Open Virtual Appliance Specification

Version 1

Private Preview Release
DO NOT CIRCULATE
June 22, 2006

Ewan Mellor <Ewan.Mellor@xensource.com>
Andrew Warfield <Andrew.Warfield@xensource.com>

Executive Summary

Virtual machines have attracted a great deal of attention lately for their potential to achieve better utilization of computer hardware while simultaneously isolating coarse-grained units of application software such as web servers, virus scanners, and email clients. Isolating applications into individual VMs allows failure to be contained, enhances system security, and eases administration. Virtual machines provide a means to build ***application appliances***: complete applications, including customized Oses, may be placed inside VM containers and administered as black boxes. Software developers may construct such VM-based appliances and ship completely customized software stacks to their customers.

This document addresses an immediate concern facing the success of application appliance-based software distribution: While there are currently several popular virtual machine monitors, each with a dissimilar ***run-time*** format for VM resources such as hard disk images and system configuration, there does not currently exist a ***transport*** format for the deployment of VM-based software. A transport format is required to allow packaged VMs to be distributed to, customized for, and installed on, the various VMMs that are currently available. Runtime formats (such as VMware's VMDK and Microsoft's VHD) are simply the formats used by their associated VMMs to represent a hard disk and do not provide any details as to the customization or installation of the associated VMs.

This document details a specification for the **Open Virtual Appliance (OVA)** format. OVA is a packaging format for virtual machines that allows *virtual machine templates* to be distributed, customized, and instantiated on any OVA-supporting VMM. An OVA contains a complete application appliance, potentially composed of multiple virtual machines.

In essence, the Open Virtual Appliance format aims to establish a common standard for VM logistics. To illustrate this point, this specification details the construction of an OVA containing a CRM appliance. The application example allows a user to customize and install a collection of three virtual machines, a CRM application, an Apache web server, and a MySQL backend.

1 Contents

2 Introduction

This document describes and specifies the Open Virtual Appliance specification, also known as OVA. This specification may be openly copied, distributed, and used to guide the development of software, open source or proprietary, as dictated by the terms of the GNU Free Documentation License.

From the user's point of view, an OVA is a **packaging format for software appliances**. Once installed, an OVA adds to their cluster a self-contained, self-consistent, complete software solution for achieving a particular goal. An OVA may 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 OVA is a **transport** mechanism for **virtual machine templates**. One OVA may contain a single VM, or many VMs (it is left to the software appliance developer to decide which arrangement best suits their application).

OVAs must be installed before they can be run; a VM will never run directly out of an OVA.

Note that, as a transport mechanism, OVA differs from VMware's Virtual Disk Format and Microsoft's Virtual Hard Disk format. Both of those are run-time VM image formats, operating at the scope of a single VM disk, and though they are being used as a transport mechanism, they are not designed to solve that problem; they don't help you if you have a VM with multiple disks, or multiple VMs, or need customization at install time, et cetera.

3 VM Life Cycle

1. As VMs are increasingly used to package, deploy, and run software applications it is important to recognize that there exists a general life cycle that applies to VMs, and within which VM administration must be considered.
2. This life cycle is shown in Illustration 1, and divided into four broad stages: **Development**, **Deployment**, **Maintenance**, and **Removal**. A VM instance will spend the majority of its life in the Maintenance phase where it has been installed into a VMM environment and runs normally. The OVA specification described here aims to establish a common, flexible packaging format for VMs that guide their management throughout all four stages of this life cycle.

