Policy-based automated provisioning

by K. Appleby
S. B. Calo
J. R. Giles
K.-W. Lee

The term "policy-based computing" refers to a software paradigm that incorporates a set of decision-making technologies into its management components in order to simplify and automate the administration of computer systems. A significant part of this simplification is achieved by allowing administrators and operators to specify management operations in terms of objectives or goals, rather than detailed instructions that need to be executed. A higher level of abstraction is thus supported, while permitting dynamic adjustment of the behavior of the running system without changing its implementation. This paper focuses on the application of the policy-based software paradigm to the automated provisioning architecture described elsewhere in this issue. We show how the use of policies can enhance utility computing services by making them more customizable and more responsive to business objectives.

A *utility computing system* is a system for creating and managing multiple instances of a utility service within a shared IT (information technology) infrastructure.[1] A service provider maintains an aggregation of computing resources that can be allocated to different services. Customers of the utility computing system request access to services of particular types, and instances of these services are provisioned to meet their needs.

In IBM's on demand architecture, each instance of a utility service is called an "utility computing service environment" (UCSE). In creating the UCSEs, requests are made to resource managers (RMs) that keep track of certain types of resources, their allocations, and availabilities. There typically are RMs for servers, storage, switches, routers, middleware, and so forth. The RMs provide resources to the UCSEs, which are appropriately configured to meet the functionality and performance requirements of the particular service being supported. This will be described in greater detail in the section "Background."

Policies are considerations designed to guide decisions on courses of action and can be used for numerous purposes within utility computing systems. The service provider responsible for the computing environment manages the policies, which determine how the environment is shared; for example, which customers have priority, how reservations are managed, how costs are allocated, and so forth. A service provider with various environment instances (aggregations of computing resources) manages policies such as how each instance should be configured and operated, how performance should be measured, what to do in case of component failures.

The RMs that administer pools of specific computing resources manage policies regarding how resources are reserved, whether overbooking is allowed, how resources are monitored, and so forth. Resource-specific policies depend upon the characteristics that are associated with particular resource

types: for example, storage systems have different characteristics (space allocated, striping, access control) than networks (bandwidth allocation, packet loss rate). A policy framework provides a general, formalized way of controlling such customization and variability within a system through the use of policies.

As described in Reference 1, a utility computing infrastructure architecture has been defined, and instances of it are being developed in order to validate its usefulness and completeness. Policy-based technologies are also being defined,[2] and standards are evolving in certain areas.[3] These technologies, however, are still not very mature, and the structure of policies and the interrelationships among them are described differently in different systems. It is thus of interest to investigate how policies can be applied to utility computing systems, what types of policies are needed, and how they extend the capabilities of the overall environment.

In the next section, the utility computing infrastructure is presented and its principal architectural components described. In the section "Policy-based computing," the concept of policy-based management and a general framework for its application is introduced, and in the section "Policy enablement," we show how policy-based technologies can be applied to the utility computing infrastructure.

In the section "Utility computing policies," we describe the types of policies that might be used within such a system and how they may interrelate. In general, the policy architecture must capture the details of the various policy rules in a policy schema and establish how policies are created and enforced within the system. It must also indicate how policies are related to other system decision-guiding structures like service level agreements (SLAs) and rules. This is covered in the section "SLAs, policies, and rules," where it is also noted that certain resource policies may be derivable from higher level aggregate policies or business objectives, while others have to be directly specified.

In the section "A gaming service example," a detailed description of a gaming service is presented, along with specific policies that can be used in its operation. We discuss the principal sources of the policies for such a service and how they are deployed. Finally, in the section "Conclusions," we comment on the status of efforts in this area, conclusions that can be drawn, and projected future work that should be undertaken to advance the state of policy-based systems.

Background

A utility computing system creates and manages multiple instances of a utility service, each of which provides application functions to customers. Each UCSE offering defines a UCSE type that can be built and deployed on demand. Environments that are good candidates for instantiation as UCSEs are those that need a significant number of resources for a short period of time, those that have complex requirements so that users may not have the skills or time to deploy them, and those that have resource needs that vary over time and can thus take advantage of a shared resource pool. A utility computing system can rapidly deploy complex UCSEs that dynamically and autonomically adjust capacity, using the services that the utility computing system provides.

Customers subscribe to on demand services (ODSs) using the OGSA (Open Grid Services Architecture) business-provisioning service (OBPS). The OBPS contains facilities supporting subscription, authentication, metering, SLA management, pricing, and rating. In addition to the OBPS, the utility computing system contains an OGSA distributed resource manager (ODRM). The ODRM is responsible for instantiating and managing the UCSEs. Once a UCSE has been created in response to a request to the OBPS, the customer accesses the service directly. The principal architectural components that comprise the utility computing infrastructure are shown in Figure 1.

The ODRM contains a planner whose purpose is to build workflows that create, configure, and adjust the working set of resources for the UCSE. When a user subscription is processed, an ODS instance for the environment is generated. The workflows created by the planner are stored in the new instance service. To generate the workflow, the planner uses input from the parts catalog and its referenced resource managers and from resource services (RSs), an ODS template, and the policy database. The information collected from these sources is described in Table 1.

The planner first builds a parts topology tree, the leaves of which correspond to operations on RMs or RSs. The leaves of the tree encapsulate the operations that need to be performed to build the ODS environment. Workflows are generated by collecting all of the operations referenced in the topology

