An architecture for virtual solution composition and deployment in infrastructure clouds

Article · June 2009

DOI: 10.1145/1555336.1555339

Alexander V. Konstantinou
Google Inc.
25 PUBLICATIONS **331** CITATIONS

Michael Kalantar
IBM
19 PUBLICATIONS **566** CITATIONS

W.C. Arnold
IBM
11 PUBLICATIONS **489** CITATIONS

Ed Snible
Association for Computing Machinery
7 PUBLICATIONS **130** CITATIONS

An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds

Alexander V. Konstantinou, Tamar Eilam, Michael Kalantar,
Alexander A. Totok, William Arnold, Edward Snible
IBM Research
19 Skyline Dr
Hawthorne, NY, USA
{avk, eilamt, kalantar, barnold, snible}@us.ibm.com, aatotok@ieee.org

ABSTRACT

The combination of virtual server technology and the Infrastructure-as-a-Service (IaaS) approach to utility computing promises to revolutionize the way in which distributed software services are deployed. Server virtualization technology can be used to capture complete reusable software stacks, shifting the complexity of middleware installation and configuration from deployment to packaging. IaaS clouds provide a set of interfaces for controlling virtual machines and configuring their hardware and network environment, substantially reducing the complexity of service provisioning. In this paper we identify and tackle a few of the remaining challenges in fulfilling the promise of radical simplification of distributed software service composition and deployment. We propose an approach and architecture for composition and deployment of virtual software services in cloud environments. We introduce a virtual appliance model which treats virtual images as building blocks for composite solutions. Virtual appliances use a port abstraction to negotiate their communication parameters. A solution architect creates a virtual solution model by composing virtual appliances and defining requirements on the environment in a cloud-independent manner. The virtual solution model is transformed to a cloud-specific virtual solution deployment model used to generate a parameterized deployment plan that can be executed by an unskilled user. We validated our approach through a prototype implementation demonstrating flexible composition and automated deployment in our local lab virtualization infrastructure and in Amazon EC2.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software configuration management*

General Terms

Management

Keywords

Cloud computing, virtualization, model-driven deployment

1. INTRODUCTION

The deployment of a software service typically involves the installation of software artifacts, the installation and configuration of middleware containers such as web servers, application servers, databases, messaging engines and directories, and the configuration of the infrastructure for communication, isolation, and security. Server virtualization technology has provided a powerful solution to the challenge of assembling software stacks. Deployers can now utilize virtual images capturing complete and functional stacks that have been assembled by middleware experts. The emergence of the cloud computing paradigm supporting the provisioning of an Infrastructure-as-a-Service (IaaS) has similarly radically simplified the provisioning of hardware resources to run virtual images and the configuration of their placement and network connectivity. The combination of virtualization and IaaS therefore holds the potential to revolutionize software service life-cycle management. However, several key challenges are yet to be addressed to fulfill the promise:

Challenge 1: Solution Design. The structure of a multi-machine virtual software service depends on a large number of cross-cutting functional and non-functional concerns such as availability, security, performance, and cost. Solution architects must make architectural decisions on the use and type of application server clustering technology, authentication services, data storage, as well as the grouping of components into network zones, and the selection of deployment site. Leveraging pre-built virtual machines as solution components reduces the solution design space, but the number of possible combinations and concerns is still high, and their interplay complicated.

Challenge 2: Software Stack Cross-Configuration. The service architecture and topological structure dictate multiple cross-configurations of the virtual machines comprising the service. Application servers will have to be configured to use authentication servers, access databases, and be federated in clusters and cells. Cross-machine configurations cannot be statically captured ("freeze dried") on the combined set of virtual machines, since they will vary across solutions. Therefore, the needed configurations must be inferred and implemented at deployment time based on the solution design. Dependencies must be respected to consistently configure all components of a service. Often, the value of a property exposed by one component must be used to configure multiple other components. Parameterized values, satisfying all component constraints, must be selected consistently. Therefore configuring the dynamic parts of a solution is still complex and error prone.

Challenge 3: Cloud Environment Configuration. The cloud infrastructure, including the virtual hardware, network, and storage, must be configured for communication, security, availability and isolation based on the solution's requirements. The capabilities and interfaces for environment configuration vary significantly across clouds. Solution-level environment requirements must be translated to cloud-specific configurations and interface calls. Certain clouds may offer functionality too weak to meet the requirements of the service. Thus, the relationship between service requirements and cloud capabilities must be analyzed to identify the cloud most suitable for deployment.

Challenge 4: Automating the Deployment. Deploying a distributed software service requires the determination and execution of a non-trivial sequence of cloud interface calls to load and control images and to configure the cloud environment. The calls and their parameters are specific to each service and cloud environment configuration. Temporal dependencies must be taken into account to correctly implement the deployments.

In this paper we introduce a novel approach and architecture leveraging virtualization and IaaS for software service composition and deployment which addresses the above challenges. In our approach, we reduce the design space by leveraging coarse-grained image building blocks. We enable the cloud-agnostic modeling of solution environment requirements which can be transformed into cloud-specific configurations and be used to automatically generate a deployment plan. We have implemented a prototype of our approach where we compose and deploy distributed virtual software services in Amazon's EC2 [2] cloud, and in our own lab's Xen infrastructure.

