites involved in federated license administration drive us toward an architecture based on agents and policies. Each site is represented computationally by an agent who stores and applies the policies of the site. The agent of each site deals with local users as well as with a local runtime environment. The runtime environment at each site manages the resources in consideration under the control of the site's agent. Additionally, the agent at each site also plays the role of a licensee that provides license and scheduling decisions. The agents have no direct control over other agents, because each agent is autonomous, reflecting the real-world autonomy of the participants in a federation. However, the agents can communicate requests and decisions to each other.

Based on the foregoing, the following scenario describes how a typical episode of license administration might proceed.

- The licensee creates policies pertaining to the specific license of interest along with a vocabulary that describes the attributes referenced in the policies. The description includes the agent who would resolve each attribute along with other supporting information.
- Using hierarchical site policies, a licensee and host create a service agreement wherein the host would support the licensee's users for a particular resource (i.e., executable image). They also configure themselves with information on how to contact each other.
- A user requests permission to use a resource at the host site that the runtime environment forwards to the Licit agent.

- The host's agent applies its site policies to determine how to proceed. In particular, the host's agent may use a policy decision point (PDP), realized over a policy engine, to make the decision regarding whether to permit the initiation of the usage session. The PDP obtains the requisite information from an attribute service. If the host grants the request, it forwards the request to the user's licensee for a license check.
- The licensee, in a manner similar to that of the host, uses a PDP to perform a license check. Here the licensee may request its own attribute service or (indirectly) that of the host to resolve some attributes so as to reach a decision.
- If a conflicting demand exists and a scheduling policy applies, the licensee uses the scheduler that—as described in Section 2.3 and Section 3.3—determines how to prioritize between the current request and other running instances. If, according to the scheduling policy, some instance needs to be terminated then the licensee notifies the appropriate host's agent. The host's agent in turn notifies the runtime environment.
- If the decision to the current request is favorable and a scheduling policy exists that applies to it now, or may apply to it in the future, then, before replying, the licensee records information such as (1) the request identifier, (2) the host at which the request will be instantiated, and (3) the normalized value of the attribute specified in the ranking criteria—described in detail in Section 3.3. Thus, the licensee builds an active-requests log that serves as an annotated index to the distributed instances.
- If the host receives a favorable decision, it asks the runtime environment to make the requested resource available to the user.
- The runtime environment produces a cryptographic token, which the agent conveys to the user.
- The user interacts with the runtime environment directly using the above cryptographic token.

Figure 4 depicts the main elements of our proposed architecture. These modules are installed at each site to enable it to participate in federated license administration. The user (or, rather, the user's agent) requests initiating, terminating, and suspending usage sessions. The user agent may also decide to request another host if its preferred host fails. Additionally, the user may provide values for specific attributes if requested.

The attribute service abstracts over a variety of mechanisms for obtaining values for any attributes of interest needed to evaluate the request with respect to the license. Such attributes pertain to the user, to the user's context, to the host, to ongoing interactions relevant to usage licenses, to the states of relevant licenses, and even to legacy license managers such as FlexNet, which we revisit in Section 5.1.

The Licit agent is the cornerstone of our approach as it acts as the host or the licensee, depending on the context. The Licit agent comprises the attribute finder that uses the vocabulary to direct queries to the appropriate attribute service, the PDP that interprets licenses and site policies stated in XACML, and the scheduler that evaluates the scheduling policy.

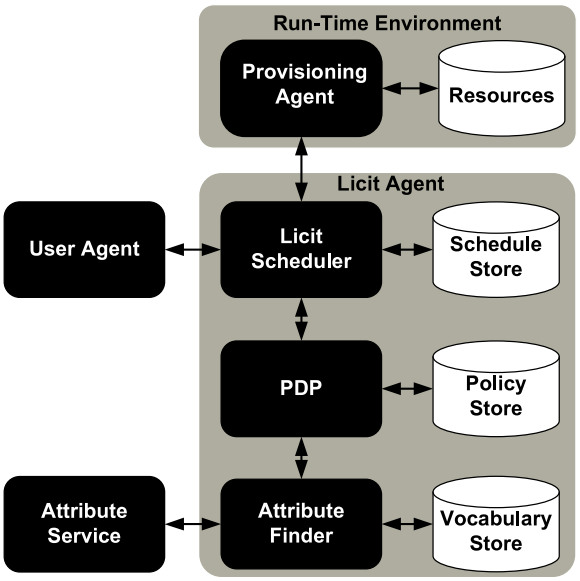


Fig. 4. An illustration of our architecture. Here the PDP refers to a policy decision point, which is the standard policy architecture component that applies policies to determine a decision as to allow or deny the given action.

3.1 Background on XACML and Scheduling Policies

Each XACML policy primarily considers four entities of interest: (1) *subject* (the entity that initiates the request); (2) *resource* (the entity to be used, such as instantiable software, as in our setting); (3) *action* (which the *subject* would perform on the *resource*); and (4) *environment* (which gathers other relevant information about the computational or user context, such as time and date). We adopt the eXtended Access Control Markup Language (XACML) [16], which has emerged as the leading standard for stating policies. XACML 2.0 (XACML) supports defining attributes and matching predicates and is sufficient for the scope of Licit. We did not find any new feature available in XACML 3.0 [17] necessary for our purposes. Therefore, to increase the generality of our approach, we assume only the weaker XACML 2.0 language.

3.2 Capturing Licenses as Policies

XACML provides the following elements: *policy sets*, *policies*, *rules*, and *conditions*. A policy set comprises other policy sets or policies; a policy comprises rules; and, a rule comprises conditions. In Licit, we use the XACML elements as follows. We group all the usage licenses for an organization in a single all-enclosing policy set. A policy set or policy elements in the enclosing set express

a license for a single *resource*. Although Licit does not limit the type of action, in our setting, the only relevant *action* that can be performed on the resource is "execute" because the resources here are installed software images. Constraints on the action are specified by rules that include the specific conditions under which the action is licensed.