Figure 1 Utility computing infrastructure architecture

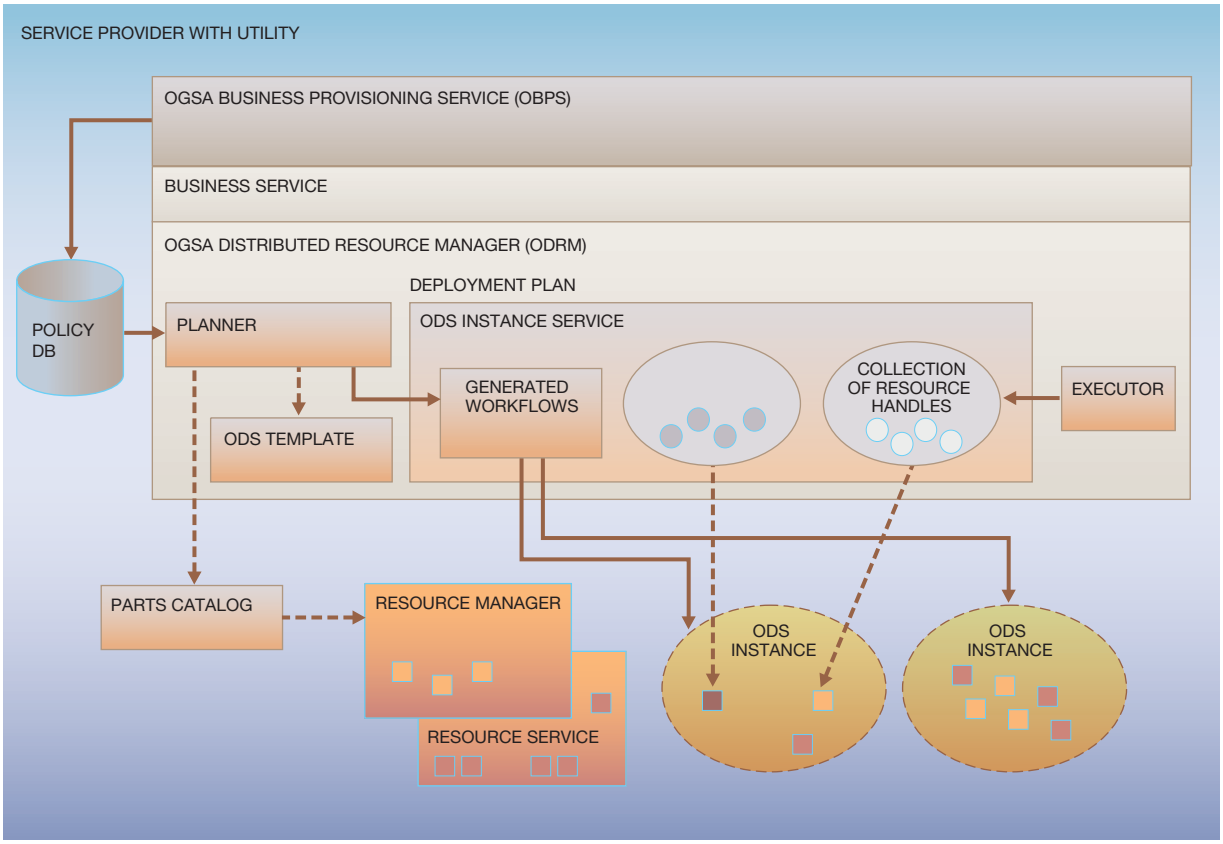


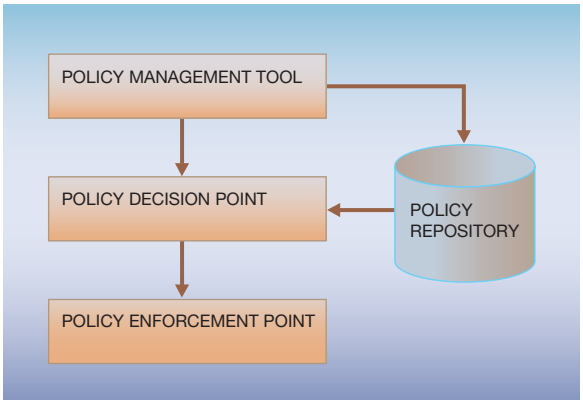
Table 1 Planning information

Parts catalog	A set of parts, each of which describes one of the provider's resources or processes. Parts are directly related to the resources they encapsulate, and map workflows to operations on resource managers (RM).
RM	An RM manages a specific type of resource. There is one RM for each resource type. It may handle operations such as reservation, allocation, and configuration.
RS	An RS is any service operating on infrastructure resources that is not a base resource operation.
ODS template	The ODS template contains a pointer to the root *part* of the ODS environment being built. It also contains values for the attributes found in the parts used to build the workflows that are not obtainable from the policy database.
Policy database	The policy database contains provisioning policies such as the minimum and maximum capacity of the ODS environment, the date the system is to go online, the performance thresholds to be used, availability requirements, and service class subscribed to.

tree's leaves. Each workflow must ensure that every variable is generated by a process that computes its value before it is passed to any other process as an input.

After the ODS instance is instantiated with its workflow set, the reservation workflow can be invoked. It is not until this workflow has been successfully run that the reservation can be accepted and committed

Figure 2 General policy framework



to the system. At activation time, the "create" workflow moves resources from the free pool and configures them appropriately for the new ODS environment that they will support.

Policy-based computing

The Internet Engineering Task Force (IETF) has adopted a general policy-based administration framework that consists of the four basic elements shown in Figure 2: a policy management tool, a policy repository, a policy decision point, and a policy enforcement point.

Administrators define the policies that they wish to use in the operation of the system with the policy management tool. Once defined, these are stored in a policy repository. In order to ensure interoperability across products from different vendors, information stored in the repository must follow an information model specified by the IETF's Policy Framework Working Group.[4] The actual points within the system software at which policies are executed are known as *policy enforcement points* (PEPs).

Instead of communicating directly with the repository, policy enforcement points use intermediaries known as *policy decision points* (PDPs). The PDPs are responsible for interpreting the policies stored in the repository and communicating them to the associated policy targets in whatever format is appropriate. Associated PEPs and PDPs may be in a single device or in different physical devices. Different protocols may be used for various parts of the architecture, for example the COPS (Common Open Policy Service) protocol[5] or SNMP (Simple Network Management Protocol)[6] can be used for communication between PDPs and PEPs, and the policy repository can be a network directory server that is accessed using LDAP (Lightweight Directory Access Protocol).[7,8]

The Distributed Management Task Force (DMTF) is also involved in the creation of policy standards. A high-level policy schema is included as part of the overall common information model (CIM) schema.[9] This defines policies as "condition, action" rules that can be aggregated into policy sets that have policy roles.

The primary advantage of a policy-based approach is that it simplifies the complex task of administering large, distributed systems by allowing the specification of management operations in terms of high-level objectives rather than detailed device-specific parameters. It also supports separation of concerns, in that decision points are called out explicitly, and considerations that would lead to various alternatives are captured in the policies. The use of a logically centralized repository also enables detection of possible conflicts between the policies assigned to different devices.

