WHITE PAPER

The Open Virtual Machine Format Whitepaper for OVF Specification

version 0.9

Executive Summary

The rapid adoption of virtual infrastructure has highlighted the need for a standard, portable meta-data model for the distribution of virtual machines to and between virtualization platforms. Packaging an application together with the operating system on which it is certified, into a virtual machine that can be easily transferred from an ISV, through test and development and into production as a pre-configured, pre-packaged unit with no external dependencies, is extremely attractive. Such pre-deployed, ready to run applications packaged as virtual machines (VMs) are called virtual appliances. In order to make this concept practical on a large scale it is important that the industry adopts a vendor-neutral standard for the packaging of such VMs and the meta-data that are required to automatically and securely install, configure, and run the virtual appliance on any virtualization platform.

Virtual appliances are changing the software distribution paradigm because they allow application builders to optimize the software stack for their application and deliver a turnkey software service to the end user. For solution providers, building a virtual appliance is simpler and more cost effective than building a hardware appliance, since the application is pre-packaged with the operating system that it uses, reducing application/OS compatibility testing and certification, and allowing the software to be pre-installed in the OS environment it will run in – by the ISV. For end users, virtual appliances offer an opportunity to dramatically simplify the software management lifecycle through the adoption of a standardized, automated, and efficient set of processes that replace OS and application specific management tasks today.

Whereas current virtual appliances contain a single VM only, modern enterprise applications model service oriented architectures (SOA) with multiple tiers, where each tier contains one or more machines. A single VM model is thus not sufficient to distribute a multi-tier service. In addition, complex applications require install-time customization of networks and other customer specific properties. Furthermore, a virtual appliance is packaged in a run-time format with hard disk images and configuration data suitable for a particular hypervisor. Run-time formats are optimized for execution and not for distribution. For efficient software distribution, a number of additional features become critical, including portability, platform independence, verification, signing, versioning, and licensing terms.

The Open Virtual Machine Format (OVF) specification is a hypervisor-neutral, efficient, extensible, and open specification for the packaging and distribution of virtual appliances composed of one or more VMs. It aims to facilitate the automated, secure management not only of virtual machines but the appliance as a functional unit. For the OVF format to succeed it must be developed and endorsed by ISVs, virtual appliance vendors, operating system vendors, as well as virtual platform vendors, and must be developed within a standards-based framework.

Introduction

The Open Virtual Machine Format (OVF) describes an open, secure, portable, efficient and extensible format for the packaging and distribution of (collections of) virtual machines. The key properties of the format are:

- **Optimized for distribution**
  Supports content verification and integrity checking based on industry standard public key infrastructure, and provides a basic scheme for management of software licensing.

- **Optimized for a simple, automated user experience**
  Supports validation of the entire package and each virtual machine or meta-data component of the OVF during the installation phases of the VM lifecycle management process. It also packages with the appliance relevant user-readable descriptive information that can be use by a virtualization platform to streamline the installation experience.

- **Supports both single VM and multi-VM configurations**
  Supports both standard single VM packages, and packages containing complex, multi-tier services consisting of multiple interdependent VMs.

- **Portable VM packaging**
  OVF is virtualization platform neutral, while also enabling platform-specific enhancements to be captured. It supports the full range of virtual hard disk formats used for VMs today, and is extensible to deal with future formats that may arise. Virtual machine properties are captured concisely and accurately.

- **Vendor and platform independent**
  The OVF does not rely on the use of a specific host platform, virtualization platform, or guest operating system (within the appliance).

- **Extensible**
  OVF is immediately useful – and extensible. It is designed to be extended as the industry moves forward with the virtual appliance technology. It also supports and permits the encoding of vendor specific meta-data to support specific vertical markets.

- **Localizable**
  Supports user visible descriptions in multiple locales, and supports localization of the interactive processes during installation of an appliance. This allows a single packaged appliance to serve multiple market opportunities.

- **Open standard**
  The OVF has arisen from the collaboration of key vendors in the industry, and will be developed in an accepted industry forum as a future standard for portable virtual machines.

From the user's point of view, an OVF is a **packaging format for software appliances**. Once installed, an OVF adds to the user’s infrastructure a self-contained, self-consistent, software solution for achieving a particular goal. For example, an OVF might contain a fully-functional and tested web-server / database / OS combination, such as a LAMP stack (Linux + Apache + MySQL + PHP), or it may contain a virus checker, including its update software, spyware detector, etc.