The following snippets illustrate our XACML representation for licenses. The snippets when put together specify NCCU's license that states that any user from NCCU is permitted to execute MATLAB, but only on a node running "Windows 7" that has no more than two processors.

- As mentioned above, policy sets group all licenses for an organization. Hence, in the example below, the match criterion in the policy set element specifies that subject:org-id, a standard XACML attribute meaning the subject's organization, must equal "NCCU." Thus policies in this policy set apply to a request from anyone from NCCU.

```
<PolicySet PolicySetId="NCCU:LICENSES">
 <SubjectMatch
    MatchId="function:string-equal">
  <AttributeValue> NCCU </AttributeValue>
  <SubjectAttributeDesignator
     AttributeId="subject:org-id"/>
 </SubjectMatch>
</PolicySet>
```

- Enclosed in the policy set above, another policy set element specifies a license for a single resource. In the example for a MATLAB license listed below, the policy set specifies that resource:prettyname, an attribute declared in the licensee's vocabulary, must be tested for string equality with the name (here, "MATLAB") of the resource to which this license applies.

```
<PolicySet
   PolicySetId="NCCU:MATLAB:LICENSE"
 <ResourceMatch
    MatchId="function:string-equal">
  <AttributeValue>MATLAB</AttributeValue>
  <ResourceAttributeDesignator
     AttributeId= "resource:prettyname"/>
 </ResourceMatch>
</PolicySet>
```

- Enclosed in the policy set element from the previous listing, the policy element here corresponds to the license for a group of users to carry out an action. The group of users may be defined using the *SubjectMatch* element as shown in the first listing. However, since the example license is for all users from NCCU, we only need specify that the attribute action-id, a standard XACML attribute, must have the value "execute."

```
<Policy
 PolicyId="MATLAB:LICENSE:TO:EXECUTE">
 <ActionMatch
    MatchId="function:string-equal">
  <AttributeValue>execute</AttributeValue>
    <ActionAttributeDesignator
       AttributeId="action:action-id"/>
 </ActionMatch>
</Policy>
```

- Finally, the rule element specifies the *Condition* and *Effect* that determines whether to permit or deny a request if the stated conditions are met. In the Licit approach, the effect is always to permit because it is only when an action is permitted under some condition that it is included in the license. The condition element specifies a conjunction or disjunction of nestable Boolean functions that operates on attributes and values. The example below specifies a condition to check that the operating system (given by the attribute node:OS) equals "Windows 7" and the number of processors (given by the attribute node:processors) does not exceed two.

```
<Rule Effect="Permit" RuleId="CommitRule">
 <Condition FunctionId="function:and">
  <Apply
   FunctionId="integer-less-than-or-equal">
   <SubjectAttributeDesignator
    AttributeId="node:processors"/>
   <AttributeValue>2</AttributeValue>
  </Apply>
  <Apply
   FunctionId="function:string-equal">
   <SubjectAttributeDesignator
    AttributeId="node:OS"/>
   <AttributeValue>Windows 7
                  </AttributeValue>
  </Apply>
 </Condition>
</Rule>
```

- In special cases, for example, of licenses based on *concurrent users* that require accounting, the policy element above includes an obligation element. The obligation requires the licensee to count the license's usage when the license evaluation results in granting the permission.

```
<Obligations>
 <Obligation
    ObligationId="count_license"
    FulfillOn="Permit">
  <AttributeAssignment
    AttributeId="license_id"
    DataType="XMLSchema#string">TC-010
  </AttributeAssignment>
 </Obligation>
</Obligations>
```

- A request is processed as follows. The subject's organization determines which top-level policy set to use. The request is evaluated against the next level policy set that specifies the resource requested. Finally, the action requested and possibly other attributes of the subject determines the applicable policy where relevant conditions in the rule determine the effect. If the license contains an obligation, the licensee creates a checked-out record that corresponds with the *in-use* state in a license's life-cycle.

- Since XACML combines the outcomes of all the policies that apply, we need to consider the case that Alice may be able to use MATLAB either because she is enrolled as a student in Engineering or is employed as a staff member in Science. Thus, we disjoin multiple licenses by setting PolicyCombiningAlgId to "permit overrides," because Alice must not be denied access to MATLAB just because she does not have access to some other resource. Similarly, at the rule level, we always set RuleCombiningAlgId to "permit overrides," because we want a usage request to be granted if it is permitted under any rule.

```
<PolicySet PolicySetId="NCCU:MATLAB:LICENSE"
 PolicyCombiningAlgId=
 "policy-combining-algo:permit-overrides">

<Policy PolicyId="MATLAB:LICENSE:TO:EXECUTE"
 RuleCombiningAlgId=
 "rule-combining-algo:permit-overrides">
```

3.3 Expressing Strategies as Policies

A scheduling strategy identifies the license to which it applies. Given the currently active sessions, the strategy determines whether the incoming request should be rejected or a response to it delayed. Or, whether one of the active sessions should be preemptively suspended or ended after some time to free the license needed to accept the incoming request.

Since a XACML response can only be permit or deny, we need to go beyond XACML to treat strategies properly. All scheduling policies can be grouped together for an organization, as we showed above for the licenses. Further, each scheduling policy must include—what is known in XACML terms as—the target that includes exactly one LicenseID element that identifies the license to which the policy applies. However, beyond this, licenses and scheduling policies differ. As seen in the listing below, scheduling policies hold additional information. In particular, a scheduling policy contains information regarding the *ranking criteria* and the *preemptive action* to take after some *time* on the lowest priority request determined by the policy.

