Automated Generation of Resource Configurations through Policies

Akhil Sahai, Sharad Singhal, Vijay Machiraju, Rajeev Joshi
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2004-55
March 29, 2004*

E-mail: {akhil.sahai, sharad.singhal, vijay.machiraju}@hp.com, rajeev.joshi@jpl.nasa.gov

policy, resource, utility computing, configuration, workflow, deployment, management, deployment

Resource Management systems have been attempting to undertake automated configuration management. Automated configuration management involves considering user requirements, operator constraints and technical constraints of the system to create a suitable configuration, and to create a workflow to deploy it. In this article we propose a policy-based model that we have used for automating these configuration management aspects.

* Internal Accession Date Only
A part of this technical report will be published as a short paper in IEEE Policy 2004, 2-3 June 2004, Yorktown Heights, NY Approved for External Publication

Automated Generation of Resource Configurations through Policies

Akhil Sahai, Sharad Singhal, Vijay Machiraju
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94034
Rajeev Joshi[1]
Jet Propulsion Laboratories
4800 Oak Grove Drive, Pasadena, California 91109
`{akhil.sahai sharad.singhal vijay.machiraju}@hp.com`
`rajeev.joshi@jpl.nasa.gov`

Abstract: Resource Management systems have been attempting to undertake automated configuration management. Automated configuration management involves considering user requirements, operator constraints and technical constraints of the system to create a suitable configuration, and to create a workflow to deploy it. In this article we propose a policy-based model that we have used for automating these configuration management aspects.

Keywords: policy, resource utility, utility computing, configuration, workflow, activity, management, automated, constraint-satisfaction.

Introduction

Resource management systems have been trying to create systems that provide automated provisioning, configuration, and lifecycle management of a wide variety of resources. The current trend in utility computing is a step towards creating such automated resource management systems. HP's Utility Data Center product [1], IBM's "on-demand" computing initiative [2], Sun's N1 vision [3], and Microsoft's DSI initiative [4], Grid initiative [5] are examples of this trend. However, the resources that are available to these resource managements systems are "raw" computing resources (e.g., servers, storage, network capacity) or simple clusters of machines. The user has to still manually install and configure applications, or rely upon a managed services provider to obtain pre-configured systems from service providers.

Because every user's needs are different, it is usually not possible to create custom environments for every user—managed service providers rely on a small set of pre-built (and tested) application environments to meet each user's needs. However, this limits the ability of users to ask for applications and resources that have been specially configured to meet their needs. In our research, we are focusing on how complex application environments (e.g., an e-commerce site) can be automatically "built-to-order" for users. In order to create a custom solution that satisfies user requirements many different considerations have to be taken into account. Typically, the underlying resources have technical constraints that need to be met in order for valid operations, e.g., not all operating systems will run on all processors, and not all application servers will work with all

[1] Work done while author was at Hewlett Packard Laboratories.

databases. In addition, system operators may impose constraints on how they desire such compositions to be created. For example, when resources are limited, only certain users may be able to request them. Finally, the users themselves have requirements on how they want the system to behave. Thus, automating the design, deployment and configuration of such complex environments is a hard problem.

In this paper we describe a model for generating specifications for such environments based on policies. Policies have been traditionally described as rules that change the behavior of a system [18] and policy based management has been viewed as an administrative approach to simplify management by associating certain conditions with actions.

In our model for resource composition, the complex environments themselves are treated as higher-level resources that are composed from other resources. Policy is embedded in the various resource types, specified by the operators of the resource pool, or by users as part of the requests for resources, and restricts the composition choices used when composing higher-level resources from the component resources. Unlike traditional policy systems, our policy model does not couple actions to constraints, and actions (workflows) needed for realizing the specification of the higher level resource are automatically generated. By guiding the composition using policy, our model offers the following advantages over other methods of resource composition:

Component specification is easier than traditional approaches. Policy can be specified in a distributed and hierarchical manner by specifying the behavior of individual entities in the system. The designer only needs to specify constraints (as policy) that relate locally to the component(s) of interest, without worrying about global conflicts during composition. All policies that are relevant are automatically combined to ensure that the system conforms to the relevant constraints.

The system designer no longer has to be concerned with *how* a given composition can be realized. Because configuration workflows are also generated as generated as part of the specification, the designer only needs to specify the configuration actions available on individual entities in the system.

Updating components becomes easy. If a particular constraint on a component is modified, a new set of attribute values are computed by the policy system that would satisfy all policy constraints in the system, as well as the new configuration workflow that is needed to realize the system.