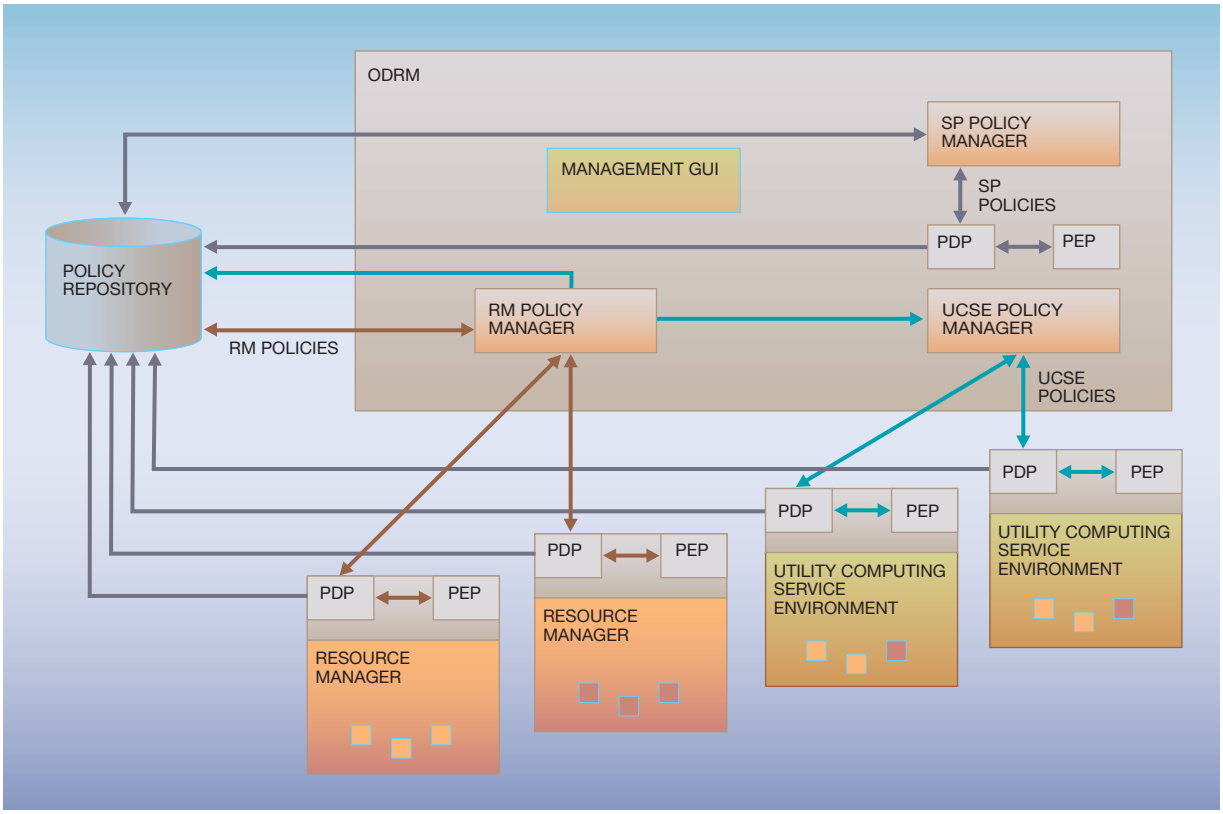
Policies may be grouped according to *disciplines*, that is, the particular aspects of a system that they have been created to support (e.g., network quality of service [QoS] or intrusion detection). Some aspects of applying policies to a given discipline are specific to the particular discipline, whereas others can be provided in a generic manner by using a set of common algorithms and functions. Examples of capabilities that can be provided in a generic fashion are: checking if a policy is valid (Does it conform to the policy schema?), determining whether a set of policies is consistent (Are there conflicts?), checking if a policy is redundant (Is it covered by a combination of other policies?), and determining if a policy is feasible (Would it ever be executed?).

Additionally, each PEP and PDP in the system may take on one or more *roles*. Roles further refine policies within a discipline. Thus, for network QoS, all access routers may have the role of "edge," whereas all internal routers may have the role of "core," and they retrieve their respective policies from the policy repository based upon their roles.

Policy enablement

The utility computing infrastructure can be "policy enabled" by incorporating a *policy service* (made up

Figure 3 Utility computing policy service



of the components depicted in Figure 3) in the overall architecture. There are three policy managers that control access to policy creation and maintenance for the three different types of policies in the system, namely: (1) those for the *service provider* (SP), which deal with the sharing of the computing infrastructure among different ODSs; (2) those for *utility computing service environments*, which deal with policies particular to the allocation and management of computing resources supporting a given ODS; and (3) those for *resource managers,* which deal with the administration of pools of specific resources.

PDPs are associated with each SP, UCSE, and RM instance. These, in turn, have associated PEPs that are responsible for the enforcement of policies particular to their specific context. Figure 3 depicts the logical components of the policy service. These can be implemented in various ways. The PEPs for each UCSE, for example, could be provided by a single software component that has knowledge of the different instances to which the policies must be applied. Policies are communicated via a common policy repository, where they are stored as XML (Extensible Markup Language) documents. The policy service has a common management graphical user interface (GUI) with plug-ins for each of the different types of policies being supported (SP, UCSE, and RM).

The policy managers are meant to deal with policy information at a level of abstraction that is appropriate for interactions with a system administrator. The policy schema specifying the interactions between the discipline GUI plug-in and the policy service manager should thus capture policy information at this level. The policies used by a PDP are meant to cover an administrative domain, and so the policy schema used between the policy service manager and the PDP should be at this "domain level." The policies used by a PEP need to specify device-specific rules for a particular functional area. The "device-level" policy schema used between the PDP and the PEPs must reflect this system- and capability-specific information. It would thus typically not be

Table 2 Basic policy types by category

	Policy name	Description
SP utility policies	Resource distribution policy	Arbitration
	Reservation policy	How far in the future
	Pricing policy	Price of environment
UCSE policies	Resource acquisition policy	Maximum number of servers; when to add
	Failure recovery policy	Replacement policy
	Pricing policy	Price to UCSE user
	Monitoring policy	Aggregate metrics
	Service-specific policy	Provisioning constraints, workload management, etc.
RM policies	Overbooking policy	Amount of overbooking allowed
	Forecasting policy	Order new resources
	Pricing policy	Price of resource
	Resource monitoring policy	Availability
	RM-specific policy	For example, networks, storage, and servers

stored in the policy repository and may take any proprietary form, from an XML schema to a set of system-specific configuration parameters.

The manner in which the policies at the different levels are created, deployed, updated, and enforced within the system is part of the policy architecture that must be specified for every different type of utility computing service. Certain device-level policies might be derivable from domain-level policies, or explicit transformations might be incorporated into the PEP specification. Alternatively, certain policies (like access control lists) may need to be specified at the administrator level and simply passed through to the particular devices that use them.

Utility computing policies

As mentioned above, there are three principal categories of policies related to the provisioning of services within a utility computing system: those that deal with the sharing of the computing infrastructure among different ODSs; those that deal with policies particular to a given ODS; and those that deal with the administration of pools of specific resources. The general types of policies within each of these categories are listed in Table 2.

Each of these policy types will be described in the following sections. The list is not meant to be exhaustive, but to provide examples of the kinds of considerations that might be applicable in utility computing systems. Each class of utility typically has its own specific sets of policies. We present here those that are considered to be somewhat generic; in a subsequent section we provide examples of policies that are relevant to a specific type of utility.