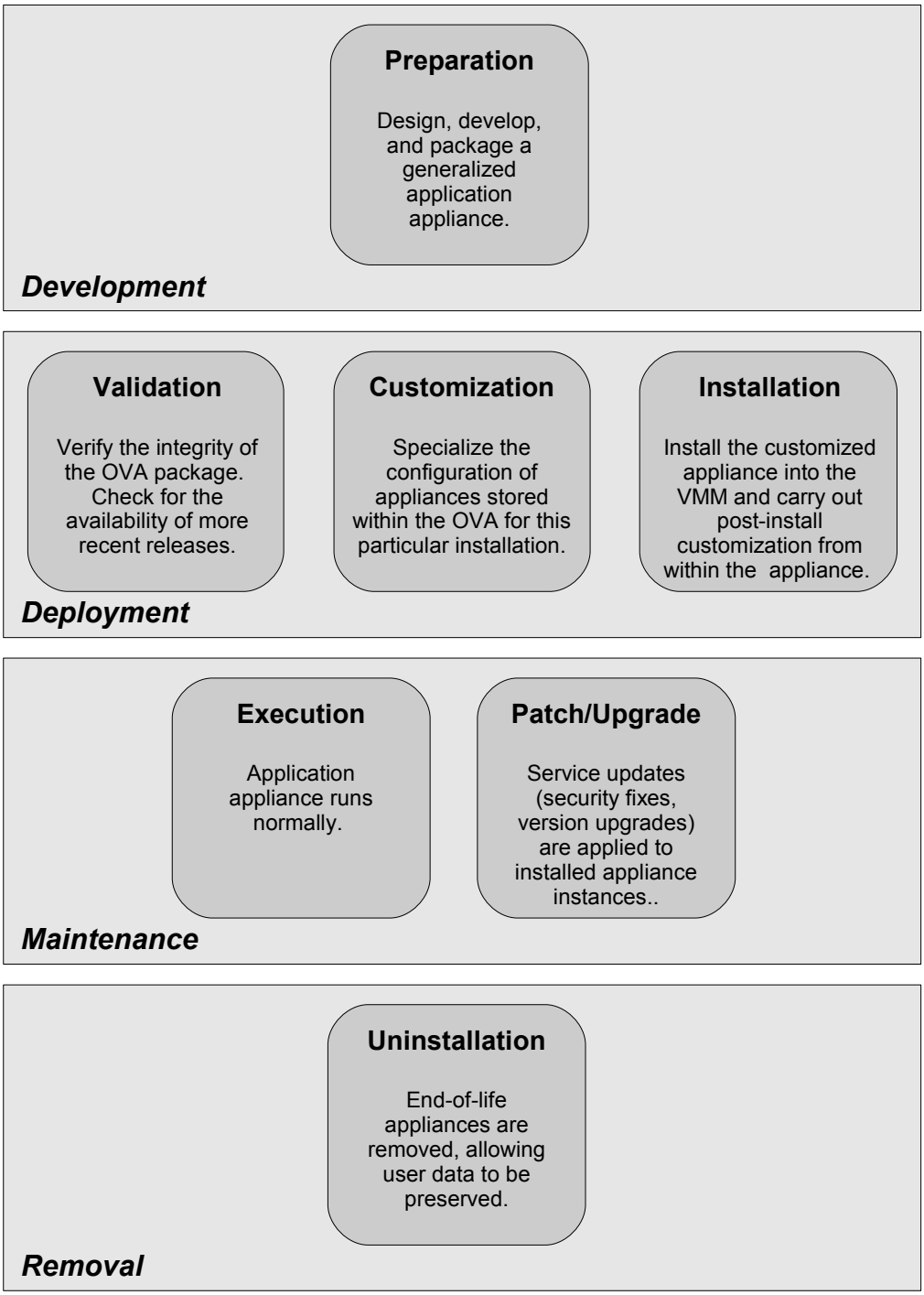

Illustration 1: VM life cycle

3. In the **Development** stage, an OVA is constructed based on one or more existing VMs. Commonly this will involve the construction of a "model" appliance that is to be packaged and distributed. The OVA specification allows such model VMs to be *generalized*: fixed configuration options in the model VM such as resource limits and network configuration may be associated with parameters that will be set when the OVA is used to install a new appliance instance. We anticipate that tool support will become available to assist in the packaging and

generalization of model appliances into OVAs. At the end of the development stage, a completed OVA package is ready for distribution.

4. The **Deployment** stage involves the use of an OVA package to instantiate a new application appliance instance on a given VMM-based system. This stage involves a number of phases. First, the OVA is ***validated*** for integrity and the installation tools may check for the availability of a newer version of the OVA. The OVA parameters are then set through interactions with system presents and the system administrator in order to ***customize*** the instance that is being created. Finally, the new instance is ***installed*** into the system, and the new appliance is started, where post-install configuration takes place.
5. Most of an appliance's life time is spent during the **Maintenance** stage, where it runs in the system providing functionality to users. The OVA specification is largely unconcerned with this stage except with regard to the patching and upgrade of installed applications in order to fix security vulnerabilities or provide new functionality. These concerns are achieved by allowing the clear separation of user and system data within an appliance description, and allowing installed appliance images to be upgraded.
6. The final stage of an appliances life cycle is that of **Removal**, in which the appliance is uninstalled from the system. The concern of the OVA specification with regard to this stage is two-fold. First, we ensure that user data generated by the appliance may be preserved safely. Secondly, the appliance must be ***completely*** removed from the system, including all configuration data installed during the deployment stage.
7. To illustrate the OVA specification, the remainder of this document focuses primarily on the deployment stage, in which a new appliance is installed from an OVA. This allows a detailed examination of the OVA format, which present the results of the development stage intuitively. As the OVA format is intended to provide a standard which may be applied across a variety of VMMs, we have attempted to avoid presenting Xen-specific installation details wherever possible.