From a technical point of view, an OVF is a **transport** mechanism for **virtual machine templates**. One OVF may contain a single VM, or many VMs (it is left to the software appliance developer to decide which arrangement best suits their application). OVFs must be installed before they can be run; a particular virtualization platform may run the VM from the

OVF, but this is not required. If this is done, the OVF itself can no longer be viewed as a “golden image” version of the appliance, since run-time state for the virtual machine(s) will pervade the OVF. Moreover the digital signature that allows the platform to check the integrity of the OVF will be invalid.

As a transport mechanism, OVF differs from VMware's VMDK Virtual Disk Format and Microsoft's VHD Virtual Hard Disk format or the open source QCOW format. These are run-time VM image formats, operating at the scope of a single VM disk, and though they are frequently used as transport formats today, they are not designed to solve the VM portability problem; they don't help you if you have a VM with multiple disks, or multiple VMs, or need customization of the VM at install time, or if your VM is intended to run on multiple virtualization platforms (even if the virtualization platforms claim support of the particular virtual hard disk format used).

Included within the OVF remit is the concept of the **certification and integrity** of a packaged software virtual appliance, allowing the platform to determine the provenance of the appliance, and to allow the end-user to make the appropriate trust decisions. The OVF specification has been constructed so that the appliance is responsible for its own configuration and modification. In particular, this means that the virtualization platform does not need to be able to read from the appliance's file systems. This decoupling of platform from appliance means that OVFs may be implemented using any operating system, and installed on any virtualization platform that supports the OVF format. A specific mechanism is provided for appliances to detect the platform on which they are installed, and react to it. This allows platforms to extend this specification in unique ways without breaking compatibility of appliances across the industry.

This document gives a detailed description of the motivation and goals behind the design of OVF, and should be read as an accompaniment to the OVF specification of the same revision number.

Appliances and Appliance Life-Cycle

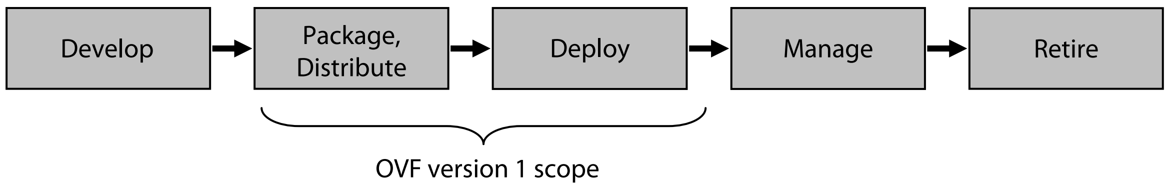
A virtual appliance is a pre-configured software stack comprising one or more virtual machines. Each virtual machine is an independently installable run-time entity comprising an operating system, applications and other application-specific data, as well as a specification of the virtual hardware that is required by the virtual machine. Many infrastructure applications and even end-user applications that are accessible over a network, such as a DNS server, a bug tracking database, or a complete CRM solution composed of a web, application and database tier, can be delivered as virtual appliances. Delivering complex software systems and services as a pre-configured software stack can dramatically increase robustness and simplify installation. Virtual appliances need not be developed and delivered by 3rd party ISVs – the concept is equally useful and often used within an enterprise in which a *virtual machine template* for a particular service is assembled, tested, and certified by an IT organization and then packaged for repeated, “cookie cutter” deployment throughout the enterprise.

Commonly, a service is implemented as a multi-tier application running in multiple virtual machines and communicating across the network. Services are often composed of other services, which themselves might be multi-tier applications or composed of other services. This is known as service-oriented architecture or SOA. Indeed the SOA-type model naturally fits into a virtual appliance-based infrastructure, since virtual appliances are typified by the use of network facing, XML based management and service interfaces that allow composition of appliances to deliver a complete application.

For example, consider a typical web application that consists of three tiers. A web tier that implements the presentation logic, and application server tier that implements the business logic, and a back-end database tier. A straightforward implementation would divide this into 3 virtual machines, one for each tier. In this way, the application can scale from the fraction of a single physical host to 3 physical hosts. Another approach is to treat each tier as a service in itself.

Hence, each tier is a multi-VM service that provides a clustered solution. This can provide far greater scalability than just up to 3 physical hosts. Taking the web-front example, a common scenario is to have many web servers, fewer applications servers, and one or two database servers. Implemented as virtual machines, each tier can scale across as many or as few physical machines as required, and each tier can support multiple instances of service VMs.

