Department of Electronics and Computer Engineering
UNIVERSITY OF WATERLOO

Delivering IBM® Tivoli Provisioning Manager as Virtual Appliance

August 2010
Alice Yeung

Abstract

Due to environment heterogeneity and component dependency, deployment of enterprise software solutions becomes more and more complex. IBM Tivoli Provisioning Manager is one of the enterprise software products that has difficult, lengthy and error-prone deployment process. An optimal solution to resolve the deployment issue of IBM Tivoli Provisioning Manager is the use of virtual appliance. This paper provides a detailed explanation of virtual appliance, its benefits and applicability and reviews various virtual appliance architectures. The paper then discusses the IBM® Tivoli Provisioning Manager (TPM) product, the difficulties of deploying TPM and proposes a solution using virtual appliance to address the weakness and challenges of the current deployment process.

Table of Contents

List of Figures

List of Tables

1. Introduction

As software solutions become more powerful and complex, software deployment becomes more difficult and expensive. Software deployment, which is the process of making a software solution ready for use, often includes deploying multiple, interrelated software components into heterogeneous environments. With increasing complexity of deployment due to environment heterogeneity and service dependency, datacenters are facing with scalability, manageability, and consumability challenges that require breakthrough technology for data center design and management.

Software appliance is an evolution to simplify software development, distribution, management and maintenance. A software appliance is a tightly integrated package of enterprise application software and an operating system, designed to run on a standard industry platform. If the appliance is designed to run on a physical server, it is a software appliance; if it is designed to run on a virtualization platform, it is a special kind of software appliance called virtual appliance.

Virtualization helps reduce operational cost by consolidating servers to improve server utilization, and to reduce storage space and power consumption. Software appliances simplify software deployment by encapsulating entire custom environments, and resolving the execution inter-dependencies through pre-installing and pre-configuring the software applications. By combining the

advantage of both virtualization technology and software appliance, virtual appliance packages eliminate the need of manual configuration and relieve management burden.

Similar to many enterprise software products, IBM Tivoli Provisioning Manager is highly complex and deploying this product presents many weakness and challenges. The current deployment process of IBM Tivoli Provisioning manager is difficult and error-prone due to large number of environment constraints imposed and large number of manual steps involved. Using virtual appliance as IBM Tivoli Provisioning Manager's delivery mechanism is a new and effective way to solve current issues.

This paper is organized as follows. First, a survey of current research in the area of virtual appliances is provided in Chapter 2. The definition, benefits and architecture of virtual appliances are discussed in the survey. In Chapter 3, a commercial product called IBM Tivoli Provisioning Manager is introduced and its current deployment mechanism is analyzed. To address the weakness and challenges of the current IBM Tivoli Provisioning Manager deployment process, a solution to deliver the product as virtual appliance is proposed in Chapter 4. Lastly, future work is discussed in Chapter 5 and Chapter 6 concludes the paper.

2. Virtual Appliance

This section provides a survey of current research on virtual appliance.

2.1 Definition

In order to understand the essence of virtual appliance, one must first understand the environment which it operates under. Virtual machine, which is an independent runtime entity that consists of virtual hardware specifications, an operating system, and a set of applications, is the backbone where virtual appliances are utilized. Virtual appliance is a software solution delivery mechanism which utilizes one or more virtual machines. Virtual appliance, in other words, is a pre-built, pre-configured, ready-to-run software solution, comprising one or more virtual machines. Each virtual appliance contains all components that are required to run the solution on top of a virtualization layer.

The applicability of virtual appliances is widespread - many applications that are accessible over a network can be delivered as virtual appliances. Examples include a defect tracking tool, a project management tool, and a complete solution that is composed of web, application and data store tiers. Each virtual appliance is packaged, deployed, maintained, updated and managed as a single unit. Virtual appliances encapsulate entire custom environment, resolve execution policy constraints and inter-component dependencies through pre-installing and pre-configuring the software applications. Delivering complex software systems and

services as a pre-configured software stack can dramatically increase robustness and simplify installation.

2.2 Benefits

Virtual appliances offer a new paradigm for software delivery, which all parties involved in the process are beneficiaries.

For application developers, it is simpler to build software that will be run on environments that are under control, and do not have to worry about various environment configurations that end users might demand. Patching software becomes simpler as well because developers can ensure compatibility between applications, operating system and virtual machine components while distributing updates to customers as a single pre-configured package. Moreover, developers can maintain tighter control over the quality and security of the products as content verification, integrity checking, software licensing management can all be done before the software is delivered to the customer such that the burden of accessing customer's hardware components is removed.

For software providers, developing and distributing enterprise software as virtual appliances is more cost effective than conventional hardware-based solutions. With virtual appliances, the burden of managing physical inventory and supporting hardware components is replaced by packaging the software solution and virtual

hardware together. The need for hardware testing, as well as application and operating system compatibility testing are therefore reduced.

For software vendors, virtual appliances reduce the amount of time it takes to configure, package and distribute software solutions, and accelerate sales cycles by eliminating the time and complexity that customers face in evaluating software. Virtual appliances can also be distributed online, which helps vendors increase market awareness, target new accounts, and engage customers that would not normally evaluate or purchase hardware-based solutions.

For customers, enterprise software deployment and configuration are normally complex, lengthy, error prone and expensive. With virtual appliances, customers can quickly deploy software solution as an integrated unit, thereby decrease the time to value of the solution purchased. Furthermore, virtual appliances can also simplify IT management by allowing customers to manage and maintain a single encapsulated solution, rather than a disparate set of applications, operating systems and server hardware. Customers of virtual appliances no longer need to contact different vendors for support; instead, a single vendor will be able to support all components in their appliance solution.

2.3 Architecture

In this section, an overview of virtual appliance architecture is provided, and detail design of the components involved is discussed.

2.3.1 Overview

Figure 1 illustrates a general architecture of a single-node virtual appliance. The virtual machine in the appliance contains a set of virtual hardware specifications, a guest operating system, and a set of applications to provide a service.