Adding or updating new entities or components is simplified. Since policy may be attached to any entity, new (or updated) entity instances and entity types can be introduced freely with their associated policies. These new policy instances are automatically considered in the policy management system when the new or updated entities are used.

In the next section of the paper we describe the policy-based model, which is followed by its application to resource composition in the subsequent section. This is turn is followed by a section on application of the policy model for automating workflow creation.

Policy-based Model for Specifying Resource Composition

When resources are combined to form other higher-level resources, a variety of rules need to be followed. For example, when operating systems are loaded on a host, it is necessary to validate that the processor architecture assumed in the operating system is indeed the architecture on the host. Similarly, when an application tier is composed from a group of servers, it may be necessary to ensure that all network interfaces are configured to be on the same subnet, or that the same version of the application is loaded on all machines in the tier. To ensure correct behavior of a reasonably complex application, several thousand such rules may be necessary if the construction of such applications is to be automated. This is further complicated by the fact that a large fraction of these rules are not inherent to the resources, but depend on preferences (policies) provided by the system operator or indeed, by the customer as part of the request itself.

In this section, we propose a policy-based model for combining resources which allows specification of such rules in a distributed manner. By capturing the construction rules as part of the specification of resource types, and by formalizing how these rules are combined when resources are composed from other resources, we provide a very flexible policy-based model for generating configuration specifications for complex resources.

In our model, we visualize policy as the entire set of strict (enforced) constraints that restrict allowable configurations of some target entity to those that satisfy some goal. Policies are therefore formulated as constraints on system composition (as opposed to conditions that arise as a result of system operation). In our model, resources and configuration activities are considered as the target entities. Each entity is characterized by a set of attributes and values taken by those attributes. For resource entities, the attributes represent configuration or other parameters of the resource that are meaningful for resource composition. For configuration activities, attributes represent if a particular activity needs to be triggered by the deployment system, and if so, parameters that are required for that activity.

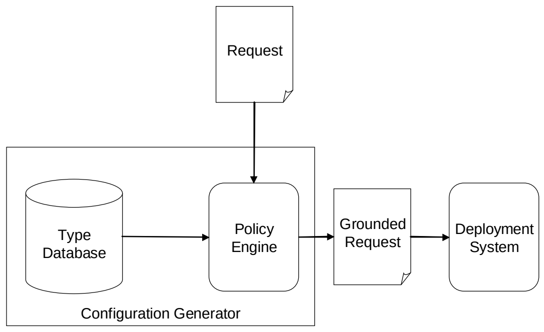


Figure 1: Resource construction process

Figure 1 shows the high level structure of the policy based configuration generator. The user creates a request (which may be minimally specific) for a composed resource. The configuration generator uses a type database and depending on the policies specified in the resource request and those associated with the resource types, generates a "grounded" request specification (i.e., a specification that is provably compliant with policy). The grounded request contains enough detail to allow a deployment system [20] to instantiate the request. The policy engine treats the user's request and the corresponding policy constraints as a goal to be achieved. It uses a constraint satisfaction engine [8] to select resource types and configuration activities, and assigns attribute values such that all of the policy constraints are satisfied.

Figure 2 shows the meta-model for construction policy. Construction policy is associated with both resources and activities that perform configuration operations on those resources. As part of creating the configuration specification, instances of resource types and activities are selected by the configuration generator such that the resulting model conforms to policy. The deployment system then uses that specification to initiate the appropriate activities to configure the resources to that specification.