The software life cycle for virtual appliances is shown below:



A service, consisting of one or more VMs and the relevant configuration and deployment meta data, is packaged into the OVF format at the end of the development phase. The components used here can be third-party components. For example, a clustered database component might be acquired from a third-party ISV. The deployment phase is the installation of an OVF package. The management and retirement phase is specific to the virtualization product used, and to the contents of the OVF itself. Management includes, for example, ongoing maintenance and upgrade of the appliance, which is likely to be highly dependent on the contents of the VMs in the OVF. In the retirement phase, the software is decommissioned and any resources it consumes are released. In this version of the OVF specification we deal specifically with the packaging, distribution and deployment phases. Later versions of the specification will address management and retirement in detail.

The OVF format has several specific features that are designed for complex, multi-tier services and their associated distribution, installation, configuration and execution workflow:

- It directly supports the configuration of multi-tier applications and the composition of virtual machines to deliver composed services.
- It permits the specification of both VM and application-level configuration.
- It offers robust mechanisms for validation of the contents of the OVF, and full support for unattended installation to ease the burden of deployment for users, and thereby enhance the user's experience.
- It uses commercially accepted procedures for integrity checking of the contents of the OVF, through the use of signatures and trusted third parties. This serves to reassure the consumer of an appliance that it has not been modified since signed by the creator of the appliance. This is seen as critical to the success of the virtual appliance market, and to the viability of independent creation and online download of appliances.
- It permits commercial interests of the appliance vendor and user to be respected, by providing a basic method for presentation and acknowledgement of licensing terms associated with the appliance.

Portability

OVF is an enabling technology for enhancing portability of virtual appliances and their associated virtual machines. An OVF package contains a recipe for creating virtual machines that can be interpreted concisely by a virtualization platform. The packaged meta-data enables a robust and user-friendly experience when installing a virtual appliance. In particular, the meta-data can be used by the management infrastructure to confidently decide whether a particular VM described in an OVF can be installed or whether it should be rejected, and potentially to guide appropriate conversions and localizations to make it runnable in the specific execution context in which it is to be installed. This includes virtualization platform specific customization such as the installation of drivers and/or agents into the virtual machine(s) of the appliance, and the optimal selection of a specific (virtualized) hardware platform to host the appliance, based on its resource requirements.

There are many factors that are beyond the control of the OVF format specification and even a fully compliant implementation of it, that determine the portability of a packaged virtual machine. That is, the act of packaging a virtual machine into an OVF package does not guarantee universal portability or install-ability across all hypervisors. While it is the goal of OVF to provide a virtualization platform independent (and hence portable) packaging format for one or more VMs, in this version of the specification the portability is restricted to so-called ***portability from origin***, rather than transparent or complete portability between virtual platforms. It is beyond the scope of this version of the specification to deal with the automation of the portability of an OVF between virtual platforms (generally referred to as Virtual to Virtual or V2V, in the industry). The aspirational goal of full portability is one that deserves attention from a broader community of vendors, within an appropriate industry body, and will be addressed in a later version of the OVF specification

Below are some of the factors that could limit portability:

- The VMs in the OVF could contain virtual disks in a format that is not understood by the hypervisor attempting the installation. While it is reasonable to expect that most hypervisors will be able to import and/or export VMs in any of the major virtual hard disk formats, newer formats may arise that are supported by the OVF and not a particular hypervisor. It may be useful in future versions of this specification, to stipulate a required set of virtual hard disk formats that must be supported by an OVF compliant hypervisor.

- The installed guest software may not support the virtual hardware presented by the hypervisor. By way of example, the Xen hypervisor does not by default offer a virtualized floppy disk device to guests. One could conceive of a guest VM that would require interaction with a floppy disk controller and which therefore would not be able to execute the VM correctly.

- The installed guest software does not support the CPU architecture. For example, the guest software might execute CPU operations specific to certain processor models or require specific floating point support, or contain opcodes specific to a particular vendor's CPU.

- The virtualization platform might not understand a feature requested in the OVF descriptor. For example, composed services may not be supported. Since the OVF standard will evolve independently of virtualization products, at any point an OVF might be unsupportable on a virtualization platform that pre-dates that OVF specification.

The portability of an OVF can be categorized into the following 3 levels:

- **Level 1**. Only runs on a particular virtualization product and/or CPU architecture and/or virtual hardware selection. This would typically be due to the OVF containing suspended virtual machines or snapshots of powered on virtual machines, including the current run-time state of the CPU and real or emulated devices. Such state ties the OVF to a very specific virtualization and hardware platform.