This paper is structured as follows. In section 2, we describe our approach and architecture. In section 3, we discuss the concept of virtual appliances used as solution building blocks. In section 4, we detail the process of composing images and constraining their deployment in a cloud-independent manner. In section 5, we outline the process of transforming the solution model into a deployment model that is cloud-specific. In section 6, we discuss the transformation of the deployment model into an executable plan. In section 7, we describe our prototype. We conclude with a section on related work, and future work.

2. A MODEL-DRIVEN APPROACH

In our approach, depicted in figure 1, domain experts construct *virtual appliances* which are implemented by virtual images with pre-built configurability points for self and cross-configuration. The virtual appliance model exposes these configurability points declaratively in a manner which admits simple composition with other virtual appliance models. Solution architects create a declarative *virtual solution model* (VSM) by selecting virtual appliance models, and specifying configurability choices (based on the manifested options). Solution architects can further describe deployment requirements that will affect and drive virtual machine placement and network configuration. The requirements are specified in a cloud-independent manner using abstractions such as availability zones, network zones, placement constraints, and resource consumption constraints and requirements. Deployment architects consume a VSM and transform it to a cloud-specific *virtual solution deployment model* (VSDM) that can be further customized with additional cloud-specific configurations. Finally, the deployment model is transformed to a cloud-specific *virtual solution deployment plan* (VSDP). The plan can be executed by a normal user, such as a software tester. The VSDP operates in two layers: (1) it invokes the cloud operations to configure the environment and control (load, allocate, reserve, activate) the virtual machines, and (2) it exercises the virtual appliance built-in configuration logic to configure the software stack based on the VSM.

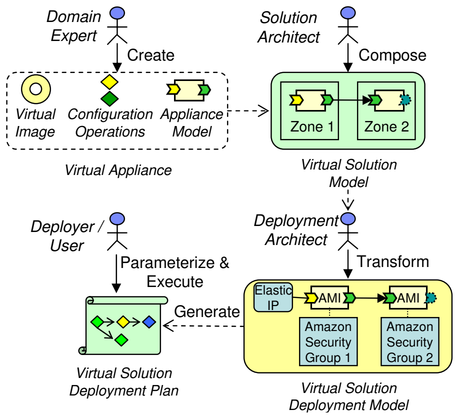

Figure 1: Virtual Solution Design and Deployment

Our approach is focused on a clear separation of roles to enable the scalable capturing of deployment logic across the rich space of possible software service solution topologies and layers. We leverage virtualization to assign the role of stack creation to the operating system, security, middleware, and scripting experts. The resulting virtual images are hardened and expose limited customization parameters at deploy time, such as host name selection. A cloud image expert will be responsible for uploading and registering each such image in clouds which can support it. Each such image self-describes the configurability choices through an appliance model that acts as an interface. Note that a single interface may be associated with multiple implementation choices, such as a VMware ESX image or an Amazon Machine image.

The complexity and risk faced by the solution architect is greatly reduced: instead of having to deal with fine grained installation and complex fine-grained middleware configuration choices, the solution architect operates within the reduced space of the options exposed by the available virtual appliances. This reduced space allows the solution architect to focus on key aspects (tiered architecture, technology selection, etc) rather than worry about fine-grained middleware configuration complexity, individual stack tuning options, and version compatibility issues. The VSM constructed by the solution architect is used to seamlessly calculate the activation parameters of the image and to ensure consistent composition with other images, with configuration parameters or properties of one image seamlessly propagated and used to cross-configure other dependent images.

The role of the deployer is then restricted to binding the VSM to one or more clouds and then generating and executing the deployment plan. We have implemented a prototype of this approach in a model-driven deployment platform we developed and that was recently released as part of IBM's RSA 7.5 product[18]. We have demonstrated this prototype in our local Xen-based virtualization lab, as well as in Amazon EC2, which we describe in section 7.

The technology that we have developed and is described in this paper can be offered as a service, termed *Composition-as-a-Service* on top of an IaaS platform. Such a service could be used by large organizations, interested in harnessing the power of cloud computing in order to reduce their infrastructure expenses (administration costs, hardware cost, etc). Typically, such large organizations have advanced IT needs that go beyond usage of single software stacks captured in individual virtual machines. Such large organizations typically develop and maintain thousands of in-house composite applications (e.g. J2EE). These applications are typically tested in a sequence of test environments (integration, stress, staging) before they can be rolled into production.

Leveraging the IaaS platform, enhanced with our composition service for application testing is a strong match: First, security requirements on testing environments are less stringent than production, thus, businesses will be inclined to move into the cloud for testing first. Second, complete test environments have to be temporarily provisioned only for the duration of the testing. Third, these environments need to be frequently and easily updated with new versions of the source code (a future piece of our work deals with the update of virtual appliances as part of a nightly build process). Fourth, different topologies will be needed based on the application structure and testing needs. Last, it is crucial to reduce the time to provision and configure the test environments (today 35% of testing time is spent on just configuring the test environment). Note that current Platform-as-a-Service (PaaS) offerings may be too restrictive for large organizations since they dictate specific technologies (e.g. the .Net model in Microsoft's Azure) or application programming model (e.g. Google App Engine). Thus, large organizations will be forced to work in the IaaS layer, and could benefit from a Composition-as-a-Service layer, that can greatly reduce deployment complexity without compromising flexibility.