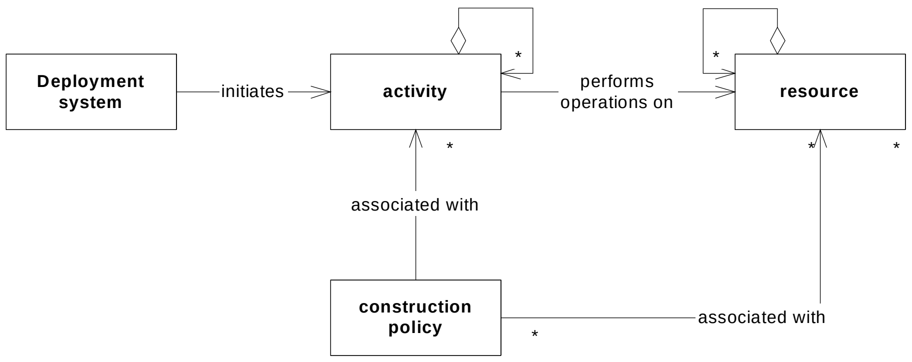


Figure 2: Relationship between policy, actors, activities, and resources

Construction policy is modeled as shown in Figure 3. Construction policy instances contain constraints that are defined using the attributes present in the associated resource type and activity type definitions. When a resource request is grounded, the configuration generator ensures that all policy constraints specified for that resource are satisfied. Because resource types can be derived from other resource types, this implies that all constraints for all composing resources are also satisfied. As mentioned above, resource attributes may contain any information meaningful for configuration. Activity models contain attributes that describe if a particular activity needs to be triggered during deployment. They may also refer to attributes of the associated resources or other associated activities. By capturing dependencies between the activities in the policy specification, workflows or methods may be modeled as composite activities. Since the configuration generator creates the union of all (relevant) constraints when creating the resource specification, it can also accommodate a variety of operator and user level policies during grounding.

Note that because the policy manager assigns values to all the attributes such that all the constraints are satisfied, explicit condition-action pairs are not needed in construction policy. Thus, the model allows complex configurations to be built without requiring the user or the operator to pre-specify which combinations and activities are valid for the overall composition and/or having to explicitly specify how such compositions can be achieved. We discuss this further below.

The configuration specification of a resource is defined to be *in compliance* with the construction policy if *all* policy constraints that refer to the attributes of the corresponding resource type are satisfied by the specification. Conversely, a specification is defined to be *in violation* of construction policy if *any* constraint that refers to an attribute of the corresponding resource type is violated by the specification. Similarly, activities are defined to be in compliance with construction policy if they are initiated when selected by construction policy, and if all pre-conditions for their initiation are satisfied. Conversely, activities are considered in violation of construction policy if they are initiated without being selected by policy, and if any pre-condition for initiation is not satisfied.

Constraints form the core of the policy specification and are defined using expressions that use policy attributes as variables. The language that we use for describing policies is derived from the SmartFrog language [20]. Constraints contain first order predicates, arithmetic and logical operators, and other structural constructs defined below:

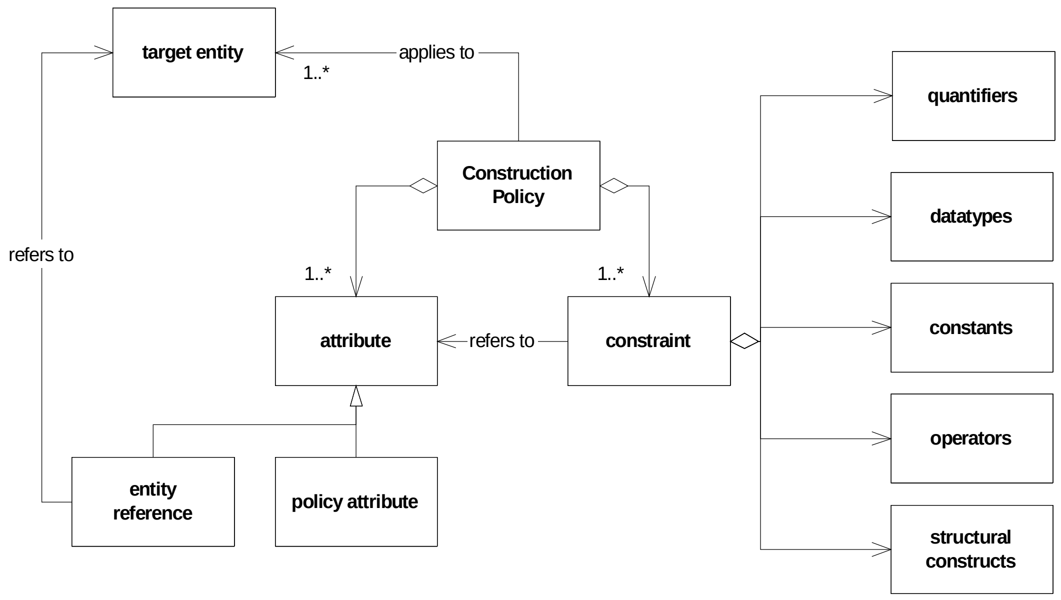


Figure 3: Elements of construction policy and its relationship with associated entities

Data types: Data types may be imposed on attributes as constraints that have to be satisfied by the corresponding attribute, e.g. constraints can specify if a particular attribute should be a String, integer, float etc. This allows validation of data types when different underlying components are used to provide similar functionality.