We capture priorities among requests through a simplified representation of the *ranking criteria* based on the values of a specified attribute, for which we provide a list of values, sorted from least to highest priority. For example, we may assert that priority is determined by the attribute *Role* using a list ⟨student, faculty⟩, which indicates that a request from a student is ranked lower than that from a faculty member.

```
<SchedulingPolicies>
 <OrganizationName>NCCU</OrganizationName>
 <Policy>
  <LicenseID>TC-010</LicenseID>
  <PreemptiveAction>
   <Action>End</Action>
   <Time>0</Time>
  </PreemptiveAction>
  <RankingCriteria>
   <Criterion>
    <AttributeId>role</AttributeId>
     <ListOfValues>
      <Value>student</Value>
      <Value>professor</Value>
     </ListOfValues>
     <SortOrder>Ascending</SortOrder>
    </Criterion>
   </RankingCriteria>
</Policy>
<SchedulingPolicies>
```

Notice that Licit separates licenses from strategies, although both licenses and strategies are represented computationally as policies. This separation facilitates authoring and reuse of licenses and strategies.

Scheduling proceeds as follows. Upon receiving a request, the licensee first checks the license to determine whether the request is legal with respect to any of the stated licenses. If so, the licensee checks whether a scheduling policy applies to the request. A rescheduling policy (specified using XACML and different from a scheduling policy) for a license is identical to the license in all respects except that the action is *reschedule* instead of *execute* and there is no check for constraints related to the attribute *concurrent users*. Thus, a rescheduling policy presupposes that the request was denied only due to constraints on *concurrent users* and that rescheduling could help. The licensee then looks at the scheduling policy and it determines whether and, if so, which of the current users to preempt.

As Sections 2.3 and 3 describe, requests may be distributed at hosts anywhere in the federation. Due to communication overhead and reliability concerns, it would generally be inadvisable for the licensee to attempt to prioritize among requests by coordinating with several hosts at the time when a decision on a new request is to be made. To avoid having to communicate with multiple parties, we provide an index sorting mechanism. The index is sorted using a normalized value calculated as follows: Before granting any request, the licensee always locally stores a normalized value of the ranking criterion for the request. For example, if the criterion is based on attribute *Role* and is specified by ⟨student, faculty⟩ and the request is from a student, the normalized value is one. It is simply the ordinal position of the value of the attribute in the list. Similarly, if the value is a date, the normalized value is the time in milliseconds from the Unix epoch (00:00:00 UTC on 1 January 1970). This list of normalized values then becomes a locally available index that a comparator can easily sort, ensuring both speed and reliability.

3.4 Structuring Policies for Enactment

As explained above, we capture the local preferences of a site, the various licenses, and the strategies to be applied during license administration as policies. Further, so as to uniformly accommodate the above policies, we organize them hierarchically using XACML's *Reference*

element. As seen in Figure 5, at the root level are the local policies of a site viewed as a host. These apply to every request and are stored in the same policy set. An example local host policy might be "Take no requests between 10:00 AM and 12:00 PM." Policies at the next level down capture the host's business relationships with various licensees. An example host policy pertaining to a licensee might be "Take no requests from NCCU on weekends." These policies are stored in separate policy sets, one for each licensee. Once the host has enforced its policies, it forwards the user request to the licensee, which then performs a license check and applies any scheduling policies. All licenses from a licensee are held in the same policy set, as explained earlier.

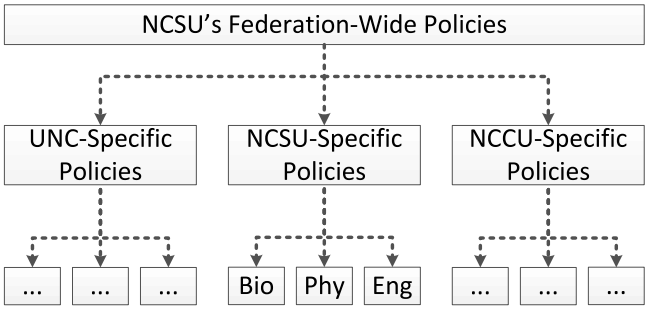


Fig. 5. Representation of hierarchical policies.

4 Implementation and Evaluation

We have applied our approach in formalizing the entire set of licenses illustrated in Table 1. Each license type is captured via a templatic policy that can be instantiated for an instance of that license type that yield an executable policy. We have also formalized as policies some sample scheduling strategies based on time information about sessions, such as their duration.

4.1 Policy Engine

Our policy engine is adapted from an open-source XACML implementation provided by Oracle (previously Sun) [18]. Specifically, we implement the following abstract classes. One, the *Policy Finder* loads policies from a data store. Currently, the data store is a directory in the file system. This class enables enforcing a hierarchical application of policies that protects the host's interest, as Section 3.4 describes. Two, the *Attribute Finder* resolves the attributes that occur in a XACML policy. Three, the *Scheduler* maintains the index to the distributed instances and loads, interprets, and enforces scheduling policies. However, neither the scheduling policies are expressed in XACML nor the *Scheduler* is part of the XACML specifications, it is mentioned here only to list the three important modules that enable policies in our approach. Our implementation enables each agent to interpret the attributes occurring in policies based on the information model described in Section 2.1 above.

Upon receiving a request, the policy engine determines the applicable license. For each attribute that the license references (here, the subject's *department*), the policy engine indirectly contacts the attribute service via the agent who is supposed to *resolve* the attribute. The policy engine then computes a decision based on the attribute values it obtains.

4.2 Realizing Licenses on Apache VCL

Apache VCL, an Apache top level project [8], is an open-source toolkit for managing a cloud environment for sharing computational resources among collaborating organizations [7]. VCL is widely deployed in academic settings wherein users (mostly students or researchers) can execute software images for preallocated amounts of time.