Service provider policies. The service provider requires policies for resource distribution, reservations, and pricing. *Resource distribution policies* are used to arbitrate between the different demands of the service instances that are being provided within the shared utility computing infrastructure. The policies may be as simple as "first come, first served," or may involve priorities with preemptions (i.e. taking resources from a lower priority instance to give to a higher priority instance) or penalty functions (i.e. taking a resource from the instance that causes the least loss in revenue). *Reservation policies* specify whether classes of resources can be reserved for use by a planned service, how far in advance reservations can be made, what happens if the service cannot be provisioned at the time promised, and so forth. Resource distribution and reservation policies interact with one another. Obviously, the manner of interaction between services that are running and services that are reserved but not yet provisioned must be specified. *Pricing policies* have to do with issues such as the way in which customers are charged (using a flat fee, based on usage, etc.), whether rebates apply and under what conditions, and whether refunds must be made if resource distribution and reservation policy goals are not met. These policies also interact with resource distribution and reservation policies, which means that potential inconsistencies and conflicts must be resolved in the design of the policy architecture and the concomitant schemas.

UCSE policies. UCSE policies capture considerations that apply to aggregations of resources allocated to a given ODS instance. Examples include policies for resource acquisition, failure recovery, pricing, and monitoring. Considerations such as when to ask for additional resources, and, for cost containment purposes, the maximum number of resources allowed per service instance, are managed by *resource acquisition policies*. *Failure recovery policies* specify whether failed resources need to be replaced, whether checkpoints need to be taken and how often, whether hot standbys are required, and so forth.

The *pricing* at this level is intermediate between the SP and RM pricing. It aggregates the resource costs and adds additional cost considerations due to overall service factors like congestion, failure, and QoS options. These have to reflect the operation of the specific configuration of the multiple resources comprising the service and may even involve other services that are subsumed (e.g., a Web hosting service might incorporate a network communications service). These costs are taken into consideration by the UCSE owner when the pricing policy is set. *Monitoring policies* describe what metrics need to be provided, to whom they should be provided, and how often; these are specified with respect to the aggregate configuration (e.g., end-to-end response time).

In addition to the types of policies discussed earlier that deal with aspects of participation in the utility, there are policies defining the behavior of specific UCSEs that would need to be specified by the ODS developer. Some of the general categories of policies of this type are those for provisioning constraints, workload management, metering, capacity planning, and failure recovery. These policies relate to particular classes of service instances and the specific control points that they externalize.

Provisioning constraints include such considerations as collocation (whether two resources must be hosted on the same server, or whether they must be on separate servers), and dependencies (whether a particular software resource requires another software resource or a particular type of hardware resource, whether one software resource needs to be started before another resource can start, etc.). *Workload management policies* deal with such considerations as the manner in which transactions are dispatched to clusters of servers and the priorities assigned to different transaction classes. To facilitate billing and accounting operations, the necessary usage and operational data must be metered. *Metering policies* determine what information needs to be gathered, how it needs to be summarized or otherwise combined, and where it is to be used. Obviously, the metering rules must support the overall pricing policies. *Capacity planning* is necessary in order to determine or predict the need for additional resources in order to meet the operational goals of the service instance, given its system context.

Resource manager policies. Resource manager policies that apply to the operation of the utility computing system include those that deal with considerations like overbooking, forecasting, pricing, and monitoring. RMs administer pools of resources, and can execute various strategies in allocating them to UCSE instances. *Overbooking policies* specify whether resources that are associated with pending reservations can be overcommitted, and, if so, what degree of risk is acceptable in this context. *Forecasting policies* determine when additional resources should be acquired, and may be simply based on the number of resources remaining idle, or may incorporate projections of load or new reservation requests. *Pricing policies* at this level deal with how the costs of individual resources are ascribed to the specific instances to which they are allocated. This can be stated in terms of a given fixed price per resource, based on a contractual commitment, or can be calculated based on a number of other considerations (e.g., amount of use, replacement cost, number remaining in free pool, etc.). *Monitoring policies* describe the metrics that need to be provided, to whom they should be provided, and how often. Typical metrics for computing resources include response times, loads, and utilizations. The RM policies interact among themselves and with the SP policies described earlier. The policy architecture needs to capture all of these relationships.

In addition to the types of policies that are defined by an administrator of the utility computing system, there are policies that manage the behavior of specific resources. These policies are specified by the RM developer and affect the various controls externalized by the particular class of resource. The most common of these policies pertain to networks, storage, and servers.

Network policies deal with such attributes as bandwidth, input and output queue-scheduling priority at the endpoint servers for each link, TCP (Transmission Control Protocol) congestion window, VLAN

(virtual local area network) configuration, VPN (virtual private network) configuration, and the number of concurrent connections supported. *Storage policies* deal with such attributes as storage allocation (size), data integrity, access control, response time, availability, request distribution (random or sequential), permanence, and so forth. *Server policies* deal with such attributes as CPU utilization, I/O rates, degree of multiprocessing, memory size, caching strategies, response time, availability, and so forth.

SLAs, policies, and rules

Any instantiation of the policy-based architecture will typically support customer services for which SLAs are defined. Certain parts of these SLAs determine policies for the system resources that were provisioned to fulfill them. SLAs are business contracts between organizations that define the obligations that each organization has to the other within some system context. Policies capture the directives of an organization and are generally configurable by administrative personnel. They provide guidance on what to do and when to do it and on the importance attached to a specific action or goal. Internalized policies or *rules* capture internal system logic and are not generally configurable by administrators although they may be changed by personnel with a detailed knowledge of how the system operates.

Thus SLAs are exposed inter-organizationally, policies are exposed within an organization, and rules are typically not exposed (except to development personnel). In certain cases there may be a one-to-one mapping between policies and portions of SLAs; in some cases, policies may be derivable from business objectives stated in SLAs. Similarly, in certain cases, policies may be rules themselves, though of necessity ones that have been externalized to system administrators, or policies may simply set conditions or action parameters in multiple system rules that are not externalized.

Policy transformations

As indicated earlier, there are a number of different types of policies at a number of different levels of abstraction that need to be considered in the utility computing infrastructure. Some of these are specified in the SLAs associated with instances of ODSs; some are derived from business objectives in the SLAs; some are specified by the service provider through an administrative console; and some are determined by the developers of the UCSEs and RMs that externalize specific *decision points* to which policies may be applied.

Ideally, one would typically like to be able to derive policies from business objectives, and derive lower-level resource policies from higher-level aggregate policies, in as automated a manner as possible. The problem of determining the right set of underlying policies to meet a business objective can be solved in different ways, depending upon the particulars of the required transformations. We consider some general approaches next.