4 Installation Overview

1. When an end user passes an OVA to an installer, any or all of the following things happen:
    1. The OVA is checked for integrity.
    2. The end user is prompted for configuration parameters.
    3. One or more virtual machines is created. The filesystems for each virtual machine are created by one of:
        1. creating a filesystem based upon data inside the OVA itself;
        2. downloading filesystem contents from some preconfigured or discovered location;
        3. prompting the user for a filesystem location, and then downloading from there.

4. In any of the above cases, the filesystem may be created by a direct copy of data blocks, or by creating a filesystem and then populating it by copying files into the new filesystem. The difference here is of no concern to the end user, but will be of interest to the person creating the OVA.
5. The end user may have been prompted for the filesystem size, or for a parameter that indirectly determines the filesystem size. If the filesystem is to be populated by copying files, then the specified size is honoured when the filesystem is created. If the filesystem is populated by copying blocks, then the filesystem is resized after creation as part of the install. If the filesystem cannot be resized by the installer, then the size specification cannot be honoured, and the installer, should fail the operation.

5 OVA Format Specification

5.1 Example Case: *eSpeil Solutions*

This section describes the OVA package format through the use of an example case. eSpeil Solutions, an imaginary appliance vendor, has prepared an OVA containing a customer relationship management (CRM) solution. The appliance includes three virtual machines: the eSpeil CRM application, and support VMs containing an Apache web server, and a MySQL database. During deployment, the OVA customizes system details such as the address of the NTP server to use, and installs the appliance onto the system.

5.2 Overview

1. An OVA is a Zip file (aka Info-ZIP or PKZIP).
2. Inside the Zip file there is a file called ova.xml, inside the root directory.
3. The Zip may contain any number of other files, but no other file inside the Zip is prescribed. All other files are referenced by ova.xml, so their name or location need not be prescribed by this specification.