When a user signs in, the VCL portal offers a list of resources that the user may request. The user may request the next available time slot or specify an acceptable interval. Taking into consideration ownership and licensing, the VCL portal determines the machine on which to provision the request. VCL currently performs only a rudimentary usage license check. We enhance it by replacing its license checking module with a request to our agent, that in turn treats the VCL Daemon (VCLD) as the resource framework that would provision the necessary resources to instantiate the request.

4.3 Performance Evaluation

Licit is a distributed approach wherein each license evaluation and associated scheduling decision is based on data aggregated by from one or more agents distributed across the federation. In such a setting, simply establishing correctness is insufficient because the system is unusable if it does not scale to handle thousands of users. Below we describe pertinent performance tests centered on the load understood as the number of concurrent requests made for resources. Note that concurrent requests is a significantly clearer requirement than concurrent users in the system since users may request resources at different times.

To evaluate the performance of our functional prototype, we set up the host and licensee on different machines with simulated users and an attribute authority running on the same machine as the licensee. Thus, this setup models a scenario where a user from one organization requests compute resources from a host elsewhere such that the host has to go back to the licensee to check on licenses before making a resource instance available. For each machine we used the default configuration of the JBoss Application Server^{TM} v4.2.3 with Java Runtime Environment^{TM} 5.0.22. The machines we used had 3.37 GB RAM with two Intel Xeon^{TM} 1.66 GHz dual core processors and ran the Windows XP Professional (version 2002) operating system.

To cover functional tests, we consider licenses based on conditions and attributes from Table 2 that fall under one or more of the following classes:

- *Host-resolved* (e.g., *Node OS*).
- *Licensee-resolved* (e.g., *Department*).
- *State-based* (e.g., *Counted*).
- *Rescheduled* (e.g., *Concurrent users*).
- Combining host- and licensee-resolved attributes.
- Attributes aggregated from multiple sources.

Next, for each license, we designed two requests such that one would result in a permit and one in a deny decision. Thus, we had 20 test cases in all from ten licenses that cover the criteria stated above. Beyond establishing correctness, the functional test cases help us evaluate the performance of our implementation.

Using the TestNG framework [19], we created a test suite that simulates a user by sequentially submitting the test cases in a single invocation of a thread. We simulated concurrent requests by increasing the number of threads. In TestNG terms, a test group is an abstraction that calls a test method. Among other parameters, this test group can be configured with invocation count that determines how many times the test method will be invoked and the number of threads that will complete the invocation. We created ten groups with the number of threads set to one, two, four, and so on up to 512, and the invocation count set to 256. Thus each group simulated a different number of concurrent requests. All the groups called the same test method 256 (invocation number) times, irrespective of the number of threads. Since the test method itself made 20 requests corresponding to the 20 test cases, each run of a group made 5,120 (256 times 20) requests for decisions. One cycle of testing ran the ten groups.

For each request, we logged its start and end times, test case name, decision, and the number of active threads. Since the positive and negative test cases were named appropriately, a simple visual inspection of the output, sorted on the test case name, was sufficient to verify the correctness of our implementation in permitting and denying requests.

The evaluations of the requests do not all require the same amount of computations. For example, a license based on the *role* of the user alone requires no accounting. By contrast, a license based on the number of *concurrent users* may require accounting and possibly rescheduling. We created ten additional licenses based on the *concurrent users* attribute. From these licenses, we fashioned two additional cycles of tests: one where the licenses only had to be accounted for and another that required rescheduling as well. In each of the three cycles, we initialized the licensee's PDP with 4,000 licenses in all, including the licenses related to the test cases.

For the data in Figure 6, we calculated the reciprocal throughput for n concurrent requests as the ratio of the *elapsed time* to the number of requests in that run. We define the elapsed time as the time taken between the first submission among the n requests and the response received on the completion of the n^{th} request. We observe

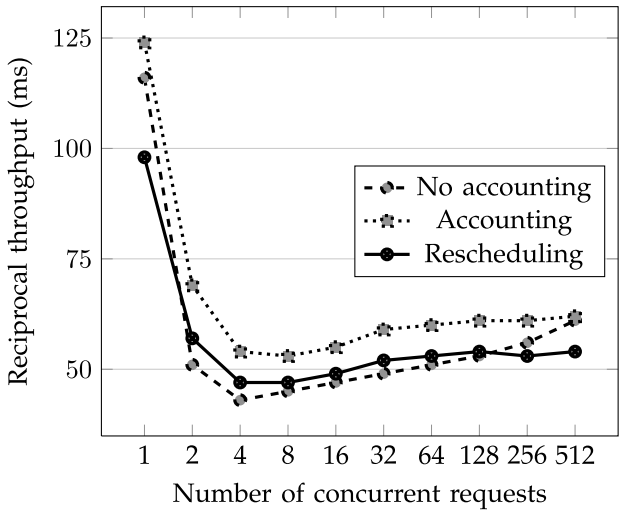


Fig. 6. Relationship between reciprocal throughput and number of concurrent requests (shown in a log scale).

that, as the concurrent load on the system increases, the throughput improves and then levels off. We observe that throughput improves significantly for up to four concurrent requests and then levels off, possibly because the machines used had four cores in total.