Constants: Numeric or string constants may used in constraints for defining the values or thresholds for attribute values.

Quantifiers: Quantifiers are often used in constraints, e.g. ∀•(for all), ∃ (there exists), etc.

Operators: A number of operators can be used to combine attributes in defining constraints. These operators fall in the following categories:

- *Arithmetic operators* (+, -, *, /): These operators can be used for constructing arithmetic expressions on literals of the allowed data types.
- *Comparison operators* (<, >, <= , >=, ==, !=): Comparison operators can be used to compare other expressions, and result in a boolean value.

- *Boolean Operators* (&&, ||, ! (unary not)): Provide logical expressions in constraints.
- *Implication Operators* (==> (logical implication), <== (reverse implication), <=> (equivalence, or if-and-only-if)): These operators allow expression of dependencies between attributes, e.,g. (name == Solaris) ==> version \in {5.7,5.8};
- *The instanceOf Operator*: The <: operator is used to denote "an instance of" relationship. This allows constraints to be created that enforce data types on components or their attributes, e.g., // ensure that component "server" is an instance of type Appserver
  server <: AppServer;
- *Set Operator*: \in operator may be used to constrain values of an attribute to be always in a set.

Structural Constructs: Other structural constructs (e.g., let in, if then else etc.) are used mostly for syntactic convenience. These familiar programming constructs simplify the task of the constraint writer when complex constraints have to be expressed in policy.

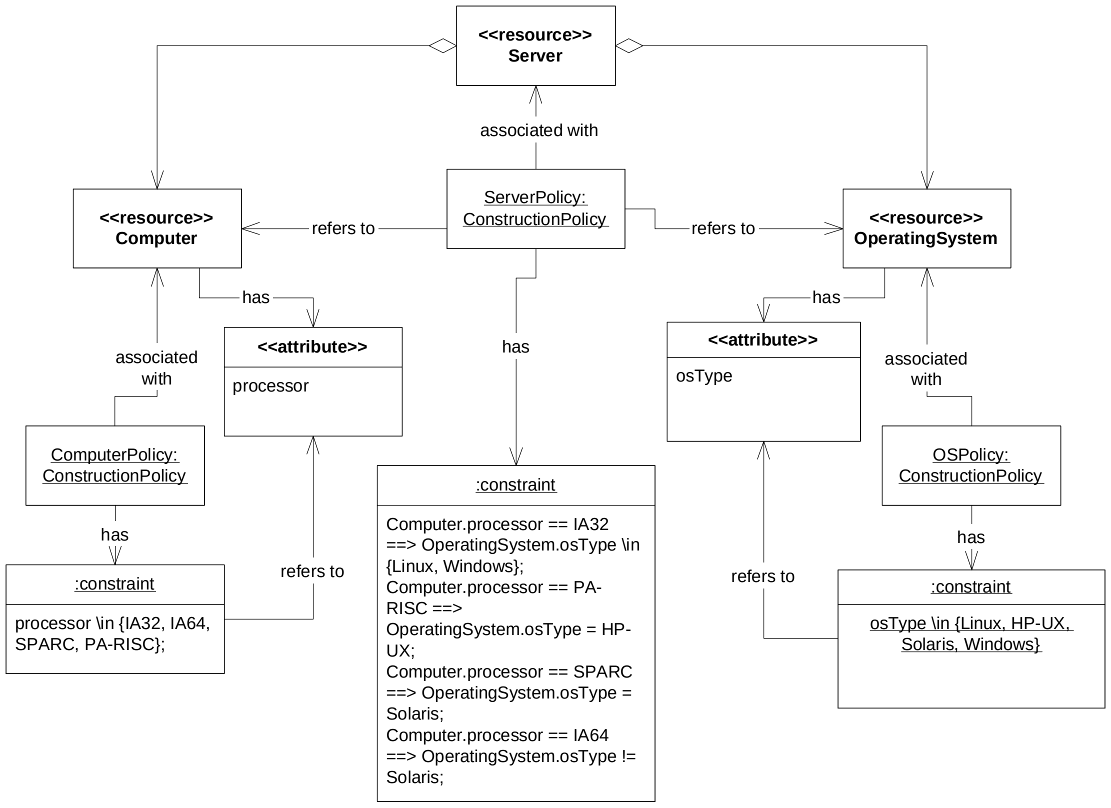
Applying Policies for creating Resource Configuration

When composing higher-level resources from other resources (e.g., an e-commerce site from servers), two issues need to be addressed: First, the configuration generator must select the composing resources such that composition is valid, i.e., the various conditions specified at different levels are met. Second, it must use the dependencies in the composition to generate a workflow that can be used by a deployment system to instantiate the composed resource.