- **Level 2**. Runs on a specific family of virtual hardware. This would typically be due to lack of driver support by the installed guest software. Examples of virtual hardware families include the VMware 4th generation of virtual hardware which is supported across the VMware products ACE, Player, Workstation, Fusion, ESX 2.x, and ESX 3.x, or the Xen 3.1 HVM or paravirtualized virtual hardware.

- **Level 3**. Runs on multiple families of virtual hardware. For example, the appliance could be runnable on Xen, KVM, Microsoft, and VMware hypervisors. For level 3 compatibility, the guest software has been developed to support the devices of multiple hypervisors. A clean install and boot of a guest OS, during which the guest OS performs hardware device discovery and installs any specialized drivers required to interact with the virtual platform, is an example of Level 3 portability of an OVF. The "sysprep" level of portability for Microsoft Windows® operating systems is another example. Such OS instances can be re-installed, re-named and re-personalized on multiple hardware platforms, including virtual hardware.

For use within an organization, Level 1 or Level 2 compatibility may be good enough, since the OVF package is distributed within a controlled environment where specific purchasing decisions of hardware or virtualization platforms can ensure consistency of the underlying feature set for the OVF. For commercial appliances independently created and distributed by ISVs, Level 3 compatibility is highly desirable. Indeed, Level 3 compatibility ensures that the appliance is readily available for the broadest possible customer base both for evaluation and production.

The OVF virtual hardware description is designed to support Level 1 through Level 3 portability. For Level 3 portability it is possible to include only very general descriptions of hardware requirements, or to specify multiple alternative virtual hardware descriptions. An example would be to stipulate that an OVF is compatible with VMware 4th generation virtual hardware and Xen 3.1 HVM hardware. The appliance provider is in full control of how flexible or restrictive the virtual hardware specification is made. A narrow specification can be used to constrain an appliance to run on only known-good virtual hardware, while limiting its portability somewhat. A broad specification makes the appliance useful across as wide a set of virtual hardware as possible. This ensures that customers have the best possible user experience, which is one of the main requirements for the success of the virtual appliance concept.

This is further discussed in the Examples and Best Practices section.

Operational Models

The creation of an OVF involves the i) packaging of a set of VMs onto a set of virtual disks, ii) appropriately encoding those virtual disks, iii) attaching an OVF descriptor with a specification of the virtual hardware, licensing, and other customization metadata, and iv) optionally digitally signing the package. The process of installing or importing an OVF occurs when a virtualization platform consumes the OVF and creates a set of virtual machines from its contents.

Installation transforms the virtual machines in an OVF package into the runtime format understood by the target virtualization platform, with the appropriate resource assignments and supported by the correct virtual hardware. During installation, the platform validates the OVF integrity, making sure that the OVF package has not been modified in transit, and checks that it is compatible with the local virtual hardware. It also assigns resources to, and configures the virtual machines for the particular environment on the target virtualization platform. This includes assigning and configuring the (physical and virtual) networks to which the virtual machines must be connected; assigning storage resources for the VMs, including virtual hard disks as well as any transient data sets, connections to clustered or networked storage and the like; configuring CPU and memory resources, and customizing application level properties. OVF does not support the conversion of guest software between processor architectures or hardware platforms. Installation instantiates one or more virtual machines with a hardware profile that is compatible with the requirements captured in the OVF descriptor, and a set of virtual disks with the content specified in the OVF package.

Creating an OVF can be made as simple as exporting an existing virtual machine from a virtualization platform into an OVF package, and adding to it the relevant meta-data needed to correctly install and execute it. This will transform the virtual machine from its current runtime state on a particular hypervisor into an OVF package. During this process, the virtual machine's disks can be compressed to make it more convenient to distribute. A simple export of a virtual machine will typically create an OVF with Level 1 or Level 2 compatibility (tied to a specific set of virtual hardware), however it is easy to extend the metaphor to support the export of Level 3 compatibility, for example through the use of utilities such as "sysprep" for Windows.

For commercial-grade virtual appliances where Level 3 portability is desired, a standard build environment may be used to produce an OVF package. For example, the OVF descriptor can be managed using a source control system, and the OVF package can be built using a reproducible scripting environment (such as `make` files) or, through the use of appliance building toolkits that are available from multiple vendors. Such toolkits will generally be used to create certified "known good" Level 3 packages of the appliance for broad distribution and installation on multiple virtual platforms, or Level 2 compatibility packages if the appliance is to be consumed within the context of a narrower set of virtual hardware, such as within a particular development group in an enterprise.