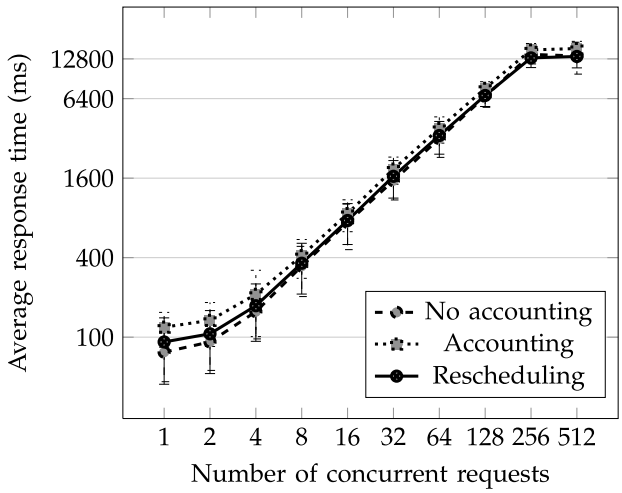


Fig. 7. Relationship between average response time and the number of concurrent requests. Both axes are in the log scale.

For Figure 7, the average response time was calculated over the entire population of samples collected from each run of a test group. In the graph, error bars for ± 1 standard deviation—calculated using the STDEV function in Microsoft Office Excel—from the average response time are shown: The standard deviation in these runs is on the order of 35% of the mean value. We see that the response time increases linearly with the number of concurrent requests.

It is worth considering the practical impact of these results. It is clear that increasing the number of concurrent requests increases the response time. However, our experiments consider a particularly demanding setting

wherein multiple users make *quasi-simultaneous* requests. Specifically, we generated peak request rates of greater than 60 per millisecond. A bursty load from a laboratory session for a university course would generally not have that many user requests hit the servers within the same millisecond. To validate the above intuition, we analyzed usage data obtained from NCSU's VCL installation over the entire semester of Spring 2012. We found that there were only six cases of two requests placed within the same second and there were zero cases of three or more requests placed within the second. Thus our tests exceed the needs of current usage patterns by a wide margin.

The longer response time seen in Figure 7 occurs because our test suite creates a sustained load of simultaneous requests that leads to a request queue building up. Thus, each decision request is simply spending time waiting to be processed. In any case, in cases where our system failed, it simply discarded the decisions and resumed normal operation when the bursty load subsided and more manageable loads resumed.

It is to be noted that some implementations of XACML, including the one in Python that one of the authors occasionally contributes to [20] and the one in Java that we used for our evaluation [18], process the policies such that the performance suffers linearly with the number of policies in the system. Perhaps in anticipation, XACML specifications [16] suggest an approach for policy indexing wherein the policies are stored in a database and the PDP queries only the applicable policies before evaluating them. In Licit's case this is the approach we took so that we cached the policies and indexed them by organization name, resource name, and action. Since all requests carry this information, the applicable policies can be easily retrieved.

Recently, Liu et al. [21] and others [22] have also tried to address the problem of linear performance. Our approach could be ported to any XACML implementation in principle. However, the above two implementations are still quite weak and lack sufficiently strong support in terms of bug fixes and continuing enhancements. Further, the lack of a standard API for XACML additionally complicates the porting task. Note that our implementation is open source and interested parties are welcome to port it to the XACML implementation of their choice.

4.4 Reproducibility of Results

Our entire code base is available as an open source project at http://www.opensource.ncsu.edu/LICIT.

In addition, the experimental setup described above for performance evaluation is available as an instantiable image. Readers are welcome to contact the authors to obtain access to that image.

5 Discussion

We find that a policy-based approach to federated usage license administration works naturally. It helps us capture the local concerns of each site while supporting collaborations between them. Whereas existing computational approaches for policies concentrate on low-level matters such as access control, we need policies dealing with the construction of cross-organizational partnerships. Thus Licit, by handling usage licenses well, can provide a well-grounded entry point into the study of policies in cross-organizations broadly.

Our approach makes licensing-relevant interactions explicit within a federation. Thus it can naturally support richer forms of accounting of licenses, such as where the licenses are being used and by users with what attributes. Such accounting can provide valuable insight needed for system administration, especially, in fine-tuning the strategies through which resources are allocated by an organization.

We are in discussions regarding moving our implementation of Licit into production.

5.1 Relevant Literature

FlexNet Manager [12] is the industry leader in software usage license administration. It enables a software vendor to specify licenses in terms of the features of its product and helps a user organization monitor its license usage for various software products. Licit is more flexible than FlexNet and supports specifying licenses based on selected attributes of the subject, resource, action, and environment. Licit not only supports a customizable vocabulary, if necessary, it can also work on top of an existing FlexNet installation. Additionally, we support specifying attributes, such as *number of processors*, which are resolved by an host agent known only at run time (see Section 2.1). FlexNet does not support scheduling of licenses, which is supported in Licit. Further, Licit does not require software vendors to embed a license agent into their software. Instead, Licit handles license administration modularly through an external agent, which moreover can comply with the local needs of each site. The terms of each usage license can be negotiated between the software vendor who provides the resource and the licensee. A policy can be authored for each license, although most practical licenses would follow the templates illustrated by the examples of Table 1.

Competing research approaches for license administration are centralized. For example, in connection with TeraGrid, Ogawa et al. [23] propose an architecture and approach for licenses that hands over the tasks of license administration totally to the service provider. In other words, the service provider obtains the licenses and merely bills the user. Although the licenses and the computational resources may be obtained from different vendors, the two are conflated by the service provider. The user organization is thus unable to communicate or employ its existing licenses. Further, the user organization is unable to apply its strategies for reallocating resources among its various users.

Perry and colleagues [24], [25], [26] also propose policy-based licenses. However, these works do not sup-

port federated settings and do not provide a systematic vocabulary with which to specify licenses. Further, unlike Zhao and Perry [24], we do not require an ontology for licenses. We apply the XACML standard to state licenses as policies. Importantly, Noorian and Perry [27] describe patterns of licenses that corroborate the license types we presented in Section 2.