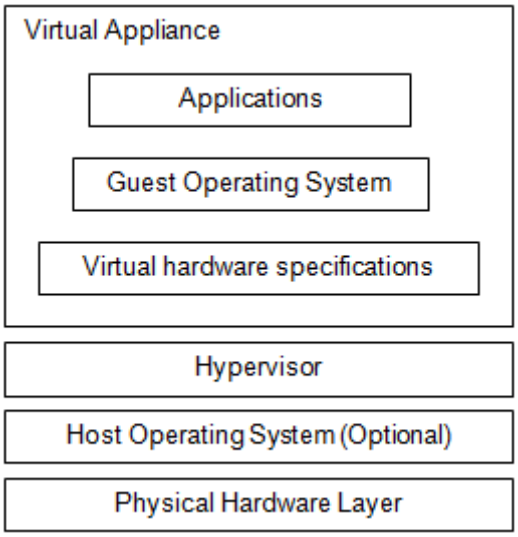

Figure 1: Single-Node Virtual Appliance

The creation phase of virtual appliances contains the following steps:

1. Install a guest operating system.
2. Install an application or a suite of applications, such as database server, directory server, application server, and applications.

3. Identify parameters that can be or need to be configured during the deployment of the virtual appliance.
4. Provide an interface or mechanism to configure the virtual appliance.

After a virtual appliance is created, it encapsulates the entire custom environment, including the execution policy constraints and inter-dependencies among the software components.

During the deployment of a virtual appliance, configurations need to be set up appropriately both inside and outside the appliance. Configuration outside the virtual appliance refers to the configuration of the virtual hardware resource, e.g. the number of virtual CPUs, memory size, virtual disk size, and network parameters. The interface provided by the hypervisor can be used for this purpose. Configuration inside the virtual appliance, on the other hand, refers to the configuration of the software installed on the virtual machines. For multi-node virtual appliances, an appliance agent is added on each of the virtual machines to assist in the configuration. These appliance agents are embedded inside the virtual machines to automate the activation of the virtual appliance and to coordinate the guest operating systems and applications configurations. The architecture of multi-node appliances with appliance agents is illustrated in figure 2.

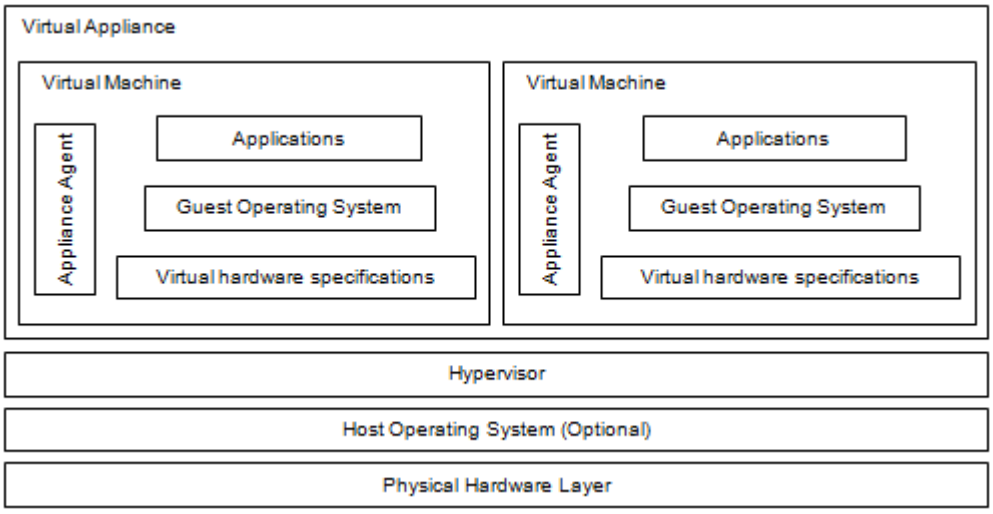

Figure 2: Multi-Node Virtual Appliance with Embedded Agents

The appliance agents are used for the configuration inside the virtual appliance, and are designed to configure the virtual appliance during the first run in a new environment. The agent reconfigures the guest operating system and applications, which are installed during the creation phase according to the requirements of the customer environment. The agent also configures the deployment order of software components and resolves the dependencies of parameters among the components.

The following is a list of configurations that could exist in any virtual appliances.

1. Configurations that are pre-configured in the creation phase of the virtual appliances, e.g. database optimization configurations.
2. Configurations that need to be configured by the appliance agent according to the activation time parameters provided by customers, e.g. hostname, IP address and user credentials.

3. Configurations that are related to other virtual machine's configurations, e.g. encryption keys generated during deployment time.

All of these configurations are completed during the first run by the virtual appliance agents. In a multi-node virtual appliance, various appliance agents would work together to ensure the dependency configuration is correct and consistent.

2.3.2 Virtual Machine Template

Virtual image templates are the core of virtual appliances, as they are the golden images of the virtual machines in the solution. A virtual image template is a ready-to-be-deployed virtual machine image which usually includes an installed guest operating system and a set of pre-configured applications. Templates help enforce consistency and standards. Company standard software, e.g. anti-virus, together with standard settings, e.g. password policy, can be included in the templates such that all machines instantiated with the template will be compliant with company policies.

With small amount of customization, a template can be quickly activated to create new running virtual machines, and the error prone steps that are involved in machine provisioning can be significantly reduced.

The process of deploying virtual appliances using virtual image templates includes the following steps:

1. Install the software solutions on virtual machines.
2. Decide configurations to be exposed to enable the solution to be customized and reused in different environments.
3. Capture a snapshot of each virtual machine using virtual image templates.
4. Deploy the virtual image templates.
5. Activate the virtual machines in the new host environment, and customize the solution through exposed configurations.

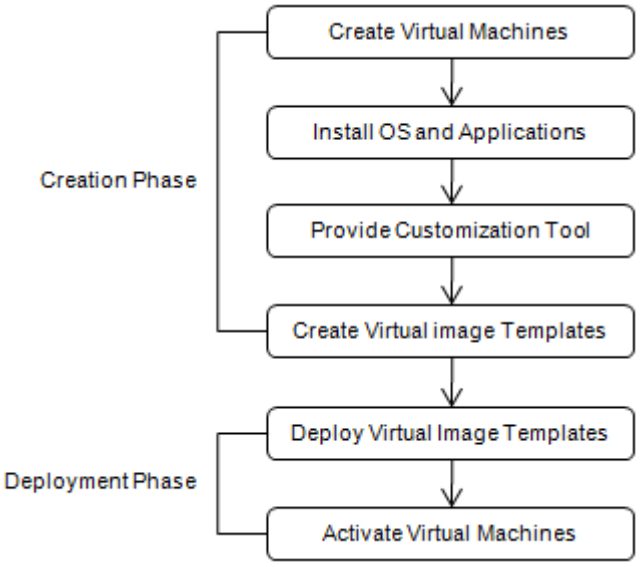

Figure 3: Virtual Appliance Deployment with Virtual Image Templates

The first decision when creating a virtual appliance is to determine what software needs to be installed on the virtual machines. Since the virtual appliance is specific to a particular solution, a minimal set of applications should be installed.