5.3 ova.xml

5.3.1 Example

```
<?xml version="1" ?>
<!DOCTYPE appliance PUBLIC "-//OVA//DTD OVA 1//EN"
          "http://www.openvirtualappliance.org/schema/ova-1.dtd">
<appliance xmlns='http://www.openvirtualappliance.org/schema'
           version="1">
  <label xml:lang=”en”>eSpiel Sales Solution</label>
  <version>2.1</version>
  <shortdesc xml:lang=”en”>eSpiel Sales Solution</shortdesc>
  <longdesc xml:lang=”en”>
    eSpiel Sales Solution: A CRM, database and webserver packaged
    solution.  http://www.espiel-sales.com/
  </longdesc>
  <detail>
Copyright &copy; 2006 eSpiel Sales Solution Inc.
...
  </detail>

  <manifest     src="manifest.txt" />
  <signature    src="signature.asc" />
  <mf-signature src="mf-signature.asc" />

  <vm name="crm">
    <label xml:lang=”en”>eSpiel CRM</label>
    <shortdesc xml:lang=”en”>eSpiel Sales Solution CRM</shortdesc>
    <longdesc xml:lang=”en”>
      eSpiel Sales Solution CRM, running on Debian 3.1, packaged and
      tested by eSpiel.  http://www.espiel-sales.com/
    </longdesc>
    <detail>
Copyright &copy; 2006 eSpiel Sales Solution Inc.
Debian 3.1r2, Mon 17 Apr, 2006.
...
    </detail>
    <config file="crm.cfg" />
    <vbd fs="crm-root" variety="system" device=”sda1” mode="w" />
    <vbd fs=”backup”   variety="user" device=”sda2” mode="w" />
  </vm>

  <vm name="mysql">
    <label xml:lang="en">eSpiel MySQL</label>
    ...
    <vbd fs="mysql-C" variety="system" device="C:" mode="w" />
  </vm>

  <vm>
    <label xml:lang="en">eSpiel Apache</label>
    ...
    <vbd fs="apache-root"  variety="system"    device=”sda1” mode="r" />
    <vbd fs="backup"       variety="user"      device="sda2" mode="w" />
    <vbd fs="apache-cache" variety="ephemeral" device="sda3" mode="w" />
  </vm>
```

```
  <fs name="crm-root" source="file:///crm.qcow">
    <label xml:lang=”en”>eSpiel CRM Root filesystem</label>
    <shortdesc xml:lang=”en”>...</shortdesc>
    <longdesc xml:lang=”en”>...</longdesc>
    <detail>...</detail>
  </fs>
  <fs name="mysql-C" source="file:///mysql.vhd">
    ...
  </fs>
  <fs name="apache-root"
      source="file:///apache-root.tar.bz2"
      type=”ext3” size=”${apache-root-fs-size}”>
    ...
  </fs>
  <fs name="backup" source="file:///backup.qcow">
    ...
  </fs>
  <fs>
  <fs name="apache-cache" type="ext3" size="500MiB">
    ...
  </fs>

  <properties>
    <property name="ntp-server" source="prompt:ntp-server" />
    <property name=”mysql-root-fs-size” source=”prompt:mysql” />
    <property name=”apache-root-fs-size" source=”script:apache” />
    <property name=”apache-webspace"
              source=”script:apache-webspace” />
    <property name=”apache-logspace"
              source=”script:apache-logspace” />
  </properties>

  <prompts>
    <prompt name="ntp-server" default="pool.ntp.org">
      <shortdesc lang="en">
        Please specify the address for your NTP server, or use
        pool.ntp.org if you do not have an NTP server
        locally.
      </shortdesc>
    </prompt>
    <prompt name="mysql" default="2 GB">
      <shortdesc xml:lang=”en”>MySQL filesystem size</shortdesc>
    </prompt>
    <prompt name="apache-webspace">...</prompt>
    <prompt name="apache-logspace">...</prompt>
  </prompts>

  <prompt-values>
    <prompt-value param=”ntp-server” value="ntp.intra" fixed="no" set-
by="Claire Watts" set-on="2006/04/31 12:13:34" />
  </prompt-values>

  <scripts>
    <script name=”apache” script=”scripts/apache-root-fs-size.sh”>
      <param>${apache-webspace}</param>
```

```
      <param>${apache-logspace}</param>
    </script>

    <script name="registration" script="scripts/registration.sh">
    </script>

    <aux-file>scripts/common.sh</aux-file>
  </scripts>

  <hooks>
    <hook event=”post-install” script=”registration” />
  </hooks>

  <admin vm="crm" port="8088" path="/" />

</appliance>
```

5.3.2 Elements

<appliance>

1. The version number here is the version of this specification to which the OVA is constructed. This specification only currently defines version 1.

<version>

2. The version number here is the version of the appliance itself.

<label>
<shortdesc>
<longdesc>
<detail>

3. The label, shortdesc, longdesc, and detail elements are for the end user's benefit, and will be presented by the UI as appropriate. The detail element should include all the necessary copyright, licence, and distribution details, as well as all the version details necessary for the end user to check whether the appliance is vulnerable to any known security holes.

<manifest>

4. The manifest element specifies a single file that contains a list of entries, one entry per line, giving for each file in the OVA the SHA1 hash of that file, followed by the filename itself, separated by whitespace. (This is the output format used by the sha1sum tool.)
5. An OVA-compliant installer **must** check the integrity of each file in the OVA against the given SHA1 hash, and **must** refuse to start any VM in the OVA if the integrity check fails. It **must** also refuse to use or copy any file from the OVA if that file is not listed in the manifest.

<signature>
<mf-signature>

6. The signature and mf-signature elements specify one file each giving a cryptographic signature for the ova.xml file and the manifest file respectively.
7. The signature should be an OpenPGP-compliant signature, ASCII armoured.

8. An OVA-compliant installer **must** check the validity of the two signatures. If there is no established chain of trust for the signatures, then the end user **must** be prompted, giving all the information in the signature, and the end user must be allowed to make the trust decision themselves. If the validity check fails or the chain of trust cannot be established and the user refuses to validate the trust, then the installer **must** abort the installation.
9. If possible, the OVA installer **must** check wwwkeys.pgp.net (or one of its mirrors) for the key used in the signature, to check whether that key has been revoked. If the key has been revoked, the installer **must** abort the installation.
10. If it is not possible for the OVA installer to check wwwkeys.pgp.net, for example because the destination host is not connected to the network, then the installer must inform the end user of that fact, and allow them to abort the installation.
11. An OVA author **must** upload their key to wwwkeys.pgp.net if they distribute their OVA, so that OVA recipients can check whether that key has been revoked.