The European Union's SmartLM project [28], [14] recognizes the same situation as Licit, namely, that the challenge of licensing is the main obstacle to the expansion of cyberinfrastructure technologies. However, SmartLM's technical approach is different from Licit's in some key respects. SmartLM emphasizes service level agreements between a user organization and a provider. In this respect, SmartLM is similar to Ogawa et al.'s approach, as described above, and different from Licit. Licit supports cases where the licenses belong to the user organization—that is, they are prepaid. The Licit approach, although not the current implementation, can also accommodate cases that require incremental payment for licenses. Further, Licit includes support for licenses for being transmitted under the control of the licensee (user) organization, whereas SmartLM appears to treat licensing in a manner that hides considerations of licensing from the user organization. In SmartLM, the orchestration service coordinates between licensee and host but only for the coordinated allocation of licenses and resources.

More generally, however, SmartLM does not support federation in that it provides neither an interaction model using which heterogeneous sites can communicate nor an information model for the bases underlying license application. As we showed above, there are natural situations in current practice where the licensee and host must cooperate to resolve the value of a particular attribute. A telling example is the attribute *number of processors*. Thus SmartLM does not handle some important types of licenses that Licit accommodates. Also, a scheduling mechanism that compares distributed requests is missing.

GenLM [29] deals with the broad problem of licensing in grid and cloud environments. However, GenLM takes a quite different stance from that taken in Licit. In GenLM, a user sends a request (including a claimed license) to a provider (host in our terminology) and the host contacts the licensor to determine if the user has claimed a valid license. GenLM's technical contributions are in how it handles the messaging protocols, especially with regard to encryption and digital signatures. In contrast, we treat the above as established infrastructure. Our contribution is in how licenses are administered in terms of each host's policies and each licensee's strategies, which could depend on attributes not included in the license itself. Hence, we include a flexible information model using which a licensee could specify its own vocabulary.

GenLM addresses the challenge of lost messages by forcing a handshake in the spirit of the two-phase commit protocol for distributed transactions. We take an optimistic approach. Specifically, in Licit, if a message that checks in a license is lost, the state of the license at the licensee may spuriously get stuck in *in use*. We address this problem through a simple mechanism of leases wherein a license expires by default after a time period (specified by the user) for which the session is expected to last, unless it is renewed in the meantime. Our agent-based architecture supports another solution, namely, one in which the licensee conducts periodic checks with hosts: this is not yet implemented in Licit.

Kung et al. [30] posit four desirable features of infrastructure that supports authentication, authorization, and accounting. First, the infrastructure must allow subjects from one organization to use resources in another. Second, the infrastructure must be resilient to security attacks. Third, the infrastructure must facilitate usage via single-sign-on technologies. Fourth, the infrastructure must support heterogeneity. In our setting, resilience is achieved through the lower-level virtualization technologies and single-sign-on is provided by the federation through existing methods. We can safely assume that we would entertain only requests that originate from within the federation. The Licit approach clearly supports Kung et al.'s first and fourth requirements, especially due to our use of the XACML standard.

The Rights Expression Language [31] (REL) includes elements—agents, resources, rights, constraints, and requirements—that correspond to subject, resource, action, rules and conditions, and obligation in XACML. Thus XACML and thereby Licit are equivalent in expressiveness to REL. However, Licit provides scheduling, ad hoc specification of vocabularies, and late binding of values to attributes like *number of processors* that are resolved by the host. These features are essential in the self-service federated environment that is the setting for Licit.

A reduced profile of REL, such as Open Data Rights Language (ODRL) [13], can also be used to express the licenses described in the earlier sections. However, agents do not figure in the ODRL model. At best, ODRL provides the context element, *Service*, to identify a location—not an agent—that would resolve an attribute. And, as we showed in Section 2.1, doing so is not sufficient in federated settings. Although the URI meant to define a *Service* in ODRL can also be used to name an agent, ODRL did not intend for the semantics to be such. In any case, to accommodate our scenario, ODRL would need to be extended to support important licenses available in Licit, such as the concurrent use license. Dahlem et al. [32] agree on the need to extend ODRL to express software usage licenses.

The body of work by Gangadharan and colleagues on extending ODRL and presenting a sophisticated analysis of service licenses [33] and license composition [34] is valuable. If necessary, their insights can be incorporated into Licit due to XACML's equivalence in expressiveness with REL. Interestingly, Gangadharan et al. [35] affirm

the need for consumer-specified licenses in federations but address it at the level of subsumption between provider and consumer specifications. More recently, Truong et al. [36] call for a participatory effort toward standardization of terms for modeling data contracts. We conjecture that our vocabulary-based declarative approach could play well as a low-level mechanism that enables consumers to address challenges in realizing consumer-specified licenses. Related to the above is Conway et al.'s [37] approach for data governance using the iRODS system. Our approach could facilitate expanding such work to federated environments, where there are of necessity multiple loci of policy application.

5.2 Directions

Each organization may potentially use its own strategies for dealing with different license types. However, in practice each organization would converge to a few "best practice" strategies that its administrators find acceptable and which interact well with the strategies adopted by the organizations with which it interacts. An important future direction is to enhance Licit in two respects. First, for most system administrators, Licit should come ready with a small number of strategies, judged to be sound by us and validated with our target user community. Licit will include a simple dashboard by which a system administrator may specify (i) which existing strategy to use for which specific license (depending on its license type); and (ii) any parameters to be specified for that strategy. For example, a possible strategy may be to limit the amount of time each user is alloted on an application when its usage load goes above a specified threshold—the parameters would be the usage load threshold and the time alloted to each user. Second, Licit should provide a simple enactment tool that supports simulating the usage of different resources as well as the messages sent and received based on specified strategies. In this manner, system administrators could determine if the strategies they choose are unstable or imperfect in other ways.