When an OVF is created, it must be accompanied with appliance-specific post-installation configuration metadata. This includes metadata for optional localization of the interface language(s) of the appliance, review/signoff and/or enforcement of the EULA, and resource configuration. It can also involve the addition of special drivers, agents and other tools to the guest to enhance (for example) I/O, timekeeping, memory management, monitoring and orderly shutdown.

Examples and Best Practices

The following listing shows a complete OVF descriptor for a typical single virtual machine appliance:

```
<?xml version="1.0" encoding="UTF-8"?>
<ovf:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ovf="http://schemas.dmtf.org/ovf/1/envelope"
    xmlns:vssd="http://schemas.dmtf.org/wbem/wscim/1/cimschema/2/CIM_VirtualSystemSettingData"
    xmlns:rasd="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData"
    ovf:version="0.9">

    <!-- References to all external files -->
    <References>
        <File ovf:id="file1" ovf:href="vmdisk1.vmdk" ovf:size="180114671"/>
    </References>
    <!-- Describes meta-information for all virtual disks in the package -->
    <Section xsi:type="ovf:DiskSection_Type">
        <Info>Describes the set of virtual disks</Info>
        <Disk ovf:diskId="vmdisk1" ovf:fileRef="file1" ovf:capacity="4294967296"
              ovf:format="http://www.vmware.com/specifications/vmdk.html#sparse"/>
    </Section>
    <!-- Describes all networks used in the package -->
    <Section xsi:type="ovf:NetworkSection_Type">
        <Info>List of logical networks used in the package</Info>
        <Network ovf:name="VM Network">
            <Description>The network that the service will be available on</Description>
        </Network>
    </Section>
    <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="Virtual Appliance One">
        <Info>The primordial appliance</Info>
        <Section xsi:type="ovf:ProductSection_Type">
            <Info>Describes product information for the appliance</Info>
            <Product>The Great Appliance</Product>
            <Vendor>Some Great Corporation</Vendor>
            <Version>13.00</Version>
            <Full-version>13.00-b5</Full-version>
            <ProductUrl>http://www.somegreatcorporation.com/greatappliance</ProductUrl>
            <VendorUrl>http://www.somegreatcorporation.com/</VendorUrl>
            <AppUrl>http://${app.ip}/</AppUrl>
        </Section>
        <Section xsi:type="ovf:PropertySection_Type">
            <Info>Defines the properties used by the service</Info>
            <Property ovf:key="admin.email" ovf:type="string">
                <Description>Email address of administrator</Description>
            </Property>
            <Property ovf:key="app.ip" ovf:type="string" ovf:defaultValue="192.168.0.10">
                <Description>The IP address of this appliance</Description>
            </Property>
        </Section>
        <Section xsi:type="ovf:AnnotationSection_Type" ovf:required="false">
            <Info>A random annotation on this service. It can be ignored</Info>
            <Annotation>Contact customer support if you have any problems</Annotation>
        </Section>
         <Section xsi:type="ovf:EulaSection_Type">
            <Info>License information for the appliance</Info>
            <License>Insert your favorite license here</License>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Info>500Mb, 1 CPU, 1 disk, 1 nic virtual machine</Info>
            <Item>
                <rasd:Caption>1 virtual CPU</rasd:Caption>
                <rasd:Description>Number of virtual CPUs</rasd:Description>
                <rasd:InstanceId>1</rasd:InstanceId>
                <rasd:ResourceType>3</rasd:ResourceType>
                <rasd:VirtualQuantity>1</rasd:VirtualQuantity>
            </Item>
            <Item>
                <rasd:Caption>256 MB of memory</rasd:Caption>
                <rasd:Description>Memory Size</rasd:Description>
                <rasd:InstanceId>2</rasd:InstanceId>
                <rasd:ResourceType>4</rasd:ResourceType>
                <rasd:AllocationUnits>MegaBytes</rasd:AllocationUnits>
                <rasd:VirtualQuantity>256</rasd:VirtualQuantity>
            </Item>
            <Item>
                <rasd:Caption>Ethernet adapter on "VM Network"</rasd:Caption>
                <rasd:InstanceId>4000</rasd:InstanceId>
                <rasd:ResourceType>10</rasd:ResourceType>
                <rasd:AutomaticAllocation>true</rasd:AutomaticAllocation>
                <rasd:Connection>VM Network</rasd:Connection>
            </Item>
            <Item>
                <rasd:Caption>Harddisk 1</rasd:Caption>
                <rasd:InstanceId>22001</rasd:InstanceId>
                <rasd:ResourceType>17</rasd:ResourceType>
                <rasd:HostResource>disk/vmdisk1</rasd:HostResource>
            </Item>
        </Section>
        <Section xsi:type="ovf:OperatingSystemSection_Type" ovf:id="73" ovf:required="false">
          <Info>Guest Operating System</Info>
          <Description> Windows 2000 Advanced Server</Description>
        </Section>
    </Content>
</ovf:Envelope>
```