3. VIRTUAL APPLIANCES

A virtual appliance consists of three inter-related parts: (1) a virtual image, (2) a virtual appliance model, and (3) a set of configuration operations. The virtual appliance model declaratively captures the ways in which the virtual image can be customized at deploy time. The configuration operations are the means by which the values configured in the appliance model are applied to the image. It is possible to use the same appliance model for multiple images with the same customization features. For appliance models corresponding to more than one image, the configuration operations may also be the same, or differ based on their compatibility with the images.

3.1 Virtual Appliance Model

The *virtual appliance model* declaratively captures deploy time parameters in the image's internal configuration, such as the root password, and configuration of its external connectivity to other images in the virtual solution or other external resources. Our virtual appliance model consists of two parts: packaging and composition. The *packaging* part

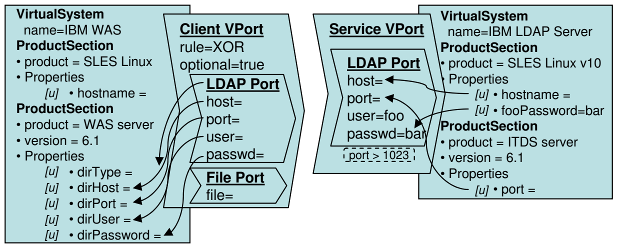

Figure 2: Virtual Appliance Model

contains meta-data about the virtual machine in a platform-independent manner. The packaging data includes product sections with the configuration attributes of different products installed on the image, such as the operating system, and identifies the ones which the user can modify at deploy time, such as the host name. The *composition* part specifies how the appliance can be connected to other appliances. We introduce the concept of a virtual port (*VPort*) to capture how the image can be linked. A VPort is a typed object which encapsulates all the properties required to effect a specific type of communication. The VPort properties often cross product sections and may overlap in their coverage with other VPorts on the same image.

We'll illustrate our modeling of virtual appliances with examples of an IBM Tivoli Directory Server (ITDS) virtual appliance (an LDAP server), and an IBM WebSphere Application Server (WAS) virtual appliance (a J2EE server). The ITDS appliance model is pictorially represented on the right side of figure 2. The packaging part exposes two product sections, for the operating system (OS) and the ITDS LDAP server. The OS section exposes two user configurable attributes: the host name of the image and the password of the "foo" user. The ITDS section exposes one user configurable attribute specifying the LDAP network port. The model for the WAS appliance is shown on the left side of the same figure. The WAS appliance also has two sections, one for the OS and another for the the WebSphere server. The section on the WAS server includes user settable attributes for optionally delegating configuration to a directory server.

We use VPorts to capture the composition parts of the ITDS and WAS appliances of figure 2. Each VPort captures all the properties required to effect a specific linkage independent of where they are defined in the product section. We have identified a number of reusable VPort patterns. For the specific example, a basic client-server communication pattern, we will model the ITDS server as a "Service VPort" and the WAS authentication client as a "Client VPort". The ITDS Service VPort contains a single LDAP VPort. The LDAP VPort exposes the host name and port number of the network service as well as the user name and password with which to authenticate using the LDAP protocol. The values for the host, port and password properties are propagated from the product sections, while the user name is hard-coded as it is not exposed as a product section attribute. In contrast, the WAS Client VPort contains two ports: an LDAP VPort and a File VPort. The Client VPort is marked as optional, to indicate that it need not be configured. Its containment semantics are declared as Boolean XOR, meaning that only one of the two contained ports may be connected. The LDAP VPort properties are propagated to the WAS product section parameters. The

```
<serviceVPort>
  <ldapAtomicPort host="" port="1099" user="foo" passwd="bar">
    <attributeMetaData attribute="user" mutable="false"/>
    <attributeMetaData attribute="passwd" mutable="false"/>
    <rangeConstraint attribute="port" minValue="1024"/>
  </ldapAtomicPort>
</serviceVPort>
<clientVPort containmentRule="XOR" optional="true"
             connected="false">
  <ldapAtomicPort host="" port="" user="" passwd=""/>
  <fileAtomicPort file=""/>
</clientVPort>
```

Figure 3: VPort XML Representation

`dirType` parameter is set based on the type of the connected contained VPort (LDAP or File). The XML representation of the VPorts in this example is listed in figure 3.

Service VPorts are used to model a server offering a shared access point that does not require per-client customization. A Service VPort contains a single *atomic* port, which is a typed object representing an individual non-divisible connectivity specification. Because Service VPorts represent a service to which multiple clients may connect, its properties must be consistently propagated to all clients. Where there is choice, such as the protocol to use, the model can expose multiple Service VPorts. *Client VPorts* are used to represent the client side in the client-server communication pattern that must be configured for communication with the server. A Client VPort is associated with a connection state: when disconnected, some of the properties may be blank, while when it is connected, its properties are expected to be set through a negotiation progress discussed in the next section. A client VPort may be "optional" if the virtual image can function with the VPort in a disconnected state. Client VPorts can have a nested port structure to allow modeling of complex port connection strategies. The `containmentRule` property specifies how the Client VPort connection state relates to the connection state of the contained *atomic* ports. When an individual atomic port is connected to an atomic port on the target Service VPort, their types must match and there should be a set of negotiated atomic port property values that satisfies the constraints on both sides. Due to space limitations, we defer a more detailed discussion and presentation of additional VPort patterns to a separate publication.