Simple transformations. In many circumstances, the mapping from higher-level policies (or business objectives) to lower-level policies can be done directly with simple transformations. This is the case when information is only being refined, as in a direct substitution (e.g., mapping domain names to IP addresses). This is also the case when the higher-level definition is actually a class definition that aggregates a number of different attributes at the lower level (e.g., associating a class of service like "gold" to a given set of goals for network parameters, such as response time and packet loss rate). Even when higher-level objectives are being mapped onto lower-level policy constructs, the transformations may be simple. In determining the number of servers that are needed to meet a desired response time and the class of service that each server must provide, a simple table of server characteristics might suffice. Also, if the underlying system only supports a small number of alternative configurations, a simple search may be sufficient to see which is needed to meet a desired objective (e.g., an availability target).

Analytic models. For more complex transformations, if an analytic model can be developed to determine the business objective as a function of the underlying policies, the model parameters that would satisfy the required business parameters can be determined. For example, if an analytical expression exists to determine the outbound bandwidth needed to support a given inbound traffic rate at a Web site, the expression can be inverted to obtain the requisite traffic rate for any desired limit on outbound bandwidth. That is to say, if there exists a known closed form expression for the function f, where $f(p) = b$, b is the bandwidth limit, and p is a vector of policy values, it would be possible to use numerical or other techniques to find the values for the components of the policy vector p that would attain the value b.

Generic models. For some performance characteristics, such as response time and throughput, computing systems are often modeled as queuing networks. Under certain sets of simplifying assumptions, queuing network parameters can be derived analytically, but most often are arrived at by simulation techniques. Given such a parameterized model, one could solve for the model parameters that would satisfy the required business parameters.

Online adaptive control. Other classes of parameterized models, like those based on concepts from control theory or statistical techniques like linear regression, can be used to determine and fine-tune the parameters needed to obtain a given business objective. Neural networks can also be used to determine the impact of specific policy parameters on the requisite business objective. The neural network or adaptive control scheme could then be used to dynamically adapt policy parameters to meet the desired goals in the running system. This approach is fairly flexible, but generally requires extensive training data.

Case-based reasoning. In certain instances, it may be possible to use learning techniques to develop implicit models of system behavior. In a case-based reasoning (CBR) approach, the settings for system parameters that attain certain goals would be learned experientially from historical data. A database of past cases is maintained, where each case is a combination of the policy parameters and the business objectives that were achieved when those parameter values were used. In order to determine the set of policies that would be needed to achieve a new business objective, the case database is searched to find the closest matching case, or the existing cases that bracket the new requirement are interpolated, to establish the appropriate settings. The definitions of the cases and the strategies for interpolation tend to be specific to the particular policy discipline being considered, but the same case manipulation software and algorithms can be used across different disciplines.

The CBR approach depends on extensive historical data in order to build a set of cases rich enough to be consulted in guiding new decisions. In a system that has been running for an extended period of time, it is possible to build cases from prior experience. However, at bootstrap time, there is no prior experience to exploit. Thus, a CBR approach needs to be combined with an heuristic approach (using an analytical expression or approximation), which is used until enough historical data is collected, or the system has to be pre-populated with a set of cases synthetically derived or obtained experimentally.

A gaming service example

Consider an instance of a utility computing system that supports multiplayer gaming, that is, an "ODS game environment offering." It provides specific services for the development, testing, and production of online games within a shared computing infrastructure. Using this gaming utility, dynamically scalable *game environments* can be deployed and managed to meet the changing needs of game developers and publishers. The creation of a *game environment instance* is presented in Reference 1 and will not be discussed here, except as it relates to policy.

Basic components. The basic game environment contains two types of dynamically allocated resources: a set of game servers that handle the game logic, and a set of proxy servers that handle player operations such as log-in, session selection, chat, and security checks, as well as functions such as session creation and management. In addition to these variable-capacity resources, there are a number of fixed resources, including:

- A database server where game states are stored
- A game ODS management node where the OBPS and ODRM components reside
- Various network components

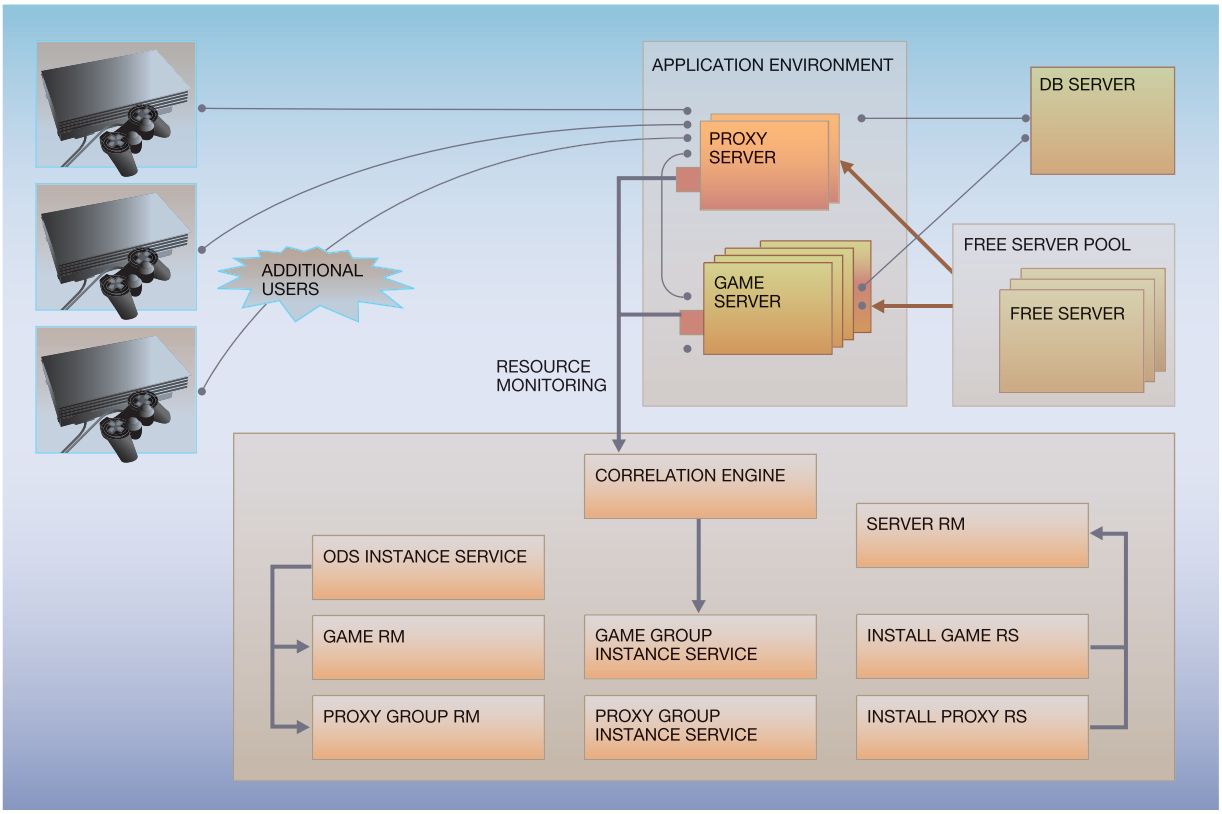
A game ODS instance, with its associated links to the main infrastructural components is depicted in Figure 4.