Resource Composition and Component Selection

In our model not only do policies enable composition they also enable component selection, in cases where a number of components satisfy the requirements. Since models for composition apply to all types of resources, they are typically general. However, not all possible combinations of resources instances allowed by the model are valid. Policies attached to resource types ensure that the resulting construction is valid. We illustrate this with an example. Let us assume that we are trying to create servers. The resource model simply defines a server as a computer system with an operating system on it. A `Server` resource is thus composed from the `Computer` resource and the `OperatingSystem` resource in the model. However, not all computer types may be available in the resource pool, and not all operating system images would work on a given computer. Thus we need to define constraints that identify which computer and operating system image combinations are valid for constructing a `Server`. While this can be done within the model itself, it makes the model very complex as the number of possible combinations increases. Instead, we capture these constraints in a policy, and attach it to the corresponding type as shown in Figure 4.

In the Figure, the `Computer` has an attribute `processor` while the `OperatingSystem` has an attribute called `osType`. A policy associated with the `Computer` type states that the attribute `processor` can only take values in the set {IA32, IA64, SPARC, and PA-RISC}. Note that this constraint is specified by the system operator (perhaps because only these instances are available in the resource pool). Similarly, the construction policy associated with the `OperatingSystem` resource type states that the attribute `osType` can only take values in the set {Linux, HP-UX, Solaris, and Windows}. Again, the set is defined by the operator based on available operating systems. Now a policy associated with the `Server` can specify which `osType` can exist with which `processor` In order to create a valid instance of `Server`.



<<resource>> Server
associated with
ServerPolicy: ConstructionPolicy
refers to
refers to
<<resource>> Computer
<<resource>> OperatingSystem
has
has
has
<<attribute>> processor
<<attribute>> osType
associated with
associated with
ComputerPolicy: ConstructionPolicy
OSPolicy: ConstructionPolicy
has
has
:constraint
processor \in {IA32, IA64, SPARC, PA-RISC};
refers to
:constraint
Computer.processor == IA32 ==> OperatingSystem.osType \in {Linux, Windows};
Computer.processor == PA-RISC ==> OperatingSystem.osType = HP-UX;
Computer.processor == SPARC ==> OperatingSystem.osType = Solaris;
Computer.processor == IA64 ==> OperatingSystem.osType != Solaris;
refers to
:constraint
osType \in {Linux, HP-UX, Solaris, Windows}

Figure 4: Example of a composition policy

Suppose a request specifies that it needs a resource of type `Server` with the additional constraint that `Server.Computer.processor` = PA-RISC. From the constraints specified as part of the `Server`, the policy engine determines that the only valid value for `OperatingSystem.osType` = HP-UX, and automatically fills that value.

This example shows a number of aspects of constraint-based construction policies:

- By associating policy constraints with the individual component types, construction using those types can be controlled. By changing the allowed policy constraints, valid (or available) configurations can be maintained by the operators, and easily changed as needs change without extensive code or model modifications about how the new types are handled.
- Policy constraints for a resource type depend only on the attributes of that type, or the attributes of the underlying resource types. This simplifies hierarchical specification of types, because such dependencies can usually be localized in the type hierarchy. The designer of a new type only has to deal with the preceding types it is using and the corresponding constraints on them.

The policy engine automatically accounts for other dependencies that are created through transitional relationships.

- The policy engine checks all constraints for validity when handling resource requests. Because it can locate constraints that are being violated, the request specification can be checked for "correctness" with respect to those constraints. Similarly, during the instantiation of new resource types, the policy engine can aid the designer by validating that at least one valid instance of the new type can be created.
- The system can also fill in attribute values based on correctness of constraints. This means that the requestor has the freedom to only specify the attributes that are meaningful, and let the system fill in the gaps. This simplifies the requests.
- The requestor can add additional constraints for the policy engine as part of the request. Because the policy engine forms the union of all constraints when constructing the system, request-specific policy can be easily incorporated during construction.

Many additional constraints can be added to this simple example to account for items such as licenses, software versions, or other attributes such as memory and CPU speed. In addition, higher level resources can be constructed in a similar manner. Additional examples of resource composition and component selection are available in [26].