Most of the descriptor is boilerplate. It starts out by describing the set of files in addition to the descriptor itself. In this case there is a single file (vmdisk1.vmdk). It then describes the set of virtual disks and the set of networks used by the appliance. Each file, disk, and network resource is given a unique identifier. These are all in separate namespaces, but the best practice is to use distinct names.

The content of the example OVF is a single virtual machine. The content contains 5 sections:

- *ProductSection,* which provides product information such as name and vendor of the appliance.
- *PropertySection*, which list a set of properties that can be used to customize the appliance. These properties will be configured at installation time of the appliance, typically by prompting the user. This is discussed in more detail below.
- *AnnotationSection,* which is a free form annotation.

- *EulaSection,* the licensing terms for the appliance. This is typically shown during install.
- *HardwareSection*, which describes the virtual hardware. This is a required section that describes the kind of virtual hardware and set of devices that the virtual machine requires. In this particular case, a fairly typical set of hardware (500 MB of guest memory, 1 CPU, 1 NIC, and one virtual disk) is specified. The network and disk identifiers from the outer sections are referenced here.
- *OperatingSystemSection*, which describes the guest operating system.

Installation Experience and Appliance Configuration

The installation experience of an OVF package depends on the virtualization platform on which the the OVF is installed. It could be command-line based, scripted, or a graphical installation wizard. The typical OVF installation tool will show .or prompt for the following information:

- Show information about the OVF package (from the ProductSection), and ask the user to accept the licensing agreement, or deal with an unattended installation.
- Validate that the virtual hardware is compatible with the specification in the OVF.
- Ask the user for the storage location of the virtual machines and what physical networks the logical networks in the OVF package should be connected to.
- Ask the user to enter the specific values for the properties configured in the *PropertySection*.

After this configuration, it is expected that the virtual machines can be successfully started to obtain (using standard procedures such as DHCP) an identity that is valid on the local network. Properties are used to prompt for specific IP network configuration and other values that are particular to the deployment environment. Once the appliance is booted for the first time, additional configuration of software inside the appliance can be done through a management interface provided by the appliance itself, such as a web interface.

Portability

The hardware description shown in the above example is as general as possible. In particular, it simply specifies that a virtual disk and a network adaptor is needed. It does not specify what the specific hardware should be. For example, a SCSI or IDE disk, or an E1000 or Vlance network card should be appropriate. More specifically, it can reasonably be assumed that if the specification is generic, then the appliance will undertake discovery of the devices present, and load relevant drivers. In this case, it must be assumed that the appliance creator has developed the appliance with a broad set of drivers, and has tested the appliance on relevant virtual hardware to ensure that it works.

If an OVF package is installed on a platform that does not offer the same hardware devices and/or categories of devices that are required by the guest OS that is included in the appliance, non-trivial and non-obvious installation failures can occur. The risk is not that the appliance will run incorrectly – more that it will fail to install and boot, and that the user will not be able to debug the problem. With this comes the risk of increased volume in customer support calls, and general customer dissatisfaction. As a result, the highest degree of certainty about the ability of an appliance to correctly install and run is when the virtual hardware specification is virtualization platform specific (eg: Xen PV 3.0.3, or VMX 3), which means that the appliance is known to install and run correctly on any of the listed virtualization platforms. Alternatively, a highly constrained and detailed virtual hardware specification can reduce the chance of incorrect execution (since the specific devices required are listed) but this will limit the number of systems upon which the appliance will correctly install.

It should be borne in mind that simplicity, robustness, and predictability of installation are key reasons that ISVs are moving to the virtual appliance model, and therefore appliance developers should create appliances for which the hardware specification is more rather than less generic, unless the appliance has very specific hardware needs. At the outset, the portability of the appliance is based on the guest OS used in the virtual machines.