    <vm>
12. The name attribute specifies a value that can be used to refer to this VM during the install. This name must be unique within the OVA, and contain only the ASCII alphanumeric, minus and underscore characters.

    <config>
13. The file attribute gives the configuration file for the VM that, in Xen installations, will be given to Xend when the domain is started.

    <vbd>
14. A specification of a virtual block device (VBD) for the VM. This refers to a particular named filesystem (specified later), and specifies the device within the guest to which that filesystem will be exported. This attribute takes a form similar to “sda1” for Linux-based VMs, and to “C:” for Windows-based ones.
15. The mode attribute may be either “r” or “w”, indicating that the device will exported to the guest in read-only or read-write mode respectively.
16. The variety attribute provides guidance to the VMM and associated management tools has to how the disks associated with an appliance are to be treated. There are three varieties of vbd: `system` disks contain appliance software and may be completely replaced in the event of an upgrade. `user` disks contain user data (for instance saved documents) that should be preserved across upgrades. Additionally, on uninstalling an appliance, the user should be given the option of preserving `user` disks rather than deleting them. Finally, ephemeral disks contain temporary data that need not be preserved across VM restarts. The VMM is free to allocate these disks on arbitrary storage and may delete them at VM shutdown.

    - Note that a single filesystem may be referenced by multiple VBDs (as shown by the “backup” filesystem in the example). As long as the access is read-only, this is safe with all filesystems. If any of the VMs use that

filesystem in read-write mode, the filesystem itself must be one designed for such use, such as OCFS2.

<fs>

17. A specification of a filesystem. This filesystem must be written to a storage substrate that will be exported to the VMs as specified by the per-VM <vbd> entries.
18. The name attribute must be of the same form as specified for the <vm> element, above. It is used by <vbd> elements for reference, and used to construct certain paths during installation, as detailed later. Each filesystem must have a unique name, of course.
19. The source for the filesystem is specified by the URI given by the source attribute. If this is a file: URI, then this is relative to the OVA root. Otherwise, it specifies a network resource, as usual.
20. Each source may be packaged using a virtual disk format (subject to installer support, of course). The best known non-proprietary virtual disk format is QEMU's QCOW, but sources could just as easily be packaged using VMware's Virtual Disk Format, Microsoft's Virtual Hard Disk format, or any similar format. With any of those disk formats, the blocks will be copied directly to the storage substrate, and there is no need to specify the FS size, because this will be taken from the size of the source.
21. As opposed to using a virtual disk format, the source may instead be a tar file (optionally compressed). In this case, the filesystem type and size must be specified. The filesystem will be created at install time, and then populated from the given tar. In this case, the filesystem must be on the Supported Filesystem List (which is not necessary when using virtual disk formats).
22. If the source is missing and the filesystem type and size are present, then an empty filesystem is created. Again, the filesystem must be understood by the installer in this case.
23. If the source and the filesystem type are missing, but the size is present, then an unformatted block device is provided, and it is the VM's responsibility to format that appropriately.

<properties>

24. Each property is declared by name, which must be unique across properties, and declared to have a particular source. The source may either be of the form "`prompt:<name>`" or "`script:<name>`", declaring that the value is obtained by prompting the user at install-time, or by running a script. In either case, the name given is a reference to an `<prompt> or <script>`, declared separately.

<prompts>

25. The prompts section gives a simple specification for prompts that can be given to users in order to receive install-time parameters. The semantics for each of the fields should be clear.

<prompt-values>

26. Prompt values may be 'late-bound', i.e. specified by the end user or a third party to create a new OVA. This allows them to specialise or partially specialise an OVA for their particular cluster, for example. These overriding values are separated out so that the history of OVA modification is clear.
27. In the example, the prompt-value line specifies that the property "ntp-server" has been set to have a value of "ntp.intra" and that this value is only a default (fixed="no"). The normal prompt for that value will still take place, but including a default of the given value. With fixed=”yes”, the prompt would not appear at all.