Creating Workflows for Automated Deployment

The deployment system has to execute a number of configuration activities to instantiate the composed resource. However, these activities cannot be executed in any arbitrary order. Just as the resources cannot be pre-composed (the composition depends on user requirements), the configuration parameters and the order of configuration cannot be pre-determined and provided to the deployment system. Depending on the exact composition, components may need to be configured differently and may need different work flows for configuration. Thus, for example, the configuration activities associated with an application server may change if the selected database server is different. Furthermore, depending on the composition, activities may need to be performed in different order. Some activities could be executed in parallel while others may need to wait for others to complete before proceeding.

Our Activity model is loosely based on the notion of task-graphs as implemented in Microsoft® Project. In our Activity model, configuration activities are modeled similar to resources, and are associated with the resources in the composition hierarchy. As the policy engine selects resources appropriate for a given request, it also selects the corresponding configuration activities. The Activity model is shown in Figure 3. An activity has set of attributes that determine when the activity will be performed. It has a duration, a startdate, enddate, and a mechanism to specify whether there is a deadline associated with the activity. It has a constraintDate and a constraintType that determines when the activity has to be executed. The constraintType could be either, As early As Possible (ASAP), As Late As Possible (ALAP), Finish No Earlier Than (FNET), Finish No Later Than (FNLT), Must Finish On (MFO), Must Start On (MSO), Start No Earlier Than (SNET), Start No Later Than (SNLT).

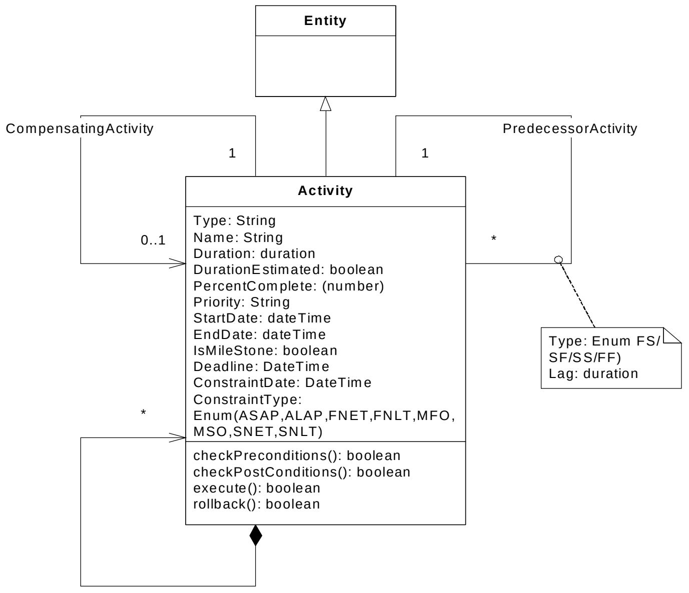


Figure 5: Activity Model

An Activity is made up of other activities. An activity may have a compensating activity and every activity may have a set of predecessor activities. Policies are associated with Activities and so the *pre-conditions* and *post-conditions* of an activity may be specified as policies. In our model, these pre-conditions and post-conditions may also be used to create a sequence of activities in a workflow. We have formalized the precedence through the type attribute defined in the association between an activity and its predecessor activities, which determines the order in which the activities are executed. These temporal planning based constructs enable creation of workflows.

- FS type means the predecessor activity is finished before the successor activity is started (sequence).
- SF means that the predecessor activity is started before the successor activity is finished
- SS means both the activities are started at the same time (parallel). Lag if present determines how much time after the predecessor activity is the successor activity started
- FF means both the activities must finish together (synchronize).

Policy constraints associated with each resource specify the associations between activities and predecessor activities of itself and its components. These predecessor activities have to follow the precedence relationships mentioned above with the successor activities. As a result of the constraint satisfaction a set of components are chosen along with a set of activities. These associations between the so chosen activities automatically establish ordering between the selected activities. A post-processor looks through all the enabled activities and using precedence relationships between the activities, creates a workflow. Figure 6 shows a part of a resource model for an e-commerce site with the associated activities. Note that where refinements exist (e.g.Oracle9i, IBMDB2 for a database) for resources, separate configuration activities may be associated with the refinements. The complete directed acyclic graph establishes the relationships between activities and their immediate predecessors.