The development of policy-based approaches for various system administration tasks is a major trend in industry and academia. Over the last few years, a number of policy approaches have been proposed for resource management in the Grid, for example, by Dumitrescu et al. [38] and Wasson and Humphreys [39]. Such approaches are valuable but they largely disregard cross-organizational aspects. Udupi and Singh [40], [41] use the term *governance* for a way to administer resources in a cross-organizational setting, and develop a policy-based approach for governance. Udupi and Singh propose a more expansive architecture than we use in Licit, though license administration is a particularly important variety of governance. Udupi and Singh's architecture includes a Policy Organization Point (POP), which supports the modification of organizational relationships. Although we follow Udupi and Singh's approach in conceptual terms, in the interest of developing a robust, ready-to-deploy system, in Licit, we have omitted the machine representation of extensive organizational relationships and reasoning and modifications about such representations.

Indeed, Licit demonstrates a particularly important and immediate example of the problem of governance of federated systems, which is drawing the attention of several researchers [9]. Specifically, Singh [42] has recently developed a representation for cross-organizational "sociotechnical" systems based on normative relationships such as commitments, authorizations, prohibitions, powers, and sanctions. These normative relationships help characterize a federated system declaratively in conceptual terms by providing a basis for capturing the interactions of autonomous parties. Each relationship provides a basis for the policies of the interacting parties. We hope to investigate the enhancement of Licit to accommodate such broader, normative, forms of governance in cross-organizational settings.

Acknowledgments

This work was supported partially by the NCSU ITng initiative. We benefited from discussions with Matthew Arrott, Wayne Clark, Aaron Peeler, Josh Thompson, David Thuente, and Mladen Vouk. In particular, Aaron Peeler provided us statistics regarding VCL usage. We are also indebted to the anonymous reviewers for helpful comments.

References

[1] I. T. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," *CoRR*, vol. abs/0901.0131, 2009, http://arxiv.org/abs/0901.0131.

[2] H. E. Schaffer, S. F. Averitt, M. I. Hoit, A. Peeler, E. D. Sills, and M. A. Vouk, "NCSU's virtual computing lab: A cloud computing solution," *IEEE Computer*, vol. 42, no. 7, pp. 94–97, Jul. 2009.

[3] M. B. Blake and M. N. Huhns, "Web-scale workflow: Integrating distributed services," *IEEE Internet Computing*, vol. 12, no. 1, pp. 55–59, Jan. 2008.

[4] S. Lu and J. Zhang, "Collaborative scientific workflows," in *Proceedings of the 7th IEEE International Conference on Web Services (ICWS)*. Los Angeles: IEEE Computer Society, 2009, pp. 527–534.

[5] J. Zhang, D. Kuc, and S. Lu, "Confucius: A scientific collaboration system using collaborative scientific workflows," in *Proceedings of the 8th IEEE International Conference on Web Services (ICWS)*. Miami: IEEE Computer Society, 2010, pp. 575–583.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, Apr 2010.

[7] M. Vouk, S. Averitt, M. Bugaev, A. Kurth, A. Peeler, H. Shaffer, E. Sills, S. Stein, and J. Thompson, "Powered by VCL-using virtual computing laboratory (VCL) technology to power cloud computing," in *Proceedings of the 2nd International Conference on Virtual Computing Initiative*, Research Triangle Park, NC, May 2008, pp. 1–10.

[8] Apache, "Virtual Computing Laboratory (VCL)," 2012, an Apache Software Foundation top level project: http://vcl.apache.org/.

[9] F. Brazier, F. Dignum, V. Dignum, M. N. Huhns, T. Lessner, J. Padget, T. Quillinan, and M. P. Singh, "Governance of services: A natural function for agents," in *Proceedings of the 8th AAMAS Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE)*, 2010, pp. 8–22.

[10] J. Park and R. Sandhu, "The UCON$_{ABC}$ usage control model," *ACM Transactions on Information Systems and Security*, vol. 7, no. 1, pp. 128–174, Feb 2004.

[11] M. Winslett, "Policy-driven distributed authorization: Status and prospects," in *Proceedings of the 8th IEEE International Workshop on Policies for Distributed Systems and Networks*. Bologna: IEEE Computer Society, June 2007, pp. 12–18.

[12] Flexera, "License administration guide," http://www.globes.com/support/utilities/LicenseAdministration.pdf.

[13] R. Iannella, "Open data rights language," Sept 2002, http://www.w3.org/TR/odrl/.

[14] C. Cacciari, D. Mallmann, C. Zsigri, F. D'Andria, B. Hagemeier, A. Rumpl, W. Ziegler, and J. Martrat, "SLA-based management of software licenses as web service resources in distributed environments," in *Proceedings of the 7th International Workshop on Grid Economics and Business Models*, ser. LNCS, vol. 6296. Ischia, Italy: Springer, August 2010, pp. 78–92.

[15] J. Li, O. Wäldrich, and W. Ziegler, "Towards SLA-based software licenses and license management in grid computing," in *From Grids to Service and Pervasive Computing*, T. Priol and M. Vanneschi, Eds. New York: Springer, 2008, pp. 139–152.

[16] OASIS, "eXtensible Access Control Markup Language (XACML) version 2.0 specification document," *OASIS Standard*, Feb. 2005, http://docs.oasis-open.org/xacml/2.0/access-control-xacml-2.0-core-spec-os.pdf.

[17] ——, "eXtensible Access Control Markup Language (XACML) version 3.0 specification document," *OASIS Standard*, Aug. 2010, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf.

[18] Oracle, "Sun's XACML implementation," http://sunxacml.sourceforge.net.

[19] C. Beust, "TestNG: Next generation Java testing," http://testng.org/doc.

[20] P. Kershaw, "NDG XACML," http://ndg-security.ceda.ac.uk/wiki/XACML.

[21] A. X. Liu, F. Chen, J. Hwang, and T. Xie, "Designing fast and scalable xacml policy evaluation engines," *IEEE Transactions on Computers*, vol. 60, pp. 1802–1817, 2011.