VPorts and their properties may be associated with constraints. For example, a constraint on the WAS server may state that the LDAP server to which it is connected be limited to a specific set of supported products. Another type of constraint may express requirements on the communication infrastructure between the two servers. For example, the ITDS port will require TCP communication over the specified port to the LDAP server. For ports which depend on link-layer dynamic configuration protocols, a constraint may require collocation of the two endpoints over the same LAN (or VLAN). The values of properties may also be constrained, for example if the LDAP server is running as a non-root user, a constraint may restrict user port selection to over 1023.

For the representation of the packaging part of our virtual appliance model we have adopted the Open Virtualization Format (OVF) specification[9]. We use the OVF *Envelope* document to capture the virtual appliance hardware attributes, such as memory, number of CPUs, I/O interfaces, disks, and network interfaces as well as the product sections. The OVF model is extensible, and therefore we can easily embed our VPort XML models, examples of which are shown in figure 3, as an additional element of the OVF envelope. As a result our combined packaging and composition virtual appliance model can be represented in a single XML document that is standard compliant (OVF explicitly supports extensions of the type we are proposing).

3.2 Configuration Operations

Every product section containing user configurable attributes must be associated with one or more *configuration operations* (COs) that will reconfigure the image. The set of operations does not only depend on the individual products whose configuration is exposed, but may also depend on the composition of the stack. For example, some middleware containers cache the host name of a server, and can break if a change at the OS level configuration is not accompanied by middleware-level configuration changes. The domain expert must capture such internal propagations in the process of associating configuration operations to product sections. There are three types of configuration operations based on the context of their execution: (1) *local COs* must be executed within the image, and thus necessitate its deployment, (2) *remote COs* can be executed on another host, but require the image to be reachable in order to perform remote management calls, and (3) *off-line COs* are executed on a separate host and operate on the image contents at the file-level. The mapping of CO parameters to virtual appliance model objects will be discussed in section 6 on planning.

The price for the flexibility afforded by an IaaS approach to cloud computing is that beyond a limited set of dynamic protocols, such as DHCP, the configuration of the image contents is the responsibility of the user. For local COs, IaaS clouds typically support the passing of opaque user data from the deployer to the image to enable deploy time customizations. The domain expert can embed a configuration operation engine in the image that will evaluate the data passed by the deployer (e.g. [17]). The user data will include an environment document with property section values, and may include COs that are not already present on the image. Local execution of COs greatly simplifies the process of image customization, and is the approach we've taken in our current prototype. In contrast, the execution of remote and off-line COs requires the availability of a workflow engine, and a set of management hosts.

4. VIRTUAL SOLUTION MODELING

A virtual solution model (VSM) is used to capture the composition of one or more virtual appliance models and the logical configuration of their deployment environment. The VSM is logical in the sense that the virtual appliance models can potentially be realized by images stored in different repositories or clouds. Moreover, the environment is defined using high-level concepts such as placement zones, network zones, and storage volumes. The logical environment concepts are cloud-independent and admit realization in more than one kind of an infrastructure cloud. The realization of some logical environment concepts may not be possible in all infrastructure clouds, and their requirement will limit the range of valid deployment options. In this section we will outline how the virtual appliance models are composed, and provide examples of logical cloud environment models.

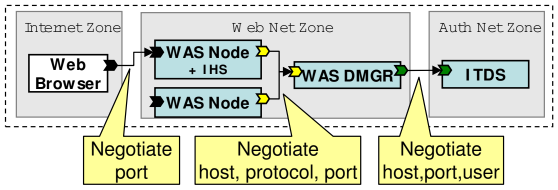

Figure 4: Virtual Solution Model

4.1 Appliance Composition

Consider the creation of a logical topology for deploying a Sun J2EE enterprise application which utilizes directory authentication. The application artifacts will include the application EAR archive, and an LDAP Data Interchange Format (LDIF) file to populate the directory server with application-specific schemas and users. Deployment of this application will therefore require a J2EE application server and an LDAP directory server. We present an example based on the virtual appliance models we described in the previous section. To develop a more interesting solution example, we will represent two types of WAS servers: an application server node, and a deployment manager node which can federate any number of nodes into a cell which can host an application cluster. In addition, we model a port for HTTP connectivity and represent an "external" resource that is independently deployed and associated with a read-only VPort model to represent an HTTP client accessing the cloud. An example of a VSM is illustrated in figure 4. In this topology the HTTP Client VPort has been connected to the HTTP Service VPort of a WAS Node which is configured to spray requests over the cluster it participates in. The cluster is configured via its Federate Client VPort connectivity to the deployment manager. Finally, the deployment manager is connected through its LDAP Client VPort to the ITDS Service VPort.

There are multiple ways in which such a logical topology can be constructed. A user may utilize a diagramming editor packaged as an application or a Web 2.0 service. The editor would support the addition of virtual appliance models, and their graphical composition using linking or error resolution gestures. Alternatively, the topology can be created through a wizard interaction in which a user is queried about the types of appliances to be added, with possible links created or auto-suggested. Finally, such a topology can be created manually by editing the low level XML representation. We have experimented with all three types of construction which we report in section 7.

Independent of the manner in which the topology is constructed, a key underlying mechanism of composition is VPort negotiation. When two VPorts are linked, they must negotiate a consistent configuration. VPorts are designed to encapsulate all properties needed to effect configuration. Therefore the negotiation can be localized to the two endpoints. When configuring communication, users often think of clients adjusting to the configuration of the server. Especially for new deployments, client configuration constraints may actually be more restrictive. For example, in the topology of figure 4, the HTTP connection between the web browser and the J2EE server will negotiate the TCP port value. If the client has been fixed to port 80, then the selected server configuration must match, or be changeable.