    <scripts>
28. Scripts are declared in the <scripts> section. The script itself is named, relative to the root of the OVA, and will be executed in the environment discussed in a later section. The param elements specify the command line parameters for the script, and may include property expansions, as shown in the example.
29. The script must exit with 0 for success, as usual, and print <name>=<value> pairs for each of the properties that it defines.
30. Given that OVA properties may be defined as the result of script execution, and scripts may take property expansions, there is an implicit script execution order, which will be discovered at install-time. If there is any cycle in the dependencies, then the install will be rejected. If the ordering between any two scripts is not constrained by property dependencies in this way, these scripts will be executed in the order that they appear in the <scripts> section.

    <aux-file>
31. The aux-file element names a file, relative to the root of the OVA. This file will be made available within the scripts' execution environment. Any number of aux-files may be named. It is not necessary to name any file referenced by the <script> element.

    <hooks>
32. Scripts may be run automatically during install and uninstall. These are specified as hooks on certain events, and reference named scripts declared in the <scripts> section.
33. Event may be “pre-install”, “post-install”, “pre-uninstall”, “post-uninstall”.

    <admin>
34. The admin element specifies a virtual machine by name, a TCP port and path. These three attributes, given the address of the virtual machine, are combined to give an HTTP URL. At this URL, there should be a web interface that can be used to administer the appliance.

5.3.3 DTD

1. Pending.

6 Script Execution Environment

1. Each of the scripts specified in the <scripts> section will be executed in a tightly controlled environment. A guest domain will be created, with a standard Linux-style (LSB) directory layout.
2. The Perl interpreter will be present. There will be a variable in the script's environment named OVA_SESSION which gives a session key pre-authenticated as the user installing this software.
3. Every file named using an aux-file element will be available under /var/lib/ova/aux-files.
4. The block devices for the VMs will be available as `/var/lib/ova/vm/`*<VM name>*`/`*<device name>*, where *<VM name>* is the value specified using the <vm> element's name attribute, and *<device name>* is the value specified as the device attribute of the <vbd> element (with the colon removed, if any). For example, the root device for the eSpiel CRM virtual machine would be at `/var/lib/ova/vm/crm/sda1`, and for the database server would be at `/var/lib/ova/vm/mysql/C`. These will be block device special files, or symbolic links to them, and may be written through. The filesystem specified using the <fs> element will already be in place, unless this script is running off a pre-install or post-uninstall hook.
5. Filesystems will also be available through `/var/lib/ova/fs/`*<name>*, where name is the attribute given on the <fs> element.
6. The VM's new filesystems may be mounted within the installation guest if the filesystem thereon is in the Supported Filesystem List.
7. The /tmp, /mnt, /var/lock, and /var/log directories of the script's guest domain will be writable; all other directories will be read-only.
8. The writable directories **may** be used by the scripts to communicate values from one to another. For example, one script may place a cache of auto-detected values, and this may then be used by later scripts, to avoid repeating the detection process.
9. The script will be run with stdout redirected to an unspecified file; the installer will use the script output to set new property values. The script's stderr will be redirected to `/var/log/ova/`*<VM name>*`/`*<script name>*. stdin will be /dev/null.
10. The contents of /var/log will be collected after the installation, and saved. The filesystem backing the writable directories will then be discarded.
11. A script may be terminated at any time, on the instruction of the user, by shutting down and then destroying the whole guest. This will abort the whole install. If an OVA script is expected to take an unusual length of time to execute, that should be mentioned in the accompanying documentation.

    - This approach has the benefit of providing a strongly isolated but highly flexible means for OVAs to customize appliances as they are installed. A linux-based execution

environment with support for scripting languages is very lightweight and should be applicable even for the customization of non-linux VMs, although if this proves inappropriate, alternate scripting environments (e.g. Windows) may be added in the future.

7 Post-installation

7.1 Store Entries

1. The first time a VM is run, sometime after installation, the OVA installer will place entries into the Xen store detailing the properties obtained or calculated during the install. This allows the guest to act upon those answers, to complete the installation.
2. A single entry `/local/domain/<domid>/ova = <path to OVA section>` will be written. The guest will be able to reference this using the relative path "`ova`", without needing to know its own domain ID. Using that path as a prefix, the directory "`properties`" can be found, and inside that directory, each of the key-value pairs given for each of the OVA's properties can be found.
3. In the given example, the domain will be able to set its NTP server configuration by using the following code:

```
ovapath=$(xenstore-read ova)
server=$(xenstore-read "$ovapath/properties/ntp-server")
```

   - Note that in this example, XenStore is simply being used to provide a mechanism to access a hierarchical XML install-time data structure. Comparable mechanisms may be provided using the established control channels available on other popular hypervisors such as VMware and Virtual Server.