Ideally, the appliance vendor will create a virtual machine that has device drivers for the virtual hardware of all of the vendor's desired target virtualization platforms. However, many virtualization platform vendors today do not distribute drivers independently to virtual appliance vendors/creators. Instead, to further simplify the management of the virtual hardware / appliance interface, the OVF model supports an explicit installation mode, in which each virtual machine is booted once right after installation, to permit localization/customization for the specific virtualization platform. This allows the virtual machine to detect the virtualization platform and install the correct set of device drivers, including any platform specific drivers (eg: VM Tools, Xen Tools) that are made available to the guest when it first re-boots (via for example, floppy or CD drives attached to the guest on first boot). In addition, for sysprepped Windows VMs, which need only re-installation and customization with naming etc, the re-boot technique allows naming and tailoring of the image to be achieved in an automated fashion.

In summary, the virtual hardware section provides 3 different dimensions of flexibility in describing the required set of hardware: i) enumerating virtual hardware generations, for vendors' virtualization platforms, ii) providing multiple alternative hardware profiles that will satisfy the appliance's needs, and iii) narrowing through the use of specific device subtypes or broadening by marking certain devices as optional. These are all orthogonal and can be used together in the same descriptor.

The first dimension enumerates the virtual hardware generations that are supported:

```
<Section xsi:type="ovf:VirtualHardwareSection_Type">
  <Info>500Mb, 1 CPU, 1 disk, 1 nic virtual machine</Info>
     <System>
        <vssd:VirtualSystemType>vmx-4, xen3</vssd:VirtualSystemType>
     </System>
     <Item>
       ...
     <Item>
     ...
</Section>
```

In the above example, it is specified that the virtual hardware supported is VMware's 4th generation hardware, or Xen's 3rd generation.

Multiple virtual hardware profiles can be specified in the same descriptor:

```
<Section xsi:type="ovf:VirtualHardwareSection_Type">
  <Info>500Mb, 1 CPU, 1 disk, 1 nic virtual machine</Info>
     <System>
        <vssd:VirtualSystemType>vmx-4 </vssd:VirtualSystemType>
     </System>
     <Item>
       ...
     <Item>
     ...
</Section>
<Section xsi:type="ovf:VirtualHardwareSection_Type">
  <Info>500Mb, 1 CPU, 1 disk, 1 nic virtual machine</Info>
     <System>
        <vssd:VirtualSystemType>xen3</vssd:VirtualSystemType>
     </System>
     <Item>
       ...
     <Item>
     ...
</Section>
```

This allows the vendor to tailor the hardware description to support different virtualization platforms and features. A specific virtualization platform may choose between any of the specific virtual hardware sections that it can support, with the assumption that the OVF installer will choose the latest or most capable feature set that is available on the local platform.

Specific device types can also be specified:

```
<Item>
  <rasd:Caption>Ethernet adapter on "VM Network"</rasd:Caption>
  <rasd:InstanceId>4000</rasd:InstanceId>
  <rasd:ResourceType>10</rasd:ResourceType>
  <rasd:ResourceSubType>VmxNet, E1000</rasd:ResourceSubType>
  <rasd:AutomaticAllocation>true</rasd:AutomaticAllocation>
  <rasd:Connection>VM Network</rasd:Connection>
</Item>
<Item>
  <rasd:Caption>SCSI Controller 0</rasd:Caption>
  <rasd:InstanceId>1000</rasd:InstanceId>
  <rasd:ResourceType>6</rasd:ResourceType>
  <rasd:ResourceSubType>LsiLogic, BusLogic</rasd:ResourceSubType>
</Item>
<Item>
  <rasd:Caption>Harddisk 1</rasd:Caption>
  <rasd:InstanceId>22001</rasd:InstanceId>
  <rasd:ResourceType>17</rasd:ResourceType>
  <rasd:HostResource>disk/vmdisk1</rasd:HostResource>
  <rasd:Parent>1000</rasd:Parent>
</Item>
<Item>
</Item>
```

In the above examples, the *ResourceSubType* is used to specify the exact devices that are supported by the guest OS in the appliance. To make an appliance VMware specific one could stipulate that the network adaptor must be a VMware VmxNet or an E1000, and that only a SCSI harddisk is supported with a controller of either LsiLogic or BusLogic.

Future Versions of the OVF Specification

The scope of OVF specification version 1.0 is the packaging and deployment phases of the virtual appliance software life cycle. The management and retirement phases are explicitly not addressed. We intend to extend OVF to assist with automated management of the management and retirement phases as a matter of priority. OVF 1.0 provides the core framework that allows workflow and system-level meta-data to be encoded, stored, and transported. In the OVF package, information can be stored that describes how the appliance is to interact with external processes and systems. Examples of such functionality are appliance upgrade, cataloging, and integrity and/or security checking, and enhanced license management.