[22] Anonymous, "Enterprise Java XACML," http://code.google.com/p/enterprise-java-xacml/.

[23] H. Ogawa, S. Itoh, T. Sonoda, and S. Sekiguchi, "GridASP: An ASP framework for grid utility computing," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 6, pp. 885–891, 2007.

[24] Q. Zhao and M. Perry, "An ontology for autonomic license management," in *Proceedings of the 4th International Conference on Autonomic and Autonomous Systems*, March 2008, pp. 204–211.

[25] Q. Zhao, Y. Zhou, and M. Perry, "Policy-driven licensing model for component software," in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, Lake Como, Italy, June 2003, pp. 219–228.

[26] ——, "Agent design of SmArt license management system using Gaia methodology," in *Proceedings of the 3rd International Conference on Autonomic and Autonomous Systems*. Washington, DC: IEEE Computer Society, 2007, pp. 9–15.

[27] L. Noorian and M. Perry, "Autonomic software license management system: An implementation of licensing patterns," in *Proceedings of the 5th International Conference on Autonomic and Autonomous Systems*. Los Alamitos, CA: IEEE Computer Society, 2009, pp. 257–263.

[28] European-Commission, "SmartLM project web pages," Feb 2008, http://www.smartlm.eu.

[29] M. Dalheimer and F.-J. Pfreundt, "GenLM: License management for grid and cloud computing environments," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Shanghai: IEEE Computer Society, May 2009, pp. 132–139.

[30] H.-T. Kung, F. Zhu, and M. Iansiti, "A stateless network architecture for inter-enterprise authentication, authorization and accounting," in *Proceedings of the 3rd International Conference on Web Services*, Las Vegas, Nevada, USA, June 2003, pp. 235–242.

[31] R. G. Gonzalez, "A semantic web approach to digital rights management," Ph.D. dissertation, Universitat Pompeu Fabra, 2005.

[32] D. Dahlem, I. Dusparic, and J. Dowling, "A pervasive applications rights management architecture (PARMA) based on ODRL," in *Proceedings of the 1st International ODRL Workshop*, April 2004, pp. 45–63.

[33] G. R. Gangadharan and V. D'Andrea, "Service licensing: conceptualization, formalization, and expression," *Service Oriented Computing and Applications*, vol. 5, pp. 37–59, 2011. [Online]. Available: 10.1007/s11761-011-0079-6

[34] G. R. Gangadharan, M. Weiss, V. D'Andrea, and R. Iannella, "Service license composition and compatibility analysis," in *Proceedings of the 5th International Conference on Service Oriented Computing*, ser. LNCS, vol. 4749. Vienna: Springer Berlin / Heidelberg, 2007, pp. 257–269.

[35] G. R. Gangadharan, H.-L. Truong, M. Treiber, V. D'Andrea, S. Dustdar, R. Iannella, and M. Weiss, "Consumer-specified service license selection and composition," in *Proceedings of the 7th International Conference on Composition-Based Software Systems*. Madrid: IEEE Computer Society, 2008, pp. 194–203.

[36] H.-L. Truong, G. Gangadharan, M. Comerio, S. Dustdar, and F. De Paoli, "On analyzing and developing data contracts in cloud-based data marketplaces," in *Proceedings of the Asia-Pacific Services Computing Conference*, Dec. 2011, pp. 174–181.

[37] M. Conway, R. Moore, A. Rajasekar, and J.-Y. Nief, "Demonstration of policy-guided data preservation using iRODS," in *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*. Pisa: IEEE Computer Society, Jun. 2011, pp. 173–174.

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

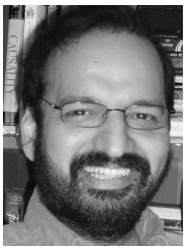
[39] G. Wasson and M. Humphrey, "Toward explicit policy management for virtual organizations," in *Proceedings of the 4th International IEEE Workshop of Policies for Distributed Systems and Networks (POLICY)*, 2003, pp. 173–182.

[40] Y. B. Udupi and M. P. Singh, "Multiagent policy architecture for virtual business organizations," in *Proceedings of the 3rd IEEE International Conference on Services Computing (SCC)*. Chicago: IEEE Computer Society, 2006, pp. 44–51.

[41] ——, "Governance of cross-organizational service agreements: A policy-based approach," in *Proceedings of the 4th IEEE International Conference on Services Computing (SCC)*. Salt Lake City: IEEE Computer Society, 2007, pp. 36–43.

[42] M. P. Singh, "Norms as a basis for governing sociotechnical systems," *ACM Transactions on Intelligent Systems and Technology (TIST)*, pp. 1–21, 2013, to appear; available at http://www.csc.ncsu.edu/faculty/mpsingh/papers.

Prashant C. Kediyal has over 10 years of software engineering experience primarily from working in the banking and insurance industry. He is now a Ph.D. student in the Department of Computer Science, North Carolina State University, Raleigh. His research interests include multiagent and policy-based systems, service-oriented architecture, and governance.



Munindar P. Singh Munindar P. Singh is a Professor in the Department of Computer Science, North Carolina State University, Raleigh. His research interests include multiagent systems and service-oriented computing with a special interest in the challenges of trust, service selection, and business processes in large-scale open environments. His books include the coauthored *Service-Oriented Computing* (Wiley, 2005). Singh is the Editor-in-Chief of the *ACM Transactions on Internet Technology* and was the Editor-in-Chief of the *IEEE Internet Computing* from 1999 to 2002. He is a member of the editorial boards of *IEEE Internet Computing*, *Autonomous Agents and Multiagent Systems*, *Journal of Artificial Intelligence Research*, and the *ACM Transactions on Intelligent Systems and Technology*.