Negotiation may not settle on a specific configuration value, but only on the fact that it will be a shared value provided at deploy time. For example, depending on the cloud infrastructure, the host name may be user selectable or may be chosen by the cloud. In either case, at the time of negotiation the value will not be known. Instead, the negotiation will focus on checking that any known constraints are satisfiable. Constraint satisfiability is a difficult problem even when the language for expressing constraints is highly restricted[30]. In such cases negotiation may either defer to the runtime, or ask the user for a representative sample to test. In our initial prototype we have used very limited set of constraints, such as *range* and *enumeration* constraints for which efficient algorithms exist.

4.2 Environment Specification

The functions and services of IaaS clouds are still far from standardized. Due to the similarity of function, however, there are some common configuration abstractions that can be generalized. For example, clouds often support constraining deployment to specific geographical regions, or segment their data centers into independent availability zones to support highly available deployments. IaaS clouds will also typically allow users to constrain the protocols and ports which are exposed in their deployed images. We will next outline three core cloud abstractions that we have developed to logically model cloud deployments.

We define a model of a *Placement Zone* that can group any number of image models. The placement zone can be constrained to be realized in one or more geopolitical regions such as continents, political and trade associations, countries, states, and so on. A zone may be further constrained to identify the level of sharing that the customer is permitting whether it is at the data-center level, the rack/switch level, or the physical machine level. Constraint relationships can be established between zones to establish anti-collocation properties. Two zones may be constrained to be realized in different cloud providers, in different availability zones within the same provider, or across geopolitical zones in the same or different clouds.

The links between VPorts represent communication constraints that must be satisfied by the cloud network infrastructure. We complement these communication constraints with a model of a *Network Zone* that can group any number of appliances. Zones enable designers to express additional communication constraints that will apply to multiple images. For example, they can be used to open communication paths that may not have been explicitly modeled using VPorts. Moreover, network zones can be used to relax communication constraints between member images. At deploy time, network zones may be realized by zones already defined in the infrastructure. The realization of network zones may require the creation of multiple zones in the cloud if functions such as intra-zone communication constraints are not supported.

We introduce the logical model of a *Storage Volume* that can be linked to any number of images. The storage volume can be a member of a placement zone. Its realization can be constrained in terms of the level of sharing of physical storage with other customers, the use and level of encryption, its physical mapping properties (e.g. whether its bootable), and its I/O performance characteristics. Users can associate an artifact to populate the device, such as an ISO file.

VSMs must be validated for internal consistency to detect link endpoint type mismatches, inconsistent VPort and product section values, constraint violations, and contradictory environment requirements. In our prototype we used the deployment validation framework we have contributed to RSA 7.5 to register domain validation logic, and report errors to the users [12].

5. SOLUTION DEPLOYMENT MODELING

The virtual solution model determines the composition of virtual appliances at the interface level and constrains their deployment environment in a platform-independent manner. In order to enable deployment of the virtual solution, the virtual appliance interfaces must be bound to specific implementations and the platform independent environment models must be realized by cloud specific configuration models. This transformation is performed by a *deployment architect* and its result is a *virtual solution deployment model* (VSDM). The deployment model is platform specific and can be used to generate a parameterized *deployment plan* to provision it in one or more clouds, as will be outlined in the next section.

In selecting one or more clouds to deploy a virtual solution, the deployment architect must consider the availability of virtual appliance implementations and the capabilities of the clouds selected to implement the environment constraints. If the images are available in an external asset repository, then the deployer will need to upload and register them with the cloud. If the format and content of the images is not directly compatible with the cloud then a domain expert may need to be involved. In this paper we will restrict our discussion to deployments of virtual appliances where matching images can be found in the targeted cloud.

Consider the transformation of the virtual solution model from the previous section for deployment on the Amazon Elastic Compute Cloud (EC2)[2]. The first step of the transformation, which is depicted in figure 5 involves identifying the Amazon Machine Image (AMIs) for each of the three types of virtual appliances in the solution. The appliance models for the AMIs may extend the solution appliance models with additional configuration properties and VPorts that are Amazon EC2-specific. For example, an AMI instance can be linked to an Amazon EC2 Elastic Block Storage volume for device level access to storage. Similarly, an AMI instance can be associated with an Elastic IP to maintain a static address that can be reassigned in case of failure. Domain-specific validation rules will also apply, such as that an AMI instance must be on the same availability zone as its EBS volumes. Finally, the mapping of an appliance interface may be more complex if its implementation involves additional resources such as a block storage device for persistent storage. Depending on the implementation, a new block storage device may need to be allocated and configured for mounting, or an existing one may need to be cloned and mounted. In the figure we show how the ITDS appliance model is realized by an AMI configured to mount a new storage volume.