Basic policies. Most of the policies used to control the gaming environment are based on templates. For example, consider the MonitorDelay policy which monitors the average processing delay of the Game servers.

```
Rule: MonitorDelay
If (averageProcessingDelay(event.serverData)
> delayThreshold(event.GameID)
  & onLine(event.serverID)
  & ruleArmed("MonitorDelay",
event.GameID)
{rebalance_load(event.ODSOwner)}
```

This rule is stored in the policy repository at ODS creation time (i.e., when the ODS gaming offering is defined). However, it is not until UCSE instance cre-

Figure 4 Gaming utility computing service



ation time that the `delayThreshold` for the given `GameID` is stored in the policy repository. Values such as these that are specific to the game instance are derived from the SLA for the particular UCSE instance. The SLA may additionally specify instance-specific rules that do not apply to all game instances. For example, the game servers that contain game-specific processes may require special process monitoring and a set of associated policies to handle these monitored values. SLAs that capture various policy components pertain between the Internet service provider and the ODS owner, the service provider and the ODS owner, the player and the ODS owner, and the ODS owner and the game developer.

The gaming utility has a number of different types of policies, each of which will be described in turn. The basic policy types are summarized by category in Table 3. These are versions of the more general policy types previously described for utility computing systems.

SP policies. Arbitration policies are simple: any game instance will not be allocated more than its maximum or less than its minimum number of servers, and allocations are processed on a "first come, first served" basis.

RM policies. The current reservation policy accepts a gaming ODS reservation if it does not request more server resources than the ODS offering maximum, is not more than some fixed number of days in advance, and allows for a sufficient number of days for resources to be acquired if they are not currently available. If the resources are not already available, the reservation policy uses the average length of time required to order and receive a new resource to determine if resources can be acquired by the target activation date. The free pool always keeps some surplus servers available for dynamic allocations.

The capacity planning policy is limited to the submission of a request for resource ordering when the

current reserve falls below a given threshold. The size of the free pool is the only parameter considered in this policy.

ODS offering policies. The performance policies for the gaming utility cover aggregation and thresholding. Metric collection agents residing on the allocated servers supply the raw performance data to the PEP. The PEP contains a single generic rule for each server type (game or proxy) that aggregates and smoothes the incoming data. Separate base rules are required, as different performance metrics are utilized for each server type. At runtime, the PEP creates an instance of the base rule for each game environment. Associated with each rule instance is a collection object (the *gameInstance-serverType* collection) that was supplied at resource allocation time. This collection object keeps track of which resources belong to which environments and the type of each resource.

Periodically, the "aggregate and smooth" data policy generates an internal summary event, which is used by a thresholding rule. Again, there is a separate rule for each server type and an instance is generated at runtime for each game environment. If the performance threshold is exceeded, the recovery action will allocate additional capacity (if available) to the server collection in question. When a recovery action is taken, the triggering rule is suspended for a period of time to allow the environment to take advantage of the added capacity. The flow just described is depicted in Figure 5. Similar rules to process the proxy server performance data are also defined.

The availability policies use a "heartbeat" mechanism. Several heartbeats (i.e., connection attempts) must be missed before a server is considered unreachable, and thus the main function of the availability rules is to collect missed heartbeat events until enough have occurred to be considered significant. Heartbeat rules are temporarily suspended for the server in question after a server has been identified as unreachable.

Metering policies use data collected using both "pull" (polling) and "push" models. For example, processing delay data are pushed from the game servers, while the current number of machines allocated to each environment instance is pulled from the management server. In general, metering data are summarized and stored. Any threshold evaluation is done off-line by the various systems that use metering data as input, such as billing.

Table 3 Gaming utility policies

SP policies	• Arbitration • Reservation
RM policies	• Capacity planning
ODS offering policies	• Environment configuration requirements • Generic performance and availability • Metering
ODS environment instance policies	• Specific game environment thresholds • Custom metering data

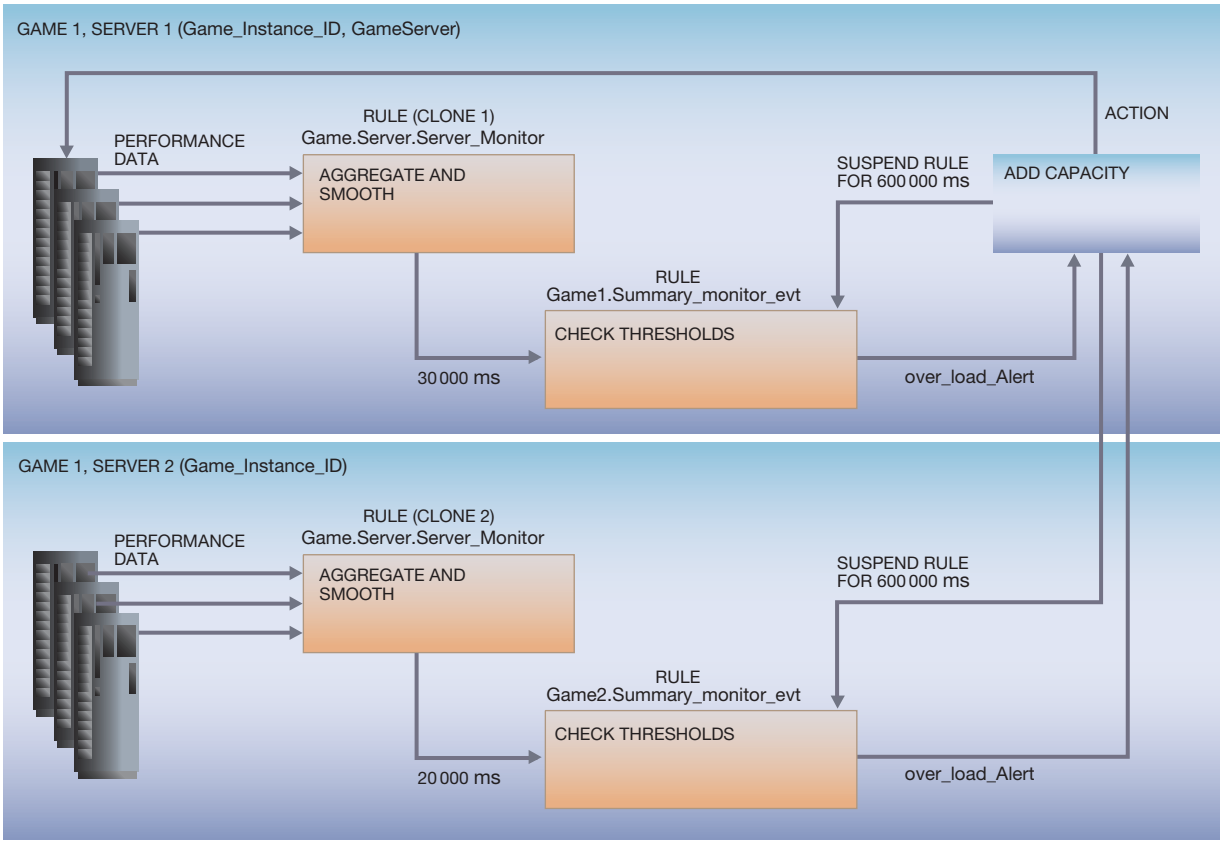
ODS environment instance policies. Policies specific to the game instance allow for custom behavior. At the very least, the threshold values used in the generic policies are specific to the instance. Threshold values can be incorporated into the copy of the generic policy for a particular UCSE or looked up dynamically when the rule is executed.