7.2 Rendevous

1. It is assumed that each VM in an appliance will in general need to be able to contact each other.
2. In order to achieve this, the user interface must enforce the fact that every VM is attached to at least one network in common across the appliance.
3. If the VM is configured to take its address from DHCP, the DHCP server will be interrogated for the VM's fully qualified domain name, using the MAC address allocated to the VM's interface on the common network as the `dhcp-client-identifier`.
4. If the VM is configured to have a static IP address on the common network, then that IP address will be used instead.
5. `On Xen, whether obtained from the DHCP server or directly from the VM's configuration, the address of each VM in the appliance on the common`

network will be placed in the Xen store, accessible by every VM at boot time. The value will be placed at $ovapath/vm/*<VM name>*/address. As in section 7.1 above, comparable approaches are possible on other VMMs.

6. Each VM may place a watch on these paths in the store, and pick up the new address when the VM's configuration is changed.
7. Alternatively, the VMs could use Zeroconf or other rendezvous technology, or they could negotiate using shared storage with a cluster filesystem. Either of these are in-guest issues, and not of specific concern to this specification.

8 Appliance Upgrade

1. Appliances may be upgraded in one of three ways. First, existing software packaging, installation, and upgrade tools may be used by the installed application to fetch upgrades from the network and install them from *within* the installed appliance instance. Second, binary patches or hotfixes may be applied to installed appliances. Third new, upgrade OVAs may be applied to installed appliances.
2. In-VM upgrade tools such as application-specific upgrade managers, the Windows patch tools, and Linux package managers such as `yum` and `apt` may be used to maintain the contents of an installed appliance. These tools function as any other VM-contained application and require no further mention in this context.
3. Binary patches and hotfixes may be provided by appliance vendors to allow updates to be applied to installed appliances externally. Packaging such updates externally 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. The binary patch format is not included as a portion of the OVA specification and will appear in a follow-up document.
4. OVA upgrade patches allow an appliance to be upgraded through the application of a newer OVA version. Generally speaking, this upgrade will reinstall all `system` vbds, erase all `ephemeral` vbds, and preserve the contents of all `user` vbds. Any post-processing (e.g. file format conversion of user data) should be done in install scripts.

9 Supported Filesystem List

1. The following filesystems are supported by the OVA installer and the script execution environment. Compliant installers **must** support creation and mount of these filesystems.
    - FAT
    - VFAT
    - NTFS
    - ext2
    - ext3

- reiserfs4
- ocfs2
- ISO 9660
- ISO/IEC 13346 (UDF)

10 Rationale

10.1 Zip

1. It is necessary to be able to extract the ova.xml file and the manifest without extracting the whole OVA, so that the end user may inspect the OVAs using appropriate tools, without experiencing delay.
2. tar definitely does not meet this requirement (one may append replacement files to a tarball, so tar must scan the whole archive, even if the file requested is found at the front of it). cpio seems not to meet this requirement. No other widely available archive formats have been suggested.
3. There are no copyright / patent / etc issues with Zip (or any of the other options mentioned here). The zip/unzip programs are from Info-ZIP, and though compatible with the commercial PKZIP from PKWARE, are open-source and unencumbered.
4. Note that we are not using the Zip format for compression, but merely as a packaging technology. The compression used for the Zip is unspecified, and left to the OVA author, but it is expected that it will be common for no compression to be used (`zip -0`) , with the compression of the disk images being handled by the disk transport formats themselves.

11 Acknowledgments

1. The idea to tag filesystems with a mode (“system”, “user”, or “ephemeral”) to allow disk-based upgrades comes from [Chandra et al].

12 Bibliography

1. [Chandra et al] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam.
   2. *The Collective: A Cache-Based System Management Architecture*. In Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI 2005), pages 259-272, May 2005.

13 Missing / Incomplete

1. VM configuration is not yet complete and is currently under revision.
2. Library support for cross-VMM access to store-like control data.
3. Tags for user disks.
4. In-guest monitoring.
5. Disk overlays (through scripts).
6. Specific discussion of P2V.

Comments welcome!