The second step of the transformation is to realize the virtual solution environment constraints in the target cloud infrastructure. In the case of Amazon EC2, network isolation zones are modeled as security groups that are associated with AMI instances and configured with IP permission rules. The rules open specific protocol port ranges and can apply

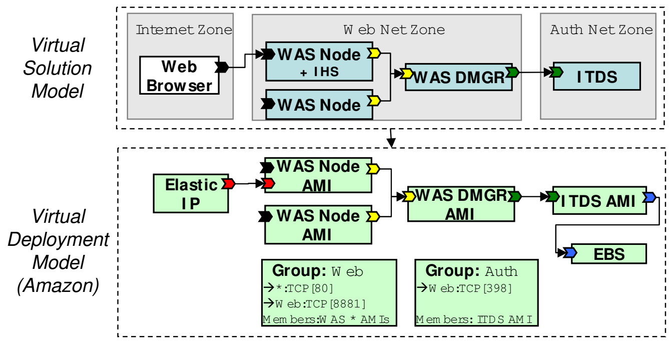

Figure 5: Virtual Solution Deployment Model

to specific IP addresses, IP subnets, or members of a security group. The default policy is to deny all traffic from outside or inside the zone. The transformation of the virtual solution model's communication constraints and network zones into Amazon EC2 availability zones can be assisted by tooling as we have reported in [11]. For each network zone a security group is created. The VPort connectivity at the virtual solution model level implies network connectivity requirements. Each VPort configuration link must be transformed into its corresponding network layer connections. For example, the WAS Nodes are configured to be federated by the WAS Deployment Manager using the JMX protocol over TCP port 8881. As all three image instances will be in the same zone (Web) an IP permission rule can be defined to allow intra-zone TCP communication on port 8881. Similarly, the Web security group can be configured to expose TCP port 80 for Internet traffic, and the Auth security group to expose TCP port 398 for LDAP traffic from the Web group.

From a functional perspective, there may be multiple realizations of a virtual solution into a set of clouds. In such cases, deployment architects will consider non-functional measures such as cost, time to deploy, latency, scalability and availability. The discussion of algorithms for availability and performance analysis is beyond the scope of this paper. See [32] for an example of how our model-driven transformation approach was applied to high-availability pattern analysis. Specific mechanisms for solution to deployment model transformation are outside the scope of this paper. We have previously reported on a deployment model transformation approach based on a search through a set of pattern-based transformations[12, 11].

6. SOLUTION DEPLOYMENT PLANNING

Once the Virtual Solution Deployment Model has been finalized the deployment architect must generate a plan to provision it in the cloud. The deployment plan is a partial order of cloud and image configuration operations. When local image configuration operations are used, the execution of the plan will occur at two levels. The top execution level will invoke the cloud APIs to configure the environment (e.g. create security groups), and to deploy the images. The deployment of each image will be parameterized by user data containing an environment document with parameter values, such as the root user password, and any configuration operations that are not present in the image. The second level of execution will occur inside the image at the time it is

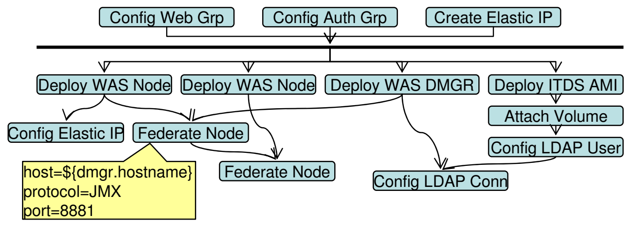

Figure 6: Deployment Plan Example

started when the configuration operation engine will invoke the operations specified in the environment document.

There are two challenges to generating such a deployment plan. The first challenge is to identify the operations which will "cover" the deployment model, and map their parameters to deployment model object attributes. The second challenge is to identify the temporal dependencies between operations to determine the partial order of execution. We have previously reported on an AI planning-based approach to deployment plan generation in [13]. The general AI planning approach is necessitated where the transition from initial to goal state involves creation of transient configurations. Due to the simpler nature of the deployment models associated with virtual solutions, it is possible to apply more restricted planners for a gain in performance and reduced algorithm complexity.

Our simplified planning approach is based on our work on deployment patterns [4]. We capture the mapping of a configuration operation against a conceptual model of the objects which it configures, including the propagation of attributes from the model to the operation. We then search through the realizations of configuration operation patterns on the VSDM topology to find a complete covering of all of the objects that need to be provisioned. The realization mapping identifies the operation and the relation of its parameters to the model attributes. In our prototype work we created operation patterns for the Amazon EC2 operations, such as image instantiation, security group creation, elastic IP creation and so on. To determine the order of the operations, we associate VPorts with a constraint indicating that their linkage implies a deploy order dependency. We then project the deployment model ordering constraints on the operation model we have created. The details of this approach will be covered in a future publication.

Figure 6 depicts a plan for the deployment model of figure 5. Based on the VPort link constraints, the plan specifies that the groups and elastic IPs have to be configured before the AMIs are instantiated. The images can be deployed in parallel, however, the configuration of the VPorts is associated with temporal constraints (not shown in the figure). One constraint is that the deployment manager can only federate one node at a time. For WAS security to be enforced, the ITDS server must be up and configured with the WAS admin user entry. Finally, the Elastic IP can only be assigned to the WAS Node AMI after it has been deployed. Every operation has parameters which are bound to constants or to values from the model. For example, the operation to federate a WAS node will be assigned the specific host name at deploy time, whereas the protocol type and port are fixed because the VPort does not expose them as a parameter.