Policy creation and deployment. This subsection describes how policies are created and how they are deployed within the example system. There are two sources of policy information: the SLA document associated with a particular offering and the administrator console for the service provider. In Figure 6 the ODS game environment offering is portrayed from a policy creation and deployment point of view. The UCSEs in this case, as described above, are game instances that consist of a collection of proxy servers, a collection of application servers, and a number of shared infrastructure components. Policy distribution consists of the distribution of the policy specifications to the policy repository, the distribution of any subscriptions that are not implicit to the policy enforcement point, and the distribution of the metric collection agents to the resources themselves.

Policies in the repository. Policies can be defined and stored in the repository before any instance that uses them is created. They are designated inactive until there is an association with an appropriate instance. Policies are created and stored in the repository during several of the environment creation phases, at the time of creation for the utility infrastructure, ODS template, subscription, or the ODS itself, and when any component is updated.

When the utility is created, the initial set of SP policies (described in the section "Basic policies") are defined and stored in the policy repository. When

Figure 5 Game server performance policies



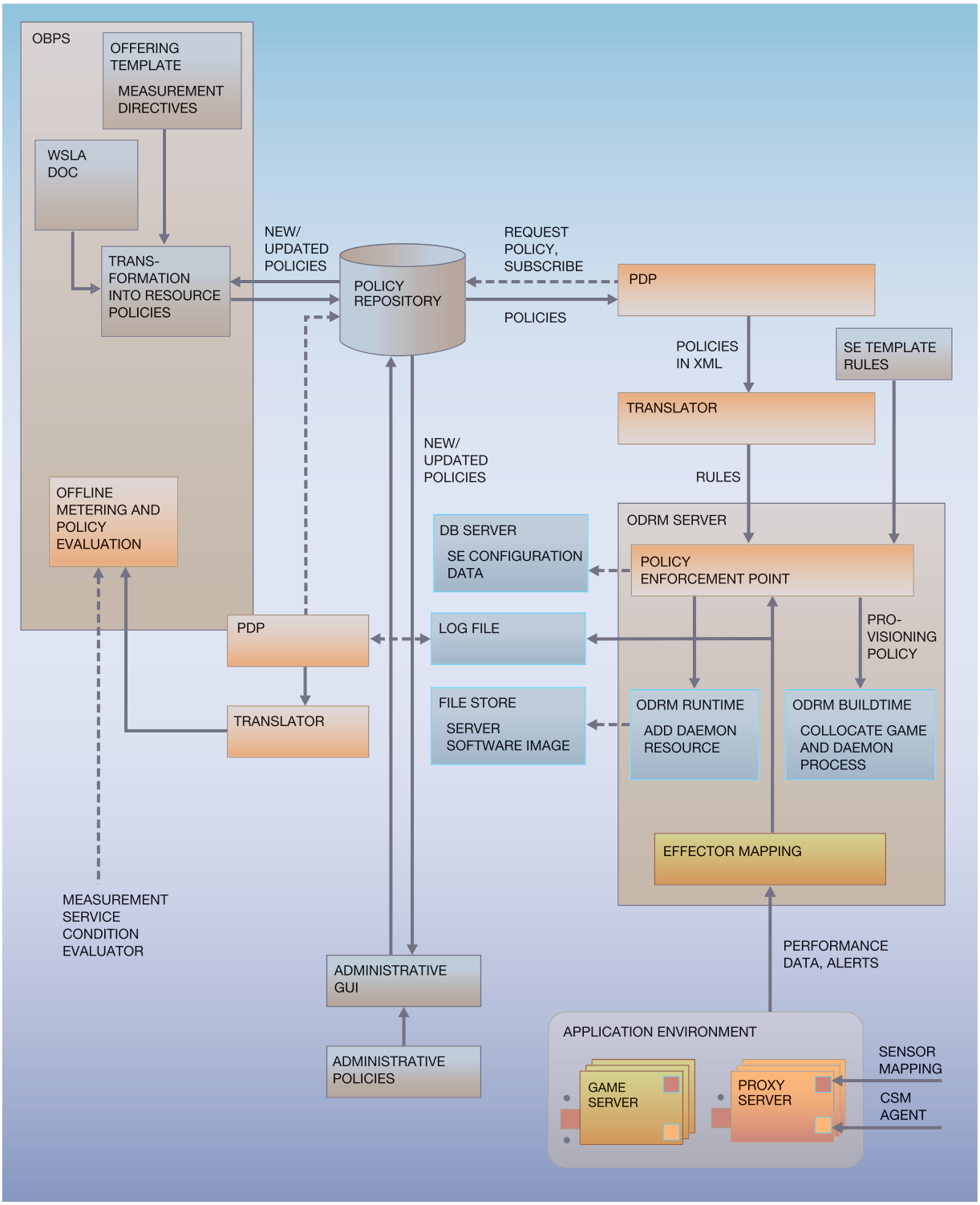
the ODS template is created, generic policies (policies that apply to all UCSE instances) and policy templates (policies with variables that will be set at subscription time) are defined. At subscription time, additional policies that are specific to the instance are created, and the template policies that need to be fully instantiated are completed. At UCSE creation time, additional policies may be generated by the build workflow.