We discuss the case of appliance upgrade in more detail below.

First, existing software packaging, installation, and upgrade tools may be used by the installed OS and application to fetch upgrades from the network and install them from within the installed appliance instance. An example would be the Windows Update mechanism, or the use of yum, apt or similar update mechanisms in Linux. This type of upgrade is expected to operate normally and does not warrant further consideration.

Second, binary patches provided by the appliance vendor may allow updates to be applied to installed appliances external to the OVF packaging. Packaging such updates independently to the OVF allows a convenient vehicle to apply changes to large numbers of virtual machines, and is particularly useful in responding to security concerns in a timely fashion.

Third, an upgraded (more recent) OVF instance/version of a previously installed virtual appliance of the same name may be used to upgrade existing installations of an OVF and its VMs. Such upgrade patches allow an appliance to be very easily upgraded through the application of a newer version of its OVF. Generally speaking, this type of upgrade requires the virtualization platform to be aware of the internal structure of the VMs in the appliance, since during execution an appliance will typically modify its virtual hard disk(s). Component-wise reinstallation of parts of an OVF therefore must be aware of the distinction between VM-instance specific state, and VM application and OS state. Separation of user and system state in an appliance is encouraged, to facilitate easy upgrade of the system components of the appliance in an automated fashion. For example, for a Linux based appliance, if all instance/user specific state is stored in a known virtualized block device or disk, or in a known sub-directory of the file system, then other parts of the appliance (system disk, or other parts of the file system) could be upgraded without risking loss of instance specific data. Notice too the distinction between separation of state at the block device level and the file system level: the former is straightforward since the upgrade can be accomplished simply through block copies, whereas the latter requires the virtualization platform to understand and have the ability to modify the file system of the VM. This is not always possible and therefore this type of upgrade (or indeed any pre/post installation configuration of the VM that requires file-system level access) would require that the appliance or upgrade pack provide a "helper" appliance or VM that can upgrade the file system of the appliance appropriately, without requiring the hypervisor itself to understand the file system of the VM. Such helper appliances can also be useful for fixing a broken appliance – for example WinPE can be used to fix the file system of corrupted Windows VMs.

The OVF specification specifies an Environment Document Protocol and an Installation Feedback Protocol, but the transport format for these protocols is not specified. Possible transports includes DVD images, floppy images, XenBus, Microsoft VMBus, VMware Backdoor, and so on. A future version of the OVF specification could standardize on one or more transport formats. The split between transport and protocol provided in the current specification allows guest software to isolate most core logic from the details of the communication channel.

Conclusion

The OVF specification offers a portable virtual appliance format that is intended for broad adoption across the IT industry. The OVF specification is intended to be immediately useful, to solve an immediate business need, and to facilitate the rapid adoption of a common, backwards compatible, yet rich virtual machine format. OVF is complementary to existing IT management standards and frameworks, and will be further developed within a standards organization. Concretely the OVF specification embraces the diversity of virtual hard disk formats available today, and in so doing reduces users' perceived risk of proprietary "lock in". OVF promotes customer confidence through the collaborative development of common standards for portability and interchange of virtual machines between different vendors' virtualization platforms, and promotes best-of-breed competition through its openness and extensibility.

The OVF specification version 1.0 focuses on the immediate need to develop a common standard for "portability from origin" of virtual machines. An OVF 1.0 compliant appliance that offers type 3 compatibility will correctly install and run on any OVF 1.0 compliant virtualization platform. The limited scope of OVF 1.0 allows the industry to unite around a key, simple use case for portability, and to collaborate in a recognized standards organization to develop specifications that solve more complex use cases, such as V2V.

The OVF specification is intended to evolve in an appropriate standards organization. However, it is the intention of the authors to ensure that the first version of the specification is implemented in their products, and so the vendors of virtual appliances and other ISV enablement, can develop to this version of the specification. Future versions will be backwards compatible with this version to the maximum extent possible. Vendors are welcome to make proposals for the evolution of this format, and to present them. The explicit copyright notice attached to this document is intended to avoid arbitrary piece-wise extensions to the format outside the context of a standards organization, while permitting free distribution and implementation of the specification by ISVs and the open source community.

Acknowledgements

The OVF 1.0 contributors from XenSource recognize with thanks the discussions and suggestions from members of the Xen community that helped to frame many of the concepts modeled in the OVF document.

The OVF 1.0 contributors from VMware recognize with thanks the discussions and feedback from our partners.

Xen Source

VMware, Inc.

IBM

HP

Dell

Microsoft