After required applications are installed, and the solution is configured, the next step is to add a tool to help customize the solution in other environments. When a template image is instantiated in a new environment, some modification is required, which may include network changes and customization of the operating system and applications, e.g. credential changes. Tools which help customize a solution on a new environment can be developed and added to the virtual machines before snapshots are taken.

Next, a golden image of the solution needs to be captured by creating virtual image templates. The template creation process is specific to the target hypervisor. Once the template image is built, it can be copied onto new host platforms.

Before activating new virtual machines, a system administrator can make changes to the default resource allocation like CPU and memory according to the requirements in the new environment. Besides resource allocation, network settings, e.g. hostname and IP address, also need to be updated. Customizing virtual machines' network identity is a challenging subject, and the steps involved depend on the hypervisor and the operating system. Despite the difference between hypervisors, network configuration files of the virtual image often need to be modified before a virtual machine can be booted up. Once the machine is started, operating system network settings also need to be changed.

After the network is properly configured, users need to invoke a customization tool, which is part of the virtual image, to properly configure the virtual machine such that the solution can be run properly in the new environment.

2.3.3 Open Virtualization Format (OVF)

In order to apply virtual appliances on a large scale, it is important to define a vendor-neutral standard for packaging and deploying virtual appliances. The Open Virtualization Format (OVF) specification [9] is a hypervisor-neutral, extensible, and open specification for packaging and distribution of virtual appliances. OVF seeks to allow virtual machine hypervisor vendors and users of virtual machine technology to create and consume virtual machine metadata free from proprietary formats. The OVF specification was co-authored by many industry leaders, including IBM, Microsoft, Dell, HP, VMware and XenSource. The specification is backed by the Distributed Management Task Force (DMTF) and almost every major virtualization vendors use OVF as an industry standard format of choice for virtual appliances.

The following are the key properties of the OVF:

- Secure distribution – content verification and integrity checks are supported based on industry standard public key infrastructure; a basic scheme for software licensing management is also provided.
- Optimized for customer experience – packages, virtual machines, as well as meta-data components can be validated during deployment time; appliance

relevant user-readable descriptive information can also packaged and used by virtualization platform to streamline the deployment process.

- Supports both single-node and multi-node configurations – simple single virtual machine solution as well as multi-tier services consist of multiple interdependent virtual machines are both supported.
- Portable packaging – although OVF is virtualization platform neutral, platform-specific enhancements can also be captured.
- Vendor and platform independent – OVF does not rely on the use of any specific host platform, virtualization platform, or guest operating system.
- Extensible – OVF is designed to be extensible such that it can deal with future technologies that may arise; encoding of custom meta-data is also supported.
- Localizable – user visible descriptions are supported in multiple locales; localization of the interactive processes during the deploying of an appliance is also supported.

Users normally view OVF as a packaging format for virtual appliances. Once installed, an OVF adds to the user's infrastructure as a self-contained, self-consistent software solution. However, from technical point of view, an OVF is in fact a transport mechanism for virtual machine templates.

OVF is not a specification that describes a virtual disk. Importing OVF content requires hypervisor compatibility with the associated virtual disk. VMWare's VMDK Virtual Disk Format, Microsoft's and XenServer's VHD Virtual Hard Disk format, and

the open source QCOW format, are all run-time virtual machine image formats and are different from the OVF in the following ways:

- They are virtualization platform dependent and cannot be run on multiple virtualization platforms.
- They operate in the scope of a single virtual machine disk, do not support virtual machine with multiple disks, and do not support solutions with multiple virtual machines.
- They do not provide customization of the virtual machine at deployment time.

An OVF package is a format for distributing software solutions to be deployed in virtual machines. It consists of an OVF descriptor (.ovf file), one or more virtual disks (e.g. .vmdk or .vhd files), a manifest file (.mf file) and an optional certificate file (.cert file). An OVF package maybe stored as a single open virtual appliance (.ova) file using the TapeARchive (TAR) format.

An OVF descriptor is an XML document that describes the metadata of virtual machines included in the package, as well as the virtual disk locations; the virtual disks can be part of the package itself, or can be referred externally via HTTP. The descriptor also contains information on how to manage the virtual machines during deployment, e.g. virtual resource requirements, end-user licensing agreements (EULA) and customization parameters. An OVF descriptor may or may not have a certificate - it is needed only if the user chooses to digitally sign the OVF package. If

no certificate file is present, the manifest file, which is used to describe the package and checksum, is also optional.

An OVF environment is a section in the OVF descriptor which defines information of the guest software. It provides a standard and extensible way for virtualization platforms to communicate deployment configuration to virtual appliances. An OVF environment contains deployment time customization information such as operating system level configuration like hostname and IP address, as well as application level configurations like DNS name and administration port of the database server. The set of properties to be configured is typically entered by the user using a wizard-style interface during deployment. However, OVF environments also allow guest software to automate the configuration between multi-node solutions. For example, an application server may automatically configure itself with the hostname of a directory server without any manual user interaction.

A sample OVF descriptor for a typical single virtual machine appliance can be found in Appendix A. The descriptor first lists a set of files that are part of the OVF package. It then describes a set of virtual disks and networks that will be used by the appliances. Lastly, it describes the content of the virtual appliance. The content consists of 5 sections:

1. ProductSection – Product information, such as name and vendor, of the appliance, and a set of properties that can be used to customize the appliance.

The customization properties will be configured at deployment of the appliance.

2. AnnotationSection – Free form annotation, a ways to extend the OVF specification.
3. EulaSection – Licensing terms for the appliance, which are typically shown during deployment time.
4. HardwareSection – Virtual hardware and devices requirements for the virtual machines.
5. OperatingSystemSection – Guest operating system information.

Figure 4 illustrates the steps that are involved in the creation of an OVF package. OVFs are built by packaging a set of pre-configured virtual machines onto a set of virtual disks, encoding virtual disks appropriately, attaching an OVF descriptor, and optionally digitally signing the package. The virtual disks in the OVF package are virtualization platform specific, and might not be understood by the hypervisor attempting the installation. Note that OVF does not support conversion of virtual disks between virtualization platforms, and does not support conversion of guest software between processor architectures and hardware platforms. As a result, tools provided by hypervisor vendors, e.g. VMware's OVF tool [14] and Citrix's XenConvert [15], need to be used to create virtualization platform neutral OVF packages.