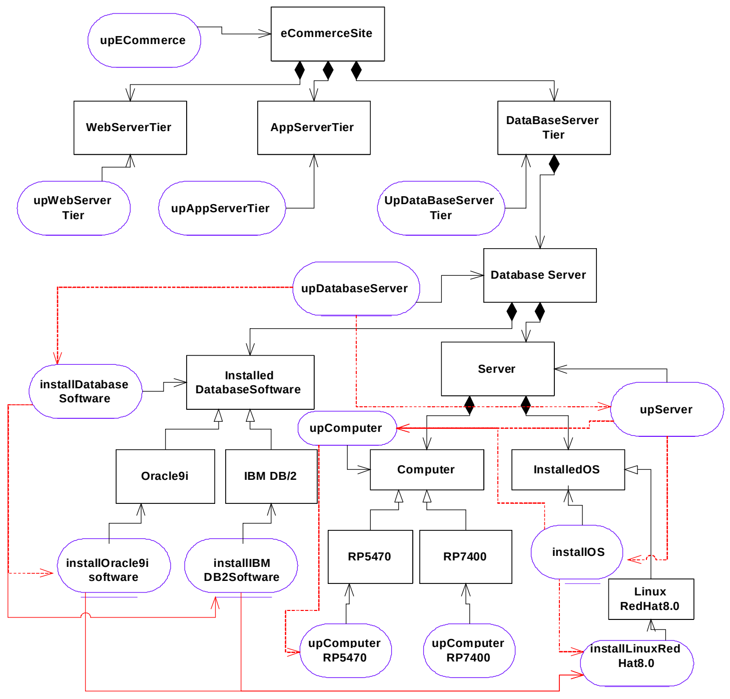


Figure 6: A resource model with associated activities

Figure 7 shows a part of the workflow generated when the model was used to instantiate an instance of an `eCommerceSite`. Note that the workflow contains both sequential and parallel activities as well as points where different activities need to be joined. Also note that it is not always possible to infer the sequence of activities from the composition hierarchy alone (e.g., the composition hierarchy does not show that the server has to be up before database software can be installed on it in order to instantiate the database server. Such sequences are determined from the dependencies specified as part of the activity model.

Related work

The ClassAds MatchMaking work [14] assumes that the match-maker matches the requestor entity's request against the provider entity's ClassAds (which are specifications in a semi-structured language). The assumption is that all the resources (like machines) exist a-priori and have been advertised. In a resource-utility environment however, some of the resource instances may not even exist a-priori (as is the case with transient/virtual resources) or may be logically constructed resources that have to be instantiated on-demand (e.g. appserver/tier/farm/e-commerce site). This causes a problem for approaches that undertake match-making only on instances. We enable construction on-the-fly by embedding constraints hierarchically in the *resource types* as described in this paper. The same concepts are extensible to resource instances as well. It is also not clear whether the ClassAds language supports first-order logic and linear arithmetic. As we have shown in the examples, it is important to have notions of quantifiers, implications, equivalences and other first order-logic expressions for reasoning.