7. PROTOTYPE IMPLEMENTATION

We have implemented a prototype of our architecture for virtual solution deployment. Our prototype is built on the commercialized version of our deployment modeling research platform[12] which is now a component of IBM Rational Software Architect (RSA) version 7.5. The platform provides an extensible framework for capturing deployment domains including schemas, templates, and associated validation and resolution rules. The platform can be extended with model providers for objects stored in external repositories. The product ships with a number of deployment domains including middleware, operating system, servers, networks, and others. Model transformers and publishers may also be registered to assist users in their model-driven workflow. These functions are exposed through a rich graphical editor that is part of an Eclipse[10] workbench.

In our prototype, we created a domain to capture the different types of VPort models we have identified. Our VPort types are associated with negotiation policies which can be customized at the instance level and are contributed in the form of validations and resolutions. When two VPorts are graphically linked, a negotiation engine we have created uses these validation and resolution rules to broker a configuration that is valid for both endpoints. We leveraged the virtualization domain that we had contributed to RSA and extended it with models for Amazon EC2. We also extended the platform with a graphical navigator for EC2 resources such as images, security groups, and elastic IPs. We wrote the transformation logic to map EC2 resource instances into our EC2 model enabling deployment architects to design complex EC2 topologies. For the generation of the deployment plan we wrote a custom pattern-based planner which we outlined in the previous section. We modeled the Amazon EC2 SOAP API and mapped it to the EC2 domain types. Finally, we developed a graphical wizard to support the role of the deployer in collecting the solution instance parameters and executing the deployment workflow.

To demonstrate our prototype we created four virtual appliances: (1) a WAS application server appliance to host the TPC-W benchmark J2EE application[29], (2) an IBM DB2 database appliance, (3) an Oracle Express database appliance, and (4) a IBM ITDS directory appliance. Our TPC-W appliance requires database connectivity in the form of an out VPort and supports custom DDLs for either DB2 or Oracle. Neither the DB2 nor the Oracle appliance has any knowledge of the TPC-W appliance. In our demonstration, solution architects can connect the TPC-W appliance to either DB2 or Oracle, resulting in appropriate bindings being generated on both endpoints. The TPC-W appliance can also optionally be connected to the ITDS directory server. The connection captures the passing of the LDIF information to create the appropriate users on the LDAP server at deploy time. A sample screen-shot of a virtual solution created in our editor is shown in figure 7.

Our virtual appliances have been implemented as SUSE v10 Xen Linux virtual images containing local configuration operations as IBM Activation Engine [17] scripts. The images were tested on our local research Xen virtualization infrastructure, and also ported as Amazon Machine Images (AMIs) in EC2. The same models for the configuration operations were used for both infrastructure targets, enabling us to test alternate appliance implementations. For our local deployment we generated an IBM Solution Assembly Toolkit

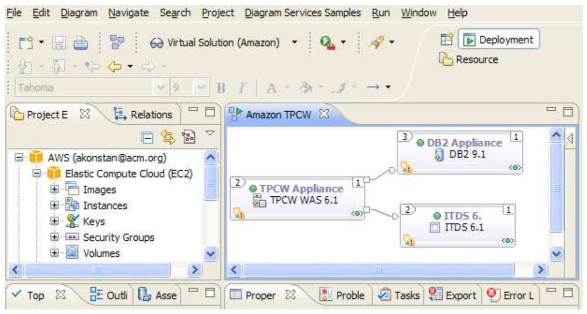

Figure 7: Prototype Composition Example

workflow to execute the Xen image copy and deployment operations. For our EC2 deployment we utilized the Eclipse job manager to execute the Amazon EC2 SOAP calls necessary to create the security groups and to deploy the images.

In our demonstration, we show how a solution architect can assemble a virtual solution model by adding virtual appliance models and simply linking their VPorts. For example, for a topology with TPC-W + DB2 + ITDS, this can be achieved in just five simple user gestures. All the negotiations over port parameters are performed automatically as part of the linking gesture. We then show how a deployment architect can realize the appliances with existing Amazon EC2 images, and add additional EC2-specific configurations, such as security groups and Elastic IPs, by dragging and dropping from the EC2 navigator, and using link gestures to the virtual appliances. Finally, we show how a deployer can start a simple wizard to deploy an instance of the solution by supplying parameters such as the Amazon machine type (most parameters are associated with valid default values). When the wizard completes, it generates the activation profiles to be passed as user data at image deployment, and invokes the Amazon SOAP API operations identified by the planner in the right order.

8. RELATED WORK

Cloud computing is an emerging technology which promises to revolutionize software and business life-cycles. It is a subject of fierce debate in forums and on-line communities, with rapid technological developments, and intense business analyses and speculations. As a research field, cloud computing is a derivative of several established research areas, including Service Oriented Architecture, grid computing, and virtualization.

Work in this area can be categorized by layer. At the base layer, there is a massive body of work on mechanisms for server virtualization in the form of software and hardware hypervisors[5, 24, 31], and control and monitoring of virtual machines[21, 22]. In the next layer, virtualization technology is leveraged for development of cloud software infrastructure offering compute services on demand[2, 16, 6, 26]. Multiple works attempt to define a unified taxonomy for cloud computing[33], or provide an open source implementation[26, 25]. Other works focus on the life-cycle aspects of individual virtual machines. Automatic or tool-aided assembly of individual virtual machines is explored in[20], and offered by companies such as CohesiveFT[7] and rPath[28]. Efficient transfer, scheduling and deployment of virtual machines is described in [19].