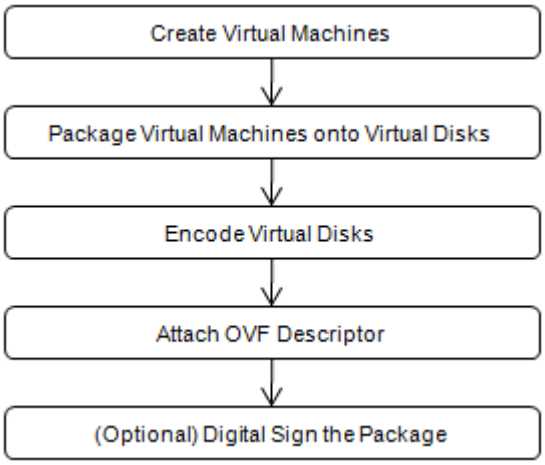

Figure 4: Open Virtualization Format Package Creation Process

Deployment of an OVF package transforms the virtual machines into the runtime format understood by the target virtualization platform. Figure 5 illustrates the steps that are involved in an OVF deployment process.

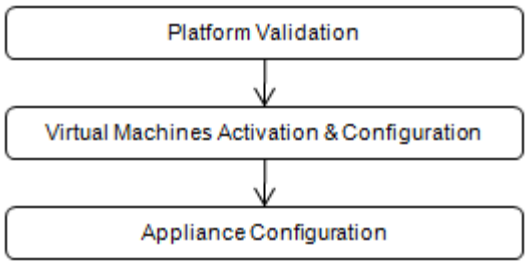

Figure 5: Open Virtualization Format Package Deployment Process

During the deployment, a virtualization platform validates the OVF integrity to ensure the OVF package has not been modified in transit, and checks the compatibility of the local virtual hardware against the information specified in the OVF descriptor. The platform then displays product information, asks for license acceptance, and prompts user for customization information such that it can

configure the virtual machines. Once virtual machine configuration is complete, it is expected that virtual machines can be successfully started with valid resource allocation and network configuration. When the appliance is booted for the first time, additional configuration of the software solution can be done through a management interface provided by the appliance. There is no standard defined way on how appliance configuration should be done - different appliance provides different management interface and may perform solution configuration in a different way.

2.3.4 Just Enough Operating System (JeOS)

General purpose operating systems, e.g. Windows Server, Red Hat Enterprise Linux and AIX, are designed to manage various hardware resources and to provide interfaces and libraries to support a wide range of applications. Overtime, due to the ever growing number of devices and applications, general purpose operating systems has grown significantly in size and complexity. However, most applications require only a fraction of the functionality provided by the overgrown operating system environments. The extra, unused packages become a liability from the security, performance and management perspectives.

Just enough operating system (JeOS) [12] is a slimmed down version of a general purpose operating system that is designed to fit the needs of a particular software solution. Only the operating system interfaces, functions, libraries, resources and third-party components that required by the solution will be provided. By ripping

out unused interfaces and libraries, as well as turning off unnecessary services, the operating system becomes smaller, more secure, easier to manage and better performing. Hardware utilization is also increased, as IT organizations can run more instances per server with smaller footprint operating systems.

Ubuntu and other Linux vendors have provided JeOS solutions that are optimized for running virtual appliances, and include only the base elements that are needed to support the application load. Table 1 provides a brief look at some of the major operating system vendors that offer appliance-optimized operating systems.

OS Vendor	JeOS Solution
Ubuntu	Ubuntu JeOS
Novell	SUSE Linux Enterprise JeOS
Red Hat	Red Hat Appliance Operating System
LiveTime	LiveTime JeOS
Oracle	Oracle Enterprise Linux JeOS
OpenSolaris	OpenSolaris JeOS
openSUSE	Linux Minimal Edition JeOS – LimeJeOS

Table 1: List of Just Enough Operating System Solutions

2.3.5 Virtualization Integrator (VSI)

Most of today's software solutions, especially those for enterprise use, consist of a large number of components and often calls on the functions of other components. Typically, each software component is deployed on a different machine to provide

good isolation. Therefore, one solution will likely span the capabilities of multiple virtual machines, and these machines need to be integrated with each other for coordinated service deployment and management.

To assist in the deployment and management of services that span multiple virtual machines, [11] proposed an appliance management tool called virtualization integrator (VSI). VSI extends the definition of virtual appliance by embedding appliance intelligence inside virtual machines and by providing integrated features inside the machines. VSI is an extendable framework, which enables appliance builders to achieve the following goals:

1. Rapid provisioning – VSI simplifies and accelerates the deployment and activation of virtual appliances by checking parameter dependencies and coordinating execution sequences.
2. Selective presentation –VSI provides flexible interfaces to present meaningful, customized information.
3. Simplified operation –VSI enables different operation granularities by coordinating the operation sequences within virtual appliances.
4. Automated consolidation –VSI enables automated service performance management by automatically monitoring and optimizing resource allocations.

In large scale datacenters, software services usually consist of a set of software resources located in multiple virtual machines. The software resources and the

virtual machines need to be integrated to provide higher-level manageability for the data center administrator and external management systems. VSI is an inner appliance management framework which connects resources distributed across multiple machines to provide autonomic management within the virtual appliance, and provides a simplified view to external management services. Figure 6 shows the overall architecture of VSI.