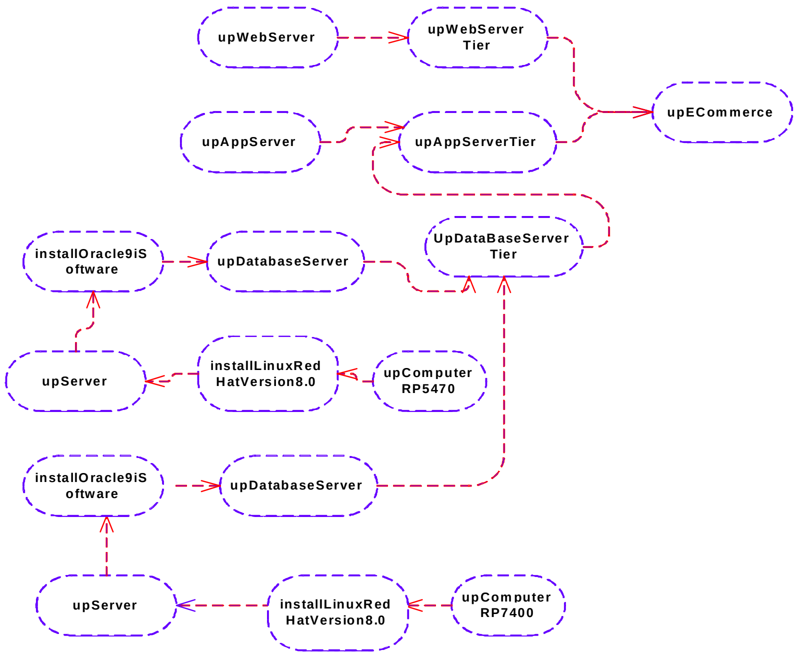


Figure 7: Part of the workflow generated when the e-commerce site is composed

In the context of VLSI CAD design [25], Constraint satisfaction technique has been used to create automated workflows. The author claims to have enhanced designer productivity in the process. We have used a similar approach for creating automated workflows for resource configuration in our work. The Workflow Management Coalition (WfMC) [23], The Business Process Execution Language (BPEL4WS)[24], and other languages have been attempts at defining the language in which the business processes and workflows may be described. In our work we have focused on generating workflows on the fly that may be handed to deployment systems to install complex resources. The specification that is generated as a result of our approach may be specified in the above-mentioned languages.

There is lot of work that has been done in the community in terms of specifying, and associating events, conditions and actions for policies, namely IETF [15], CIM [10], PARLAY[17], PONDER [7] etc. Additional work relates to using policy for SLA management [22]. These bodies of work, to the best of our knowledge, have not looked at incorporating first-order logic and linear arithmetic based constraints in resource types for automatic constructions of resources and have not used a constraint satisfaction approach for arriving at a complete deployment specification involving resource composition and workflows. The WS-Policy [19] work at OASIS has focused on generic schemas for specifying arbitrary policy assertions on web services. The constraints as specified in this article may be embedded inside these assertions.

Conclusion

In this paper, we have described a policy-based model that we used for resource composition and automated workflow generation. In our policy-based model, we embed policies into entities in a hierarchical manner. The user requirements, the technical constraints and operator preferences are captured as policies and a complex resource is composed satisfying all the requirements. In our policy-based model we also decoupled actions from constraints. This enabled us to create workflows for resource deployment in an automated manner.

References

1. HP Utility Data Center (UDC) http://www.hp.com/solutions1/infrastructure/solutions/utilitydata
2. IBM Autonomic Computing http://www.ibm.com/autonomic
3. SUN N1 http://wwws.sun.com/software/solutions/n1/
4. Microsoft DSI http://www.microsoft.com/management/
5. Global Grid Forum http://www.ggf.org
6. Platform LSF http://www.platform.com/products/LSF
7. Nicodemos Damianou, Narankar Dulay, Emil Lupu, Morris Sloman: The Ponder Policy Specification Language. POLICY 2001: 18-38
8. van Hentenryck, P. Constraint Satisfaction in Logic Programming, The MIT Press, Cambridge, Mass, 1989.
9. VMWare Virtual Machine http://www.vmware.com/
10. CIM Modeling http://www.dmtf.org/standards/standard_cim.php
11. DMTF: http://www.dmtf.org
12. Object Constraint Language (OCL) http://www.ibm.com/software/awdtools/library/standards/ocl.html
13. SNIA CIM Object Manager (CIMOM). http://www.opengroup.org/snia-cimom/
14. Raman R, Livny M, Solomon M. MatchMaking: Distributed Resource Management for High Throughput Computing. In the proceedings of HPDC 98.
15. IETF Policy. http://www.ietf.org/html.charters/policy-charter.html
16. DMTF-CIM Policy http://www.dmtf.org/standards/documents/CIM/CIM_Schema26/CIM_Policy26.pdf
17. PARLAY Policy Management http://www.parlay.org/specs
18. Moffet J, Sloman. Policy Conflict Analysis in Distributed Systems. In the proceedings of Journal of Organizational Computing,, 1993
19. OASIS WS-Policy WG. http://www.oasis-open.org
20. SmartFrog http://www-uk.hpl.hp.com/smartfrog/
21. Verma D, Beigi M, Jennings R. Policy based SLA Management in Enterprise Networks. Workshop on Policy, POLICY 2001.

22. Foster I, Kesselman C, Tuecke S. The Anatomy of Grid: Enabling Scalable Virtual Organizations. In the Proceedings of International Journal of Supercomputer Applications, 2001.
23. Workflow Management Coalition WfMC. http://www.wfmc.org
24. Business Process Execution Language for Web Services (BPEL4WS) http://www.ibm.com/developerworks/webservices/library/ws-bpel/
25. Shepelev V, Director S. Automatic Workflow Generation. Euro-DAC 96. European Design Automation Conference with EURO-VHDL'96. Sept 1996
26. Sahai A, Singhal S, Joshi R, Machiraju V. Automated Policy-Based Resource Construction in Utility Computing Environments. In the proceedings of NOMS 2004