We are not aware of other academic publications on composing distributed virtual services consisting of multiple virtual machines that are similar to the approach proposed in this paper. Our work can be generally positioned in the area of cloud management. Companies such as RightScale[27] and Elastra[14], operate in the same space. RightScale allows users to identify scripts that will be automatically executed on activated virtual machines in a cloud environment. Elastra proposes an approach very similar to ours where virtualization is leveraged for the creation of modular application infrastructure components. 3Tera[1] is another interesting company that takes a model-driven approach to design and provisioning of virtual hosting infrastructure per individual distributed application requirements. Other companies offer Platform-as-a-Service solutions that provide services and tools for the entire software life cycle but are targeted to support a very specific application programming model (e.g., Google's App Engine[3] assumes a Web application model), or platform (e.g., Microsoft's Azure[23] based on .Net). Composition is also a key aspect of the Service Oriented Computing paradigm[8], but in contrast to our work, in SOA the composition elements are web services and the composition model drives runtime behavior[15], not deploy time software configuration.

9. FUTURE WORK

Having demonstrated the advantage of using virtual appliance models to construct virtual solutions, our team is working on simplifying the task of creating virtual appliances. In parallel, we are extending our work on logical cloud environment specification, and are examining different approaches to cloud selection, and platform-independent to platform-specific transformations. High-availability requirements, combined with the emergence of private clouds and the concept of "surge computing" raise interesting challenges in multi-cloud deployments. Re-deployment of running services across clouds in reaction to failures, changes in utilization, or cost is also an area which we plan on investigating. The standardization of cloud interfaces and concepts is at a very early stage. In our work, we aim to influence the manner in which virtual solutions are designed and deployed in an IaaS cloud, and the representation of virtual appliance models.

10. REFERENCES

[1] 3Tera. http://3Tera.com/.
[2] Amazon EC2. http://aws.amazon.com/ec2/.
[3] App Engine. http://code.google.com/appengine/.
[4] W. Arnold, T. Eilam, M. Kalantar, A. Konstantinou, and A. Totok. Pattern based SOA deployment. In *ICSOC*, volume 4749 of *LNCS*. Springer, 2007.
[5] B. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. 2003.
[6] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle. Dynamic virtual cluster in a grid site manager. In *HPDC*, 2003.
[7] CohesiveFT. http://cohesiveft.com/.
[8] F. Curbera, D. Ferguson, M. Nally, and M. L. Stockton. Towards a programming model for Service-Oriented Computing. In *ICSOC*, volume 3826 of *LNCS*, pages 33–47. Springer-Verlag, 2005.

[9] DMTF. Open Virtualization Format Specification. Technical Report DSP0243, DMTF, 2009.

[10] Eclipse. http://eclipse.org, 2009.

[11] T. Eilam, M. Kalantar, A. Konstantinou, and G. Pacifici. Reducing the complexity of application deployment in large data centers. In *IM*, 2005.

[12] T. Eilam, M. Kalantar, A. Konstantinou, G. Pacifici, J. Pershing, and A. Agrawal. Managing the configuration complexity of distributed applications in internet data centers. *IEEE Communication Magazine*, 44(3):166–177, 2006.

[13] K. El Maghraoui, A. Meghranjani, T. Eilam, M. Kalantar, and A. Konstantinou. Model driven provisioning: Bridging the gap between declarative object models and procedural provisioning tools. In *Middleware*, volume 4290 of *LNCS*. Springer, 2006.

[14] Elastra. http://elastra.com/.

[15] C. Emig, K. Krutz, S. Link, C. Momm, and S. Abeck. Model-driven development of SOA services. Technical report, Forschungsbericht, Apr. 2007.

[16] Enomaly. http://enomaly.com/.

[17] L. He, S. Smith, R. Willenborg, and Q. Wang. Automating deployment and activation of virtual images. Technical Report 0708, IBM WebSphere Journal, 2007.

[18] IBM. Rational Software Architect (RSA), 2008.

[19] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, and T. Delaitre. Automatic service deployment using virtualization. In *16th Euromicro PDP*, 2008.

[20] I. Krsul, A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo. VMPlants: Providing and managing virtual machine execution environments for grid computing. In *ACM/IEEE Supercomputing*, 2004.

[21] A. Menon, A. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *USENIX*, 2006.

[22] M. Mergen, V. Uhlig, O. Kireger, and J. Xenidis. Virtualization for high performance computing. In *SIGOPS Oper. Syst. Rev.*, 2006.

[23] Microsoft Azure. http://microsoft.com/azure/.

[24] Microsoft Hyper-V. http://microsoft.com/hyperv.

[25] Nimbus Toolkit. http://workspace.globus.org/.

[26] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and Z. D. The eucalyptus open source cloud computing system. In *Cloud Computing and Its Applications*, 2008.

[27] RightScale. http://rightscale.com/.

[28] rPath. http://rpath.com/.

[29] A. Totok. TPC-W-NYU: J2EE-based implementation of the TPC-W benchmark, 2005. NYU.

[30] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press - Harcourt Brace & Company, 1993.

[31] VMware. http://vmware.com/.

[32] L. Xie, J. Luo, J. Qiu, J. A. Pershing, Y. Li, and Y. Chen. Availability “weak point” analysis over a SOA deployment framework. In *NOMS*. IEEE, 2008.

[33] L. Youseff and D. Butrico, M. Da Silva. Towards an ontology of cloud computing. In *Grid Computing Environments (GCE08)*, 2008.