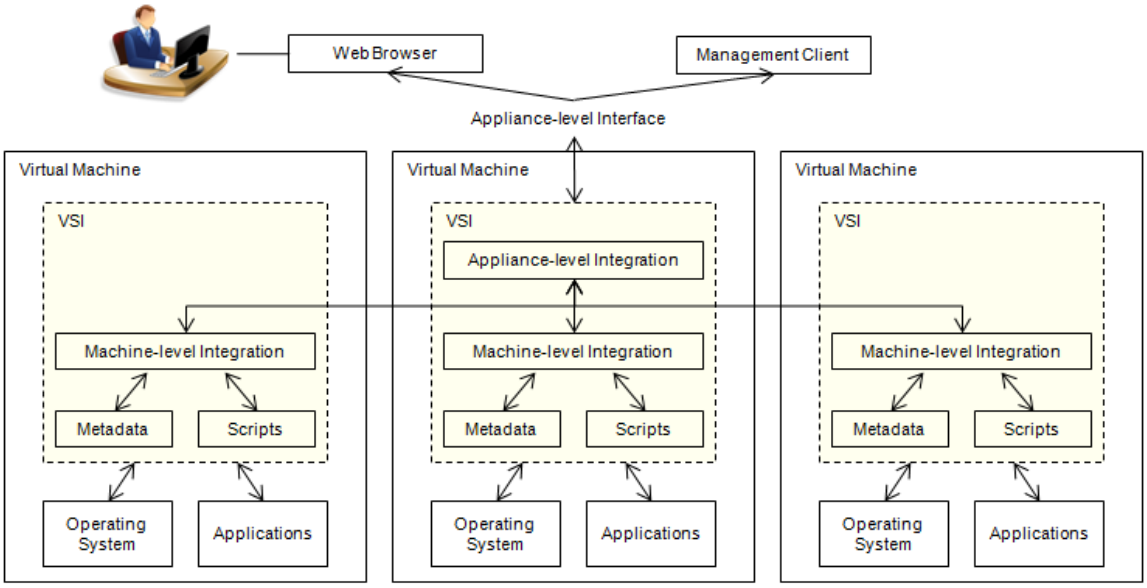

Figure 6: Overall Architecture of Virtualization Integrator [11, Figure 5]

VSI is embedded in each virtual machine as management software. The VSIs in a virtual appliance are interconnected so that VSIs can be coordinated to achieve the required management capabilities. Among these VSIs, one of them is designated as the primary coordinator which all VSIs exchange information and commands. Management capabilities of VSIs are integrated at two levels: at the machine level and at the appliance level. The machine-level capability integrates the software resources located in a virtual machine and exposes the machine-level function using

standard application programming interfaces (APIs). The appliance-level integrator, on the other hand, connects multiple machines within the appliance to compose higher-level management capabilities. Each management capability is implemented as an appliance-level agent embedded into the primary coordinator and as machine-level agents embedded into every virtual machine within the solution. Users can extend each VSI's management capability by providing software specific scripts and metadata. Scripts are used to support software configuration, control, and data-fetching operations in each machine. Metadata is used to describe the virtual image, software scripts, and functional scripts of the virtual machine. Metadata is also used to define the operation sequences and parameter dependencies in a single virtual machine and across multiple virtual machines. VSI metadata is an XML document based on the Open Virtualization Format (OVF) standard plus VSI extensions. VSI exposes its management capabilities and communicate amongst one another through representational state transfer (REST) APIs, through which external management components and users with Web browsers can access the appliance.

To build a virtual appliance with embedded VSI capability, VSI needs to be installed on each virtual machine before images are being captured, such that its components, including scripts and metadata, are embedded into the machines. The metadata includes the necessary information for the machine to be successfully configured and activated, such as network customization parameters; the script provides the capability to configure and manage the software inside the machine so the resources required during appliance deployment are provided.

During solution assembly, multiple virtual machines are assembled into a virtual appliance to provide a high-level service, and linkages are built across the virtual machines. VSI provides the function for creating cross-machine constraints, checking metadata compliance, and merging and synchronizing metadata.

When data center administrators are ready to deploy virtual appliances with embedded VSIs, they can take advantage of the VSI topology construction function to describe the inter- and intra-machine structures of the appliance. VSI enables constructing the topology of a virtual appliance at runtime, and the constructed topology provides one integrated appliance-level interface to the external systems, rather than individual machine-level interfaces.

In the service activation phase, services and machines are instantiated. VSI constructs the software solution in a coordinated manner from the pre-configured template state to the customized state with users' unique parameter values. For appliances that span multiple machines, their instantiation operation often requires domain experts to coordinate the activation sequence of all software resources involved. However, since VSI-embedded virtual machines already have the intelligence implanted by domain experts in the creation and assembly phase, data center administrators do not need to develop a thorough knowledge of the prerequisite software and their interdependencies in the deployment phase.

After the appliance is deployed and activated, users can take advantage of the management modules that are provided by VSI. The service measurement instrumentation component of VSI provides useful data that allows the virtual solution management node to monitor the health of the service and all registered machines. The management node performs its management and monitoring tasks by checking each virtual machine on their CPU, memory, network, disk storage, and energy consumption, and then provide selective summary for the service.

The VSI security compliance module is designed to track and guarantee the password consistency among software and systems. As an example, when the password of the database server is changed, the passwords stored in client applications also need to be changed in order to maintain the database connectivity. If password compliance rule is defined in metadata, VSI can automatically update the client applications when the password of the database server is changed.

The performance optimization module provides a framework to achieve automated performance optimization by utilizing the monitoring, tuning, and optimizing functions that are defined and embedded into the VSI framework in the appliance creation and solution assembly phases. The monitoring function provides the capability to obtain runtime performance metrics such as throughput and response time. The performance optimization function executes predefined optimization policies and operations automatically in runtime.

Other than the management modules that are provided by VSI out of the box, users can also extend VSI by adding additional management functions, such as problem determination module.

2.4 Industry Adoption

Virtual appliance is popularly used in the industry. Several virtualization platform vendors, e.g. VMWare and Citrix, have developed tools to create virtual appliances from virtual machines. Many applications are now delivered as virtual appliances, and are available for download from virtual appliance libraries, e.g. VMware, Turnkey Linux and Openbox.

Virtual appliances are also used in academia. For instance, Columbia University uses virtual appliance to help facilitate the teaching of operating systems which provides hands-on kernel-level project experience without the need for computer laboratory facilities [6]. In this case, a virtual appliance is created for homework assignments which can be run on students' personal computers in virtual machines without interfering with any existing software already on the students' computers.

3. IBM Tivoli Provisioning Manager Deployment

In this section, high-level information of IBM Tivoli Provisioning Manager [17] and its current deployment process will be discussed.

3.1 System Lifecycle Management

As IT infrastructures grow in size, cost and complexity, a comprehensive lifecycle management solution becomes business-critical. Failure to take a holistic, automated view of lifecycle management will inevitably and negatively affect total cost of ownership, regulatory compliance and operational efficiency.

Organizations need an integrated, automated and scalable solution to manage the complete IT infrastructure lifecycle end to end, including server and client platforms, network environments and data stores. With a lifecycle management solution, IT organizations can leverage more resources with fewer people, allowing key skilled personnel to focus on activities that grow the business rather than just maintaining it.

Figure 7 illustrates how a provisioning application, like IBM Tivoli Provisioning Manager (TPM), can help in managing system life cycle.