Policy subscriptions. Utilities may have multiple PEPs. This may be desirable because of necessary domain-specific processing or for scalability reasons. In the currently planned deployment, we have three PEPs per ODS offering. A *runtime* PEP monitors ODS environment performance and availability. One *off-line* PEP enforces billing and penalty policies, and a second one evaluates configuration constraints when resources are allocated. In this case, the PEPs can subscribe to policies based on their domain because there is only one PEP per domain. The PEPs and their subscriptions are created when the ODS offering is defined.

In the future, it is likely that the enforcement of the runtime policies will be shared by multiple PEPs. For example, we may have one PEP per UCSE instance. In this case, a new PEP will be instantiated and supplied with the appropriate subscriptions each time a new UCSE instance is created.

Metric collection agents. Whether an agent uses a push or pull model, the PEP must be configured to receive its data. The PEP registers with the utility and indicates the ODS environment instances for which it has responsibility. When an instance is created, the PEPs are told which resources have been assigned to the instance, and the PEPs are provided with a reference to the resource collections owned by the instance. Resource collections can have associated pull

Figure 6 Policy management architecture



data, push data, or both. For collections requiring metric polling, a metric polling process is created.

Conclusions

In this paper, we have presented architectural extensions to the utility computing infrastructure that enable the use of policy-based computing technologies. We have also identified classes of policies that are useful within a shared resource infrastructure and identified methodologies for creating, deriving, deploying, and enforcing these policies. A hypothetical instantiation of a utility for multiplayer gaming has been used as an example to help make the discussion more concrete, but the overall mechanisms for policy enablement have much broader use.

We have shown that various types of policies are required within these complex systems. Policies will have to be defined at various times and by various means. This is because these policies cover different disciplines (performance, availability, configuration, etc.); also, it is due to the characteristics of the entities that the policies support.

There are many outstanding research problems in this area. Some ways of capturing the interrelationships between policies and automatically deriving lower-level policies from higher-level policies or business objectives have been identified. The simplest example would be to correlate policies that reference the same parameters or resources. In other cases, the triggering of one policy can generate an additional condition used by other policies. For example, the triggering of a policy to handle a hardware failure may generate a repair-request condition, which supplies mean and maximum repair-time statistical parameters that are used by a second policy to schedule repairs.

These processes need to be applied, studied, and formalized. Detection and resolution of policy conflicts, especially those that occur between sets of policies defined for different disciplines, has only been attempted in very simple cases. Methods for representing policies in such a way as to make it easier for administrators to understand their interrelationships would also be quite useful. This entire field of study is still not well understood.

Acknowledgments

The authors would like to acknowledge the contributions, advice, and support of many co-workers that helped evolve the concepts described in this paper. Special thanks are due to: Guerney Hunt, Tamar Eilam, Sandra Miller, and the utility computing team at IBM Research for their efforts on the general infrastructure, John Pershing for his contributions to the gaming architecture work, Tan Lu for his architecture and design work and his efforts in bringing all the pieces together, and Dinesh Verma and the policy team at IBM Research for their work on policy-based management.

Cited references

1. T. Eilam, K. Appleby, J. Breh, G. Breiter, H. Daur, S. A. Fakhouri, G. D. H. Hunt, T. Lu, L. B. Mummert, S. D. Miller, J. A. Pershing, and H. Wagner, "Using a Computing Utility Framework to Develop Utility Services," *IBM Systems Journal* **43,** No. 1, 97–120 (2004, this issue).
2. D. Verma, S. Sahu, S. Calo, M. Beigi, and I. Chang, "A Policy Service for GRID Computing," *Grid Computing - GRID 2002, Proceedings of the Third International Workshop*, Baltimore, MD; Springer, Lecture Notes in Computer Science, Vol. 2356 (November 2002).
3. B. Moore, J. Ellesson, and J. Strassner, *Policy Core Information Model, Version 1 Specification*, Internet Engineering Task Force, IETF RFC 3060, ftp://ftp.rfc-editor.org/in-notes/rfc3060.txt (February 2001).
4. *The IETF Policy Framework Working Group Charter*, Internet Engineering Task Force, http://www.ietf.org/html.charters/policy-charter.html (2002).
5. D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, *The (COPS(Common Open Policy Service) Protocol*, Internet Engineering Task Force, IETF RFC 2748, http://www.ietf.org/rfc/rfc2748.txt (January 2000).
6. W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, Addison-Wesley Longman, Inc., Reading, MA (1999).
7. M. Wahl, T. Howes, S. Kille, *Lightweight Directory Access Protocol (v3)*, Internet Engineering Task Force, IETF RFC 2251, http://www.ietf.org/rfc/rfc2251.txt (December 1997).
8. T. A. Howes, M. C. Smith, and G. S. Good, *Understanding and Deploying LDAP Directory Services*, Macmillan Network Architecture and Development, Macmillan Technical Publishing (1999).
9. *The DMTF CIM Schema v2.8 (Preliminary)*, Distributed Management Task Force, http://www.dmtf.org/standards/cim/cim_schema_v28 (2003).

Accepted for publication September 18, 2003.

Karen Appleby *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (applebyk@us.ibm.com).* Karen Appleby is a member of the Network and Systems Management Department of the Watson Research Center in Hawthorne, New York. She holds an M.S. degree in computer science from New York University. Her main research interests are in the area of event correlation, problem determination, e-commerce, and policy-based management.

Seraphin Calo *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (scalo@us.ibm.com).* Dr. Calo is a research staff member at the Watson Research Center

in Hawthorne, New York. He received M.S., M.A., and Ph.D. degrees in electrical engineering from Princeton University. He has worked, published, and managed research projects in a number of technical areas, including queuing theory, data communication networks, multi-access protocols, expert systems, and complex systems management. He has been very active in international conferences, particularly in the systems management area. His current research interests include content distribution networks, distributed applications, service management, and policy-based computing.

James Giles *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (jamesgiles@us.ibm.com).*
Dr. Giles received a B.S. degree in mathematics and a B.S. degree in electrical engineering from the University of Evansville, Indiana in 1995, and M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign in 1997 and 2000. He is currently a research staff member at the Watson Research Center in Hawthorne, New York. His research interests include communication and computer networks, computer and network security, and policy management. Dr. Giles received a Best Paper Award from the Society for Computer Simulation in 2001.

Kang-Won Lee *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (kangwon@us.ibm.com).*
Dr. Lee received his B.S. and M.S. degrees in computer engineering from Seoul National University, and his Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 2000. From 1996 to 2000, he was a research assistant at the TIMELY Research Group at the University of Illinois. Since August 2000, he has been a research staff member at the Watson Research Center in Hawthorne, New York. His research interests include policy-based management of distributed computing systems, multimedia communication over the Internet, multicasting heterogeneous packet flows, Web proxy caching, and multimedia content distribution networks. Dr. Lee was the recipient of the Gear Outstanding Graduate Student Award at the Computer Science Department of the University of Illinois in 2000.