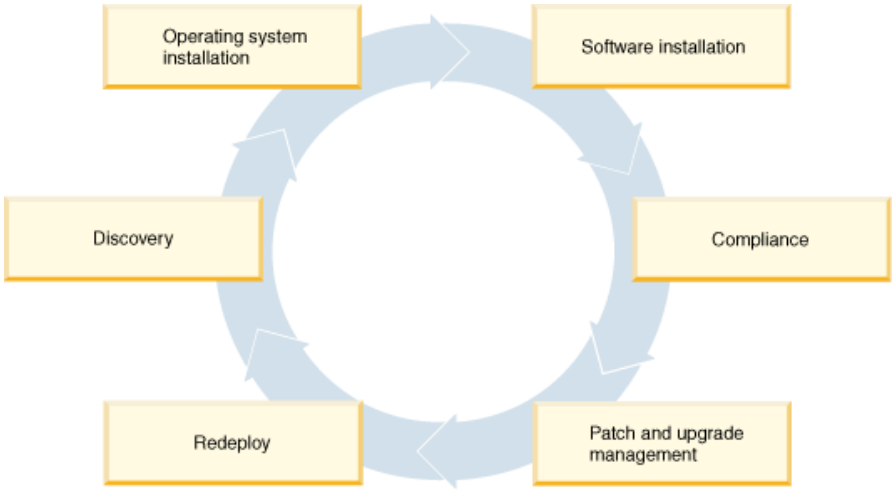

Figure 7: System Life Cycle [17]

Enterprise starts managing their datacenter by creating a list of resources that are available. Information like which operating systems and software are on each resource are also captured. The automated process of finding resources within an enterprise is called discovery. The discovered inventory is then recorded in a centralized repository on the provisioning server that represents all the physical and logical assets that need to be managed. The centralized repository is called data model.

After resources are discovered, there might be several resources that require a new operating system. The need for new operating system deployment could be triggered by adding new systems to the datacenter, creating new virtual machines in hypervisor, and redeploying existing systems for other purpose. With provisioning applications, deployment specialist can easily install and configure the operating system on each computer. These computers now need software.

All software defined in the data model is recorded in a software catalog. Software packages are stored in file repositories linked to the software catalog. From the software catalogs, deployment specialists can install the required software on each computer.

After installing operating system and software, datacenter administrators need to ensure all computers in the enterprise are compliant with the corporate policies, e.g. each computer needs to have an active antivirus service running with up-to-date virus definitions. Using compliance checks provided by a provisioning application, administrators can determine which computers are non-compliant, and then remediate the non-compliant system by installing appropriate software, and configuring appropriate processes.

When operating system vendors release patches for their operating system, the provisioning application can retrieve the available patches from a third-party repository, and then query the computers in the enterprise to see which computers need the patches. Datacenter administrators can review the result, approve the patches to be installed, as well as distribute and install the patches.

Finally, when some systems have completed their assigned mission, they can be reused for other purpose. Deployment specialists then need to replace the

operating systems with a newer version, or install the software stack with different applications. The entire cycle starts over again.

3.2 IBM Tivoli Provisioning Manager (TPM)

IBM Tivoli Provisioning Manager (TPM) is an IBM Tivoli product which provides system lifecycle management solutions. TPM helps organizations with provisioning, configuration and maintenance of servers and virtual servers, operating systems, middleware, applications, storage and network devices.

Managing servers from cradle to grave is TPM's main focus. As illustrated in figure 7, server lifecycle starts when a server is bought (or created via Virtual Machine hypervisors), put to use by installing appropriate operating system and software, managed on an ongoing basis for compliance and software maintenance, reassigned when projects finish, and ultimately retired or made surplus. Though traditionally, asset management has been used to track servers from "cradle to grave", a tool such as TPM is needed to deploy and maintain the operating systems, patches, and software. Customers, especially enterprises, not only need to track their machines through asset management and but also need to track the machines' provisioning processes as they are built and put into production.

TPM provides capabilities of automating the processes of system life cycle management. It supports customers to develop an optimized IT environment through modeling of datacenter resources and modeling of automation flows.

3.2.1 Architecture

Started from TPM version 7.1, TPM became part of IBM service management (ISM) strategy. The technical approach behind ISM strategy is to use a common base for all service management products (SMP). The platform which all ISM products are built on is called Tivoli Process Automation Engine (TPAE). The high level architecture of TPM 7.1 on TPAE is illustrated in figure 8.

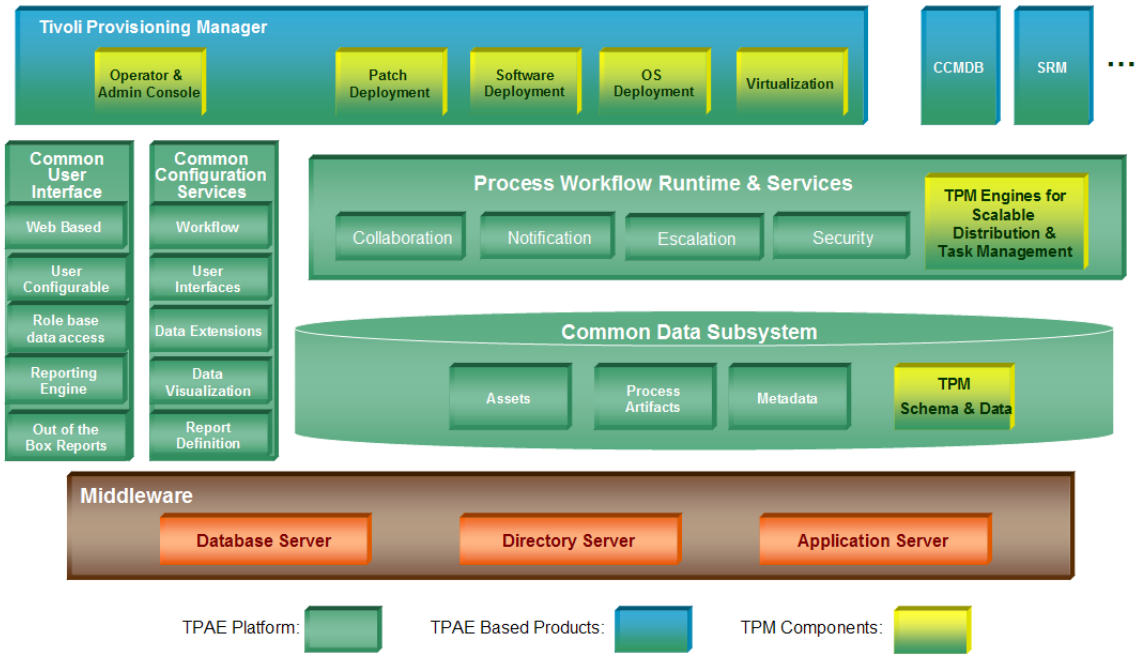

Figure 8: IBM Tivoli Provisioning Manager High-level Architecture

TPAE comes with a common user interface framework, a common process workflow runtime and a common data subsystem. TPM extends each of the TPAE common components to provide solutions for system lifecycle management. TPM leverages TPAE in multiple ways:

- TPM and other ISM offerings are built on top of the same user interface framework and therefore provide a common look and feel.
- TPM provides data integration with other SMP by using and extending the common the database.
- SMP users can use the process workflows and services of the platform to drive TPM to carry out low-level tasks, e.g. TPM provisioning workflows.

Aside from the TPAE components, TPM itself consists of a provisioning server, a Web-based administration console, and an Automation Package Developer Environment. Figure 9 illustrates the main components within TPM, and how the components interact with a managed IT infrastructure and other applications in a datacenter.

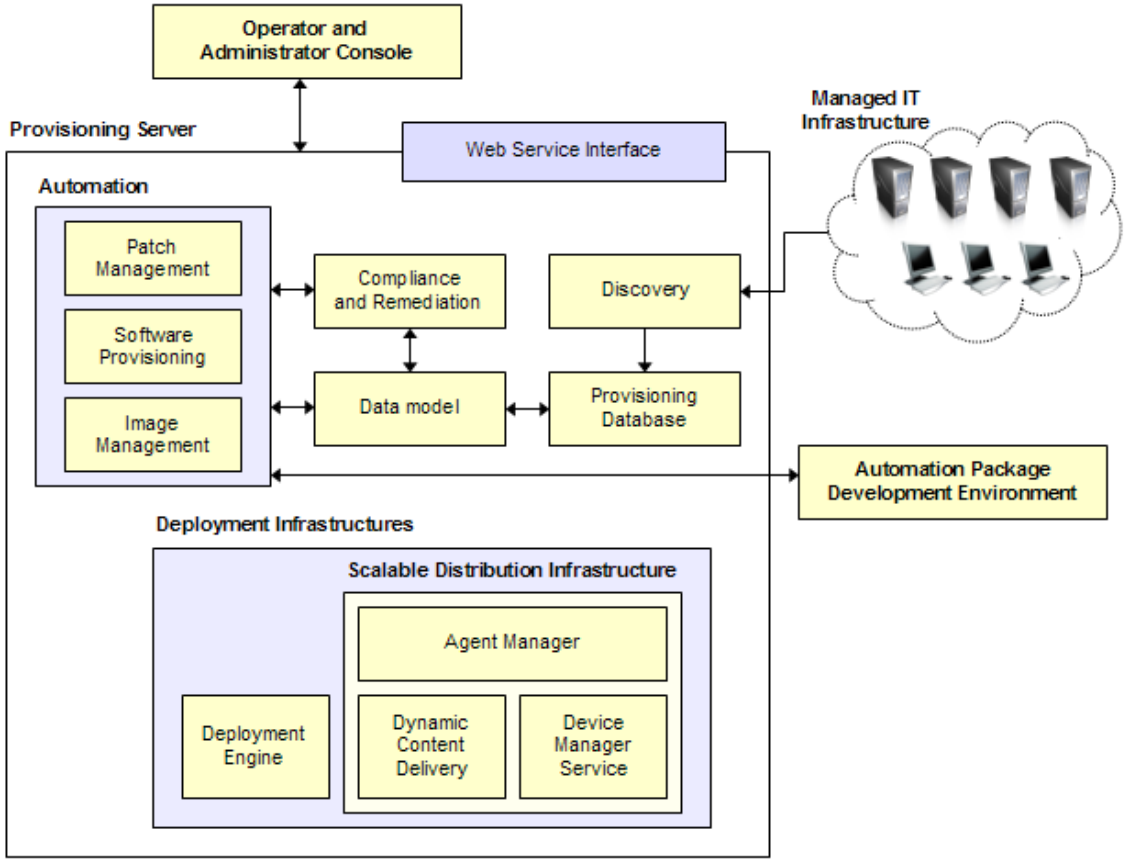

Figure 9: IBM Tivoli Provisioning Manager Component Architecture

The operator and administrator console of TPM is an extension of the TPAE common interface, which was hosted on WebSphere Application Server (WAS). The runtime environment for housing the deployment engine and other engines is Eclipse based Lightweight Infrastructure (LWI) stack. The reporting component is BIRT which uses the reporting runtime and template bundled with TPAE. The provisioning database is physically merged with TAPE database. Detail explanation of the TPM components is out of scope of this paper.

3.2.2 Deployment Mechanism/Process

Similar to many software deployment software, TPM deployment needs to resolve the environment dependencies and the dependencies among the components. Figure10 shows the set of operations involved in TPM deployment.

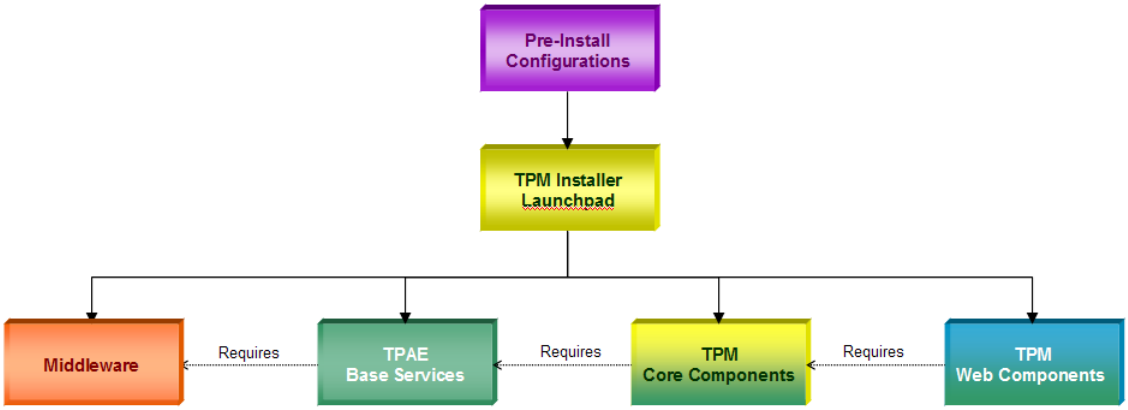

Figure 10: IBM Tivoli Provisioning Manager Deployment Process

Before deploying TPM, users need to perform a series of pre-installation validation and configuration, to ensure the environment meets basic requirements for a new TPM installation. The complete list of pre-installation tasks can be found in Appendix B.

When the environment is ready for deployment, users then need to start the TPM installer launchpad, which allows users to install all software and components required by TPM.

Several middleware products, including database server, directory server, and application server, must be deployed before installing TPM. If the required

middleware is not yet deployed, the middleware installer can be used for installing and configuring them. If the middleware installer is not used, users need to manually configure the middleware resources before proceeding to the next step.

Next in the TPM installation process is to lay down TPAE common components, which are also called base services. During base services installation, deployment software, called Process Solution Installer (PSI), first need to be installed. Since the deployment software cannot be installed on all platforms that are supported by TPM, a separate administrative workstation might be needed. The deployment software is used to deploy the bases services to the application server where TPM is hosted.

After the TPAE framework and the common process management products are deployed on the WebSphere Application Server by the base services installer (BSI), users can install the TPM core components. During this step, all components that are part of the provisioning server will be installed. After core component installation, the data model, the provisioning database, the deployment infrastructure and the automation packages are ready to be used. The only thing that is missing at this point is the web-based operator and administrator console, as well as the automation package development environment (APDE).

The last step of TPM installation is to deploy the web-based operator and administrator console. Since TPM web-based console is an extension of the TPAE common user interface, users must use the machine that has PSI installed from the

previous step to lay down the TPM web components. When web components installation is done, TPM provisioning server is ready to be used, and users can use it to automate system lifecycle management.

For APDE, it is an optional component and its deployment is not part of the TPM installation process. APDE usually get installed on developer's workstation, instead of the provisioning server, after TPM is up and running. Users do not need to install APDE if they are not going to develop their own provisioning workflows.

3.2.3 Deployment Platforms and Topologies

Different customers have different preferences on operating systems, middleware products and deployment topologies. Some customers might choose to reuse the middleware that is already deployed and in-use in their infrastructure. In order to address the various needs of potential customers, TPM supports deployment on a wide range of platforms and topologies.

Table 2 lists all the operating systems and middleware products that are supported by TPM version 7.2.

Operating System	Microsoft Windows AIX Red Hat Enterprise Linux SUSE Linux Enterprise Server

	Solaris
Database Server	DB2 Oracle
Directory Server	IBM Tivoli Directory Server Microsoft Active Directory Server
Application Server	IBM WebSphere Application Server

Table 2: Pre-requisite Software Supported by IBM Tivoli Provisioning Manager

Besides large variety of operating system ands middleware products, TPM also support different deployment options to fulfill different customers' business needs. There are two primary deployment strategies: Single-node and Multi-node.

The single-node topology consists of loading all runtime components, including all middleware, the administrative workstation, and process managers, onto one server. This is typically used for evaluation purposes, as a demonstration, or as a learning environment. For managing enterprise assets and processes, multi-node topology is typically used. Figure 11 shows a deployment with all TPM runtime components on one server with a separate administrative workstation. The administrative workstation must be on a separate computer if the provisioning server is installed on a platform that is not supported by the deployment software for the base services and Web components. The administrative workstation is used to install or update the product, but it is not required during the operation of the product.

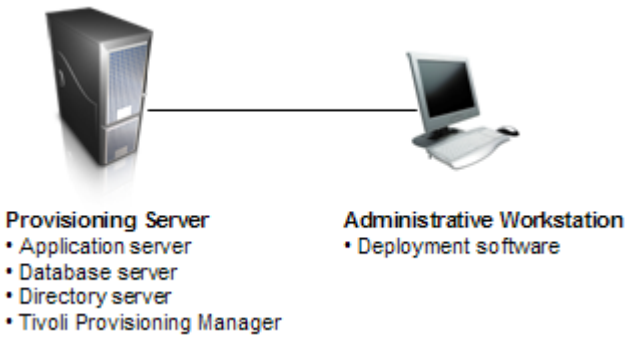

Figure 11: IBM Tivoli Provisioning Manager Single-Node Deployment Topology

The multi-node topology consists of splitting components across several different servers. Components are installed on separate systems to promote load balancing, redundancy, reuse, security, and availability. This type of deployment is typical for production use within an enterprise. Figure 12 shows a sample multi-server deployment topology. Components can also be grouped logically and installed on the same system. In a disparate environment, the collection of servers can be a mixture of Windows® and UNIX® servers.

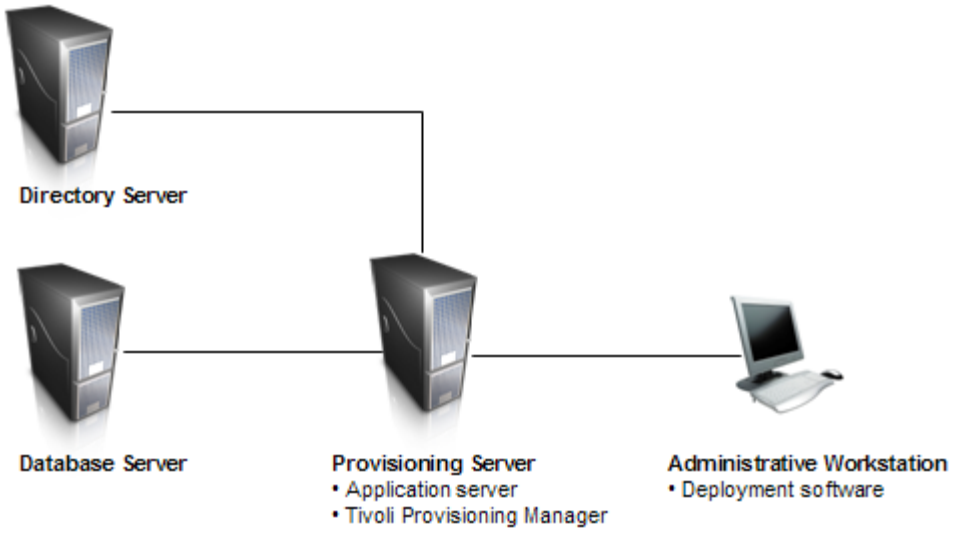

Figure 12: IBM Tivoli Provisioning Manager Multi-Node Deployment Topology

Although TPM supports a wide range of operating system and middleware, its installer cannot handle the installation of all components on all supported platform, e.g. the middleware installer cannot be used on SLES 11 and Solaris, and cannot handle Oracle and MSAD; the deployment service does not support RHEL 4, AIX 5.3, SLES 11 and Solaris. These limitations increase the complexity and difficulties of some of the installation scenarios. Figure 13 provides a summary of the installation scenarios complexities.

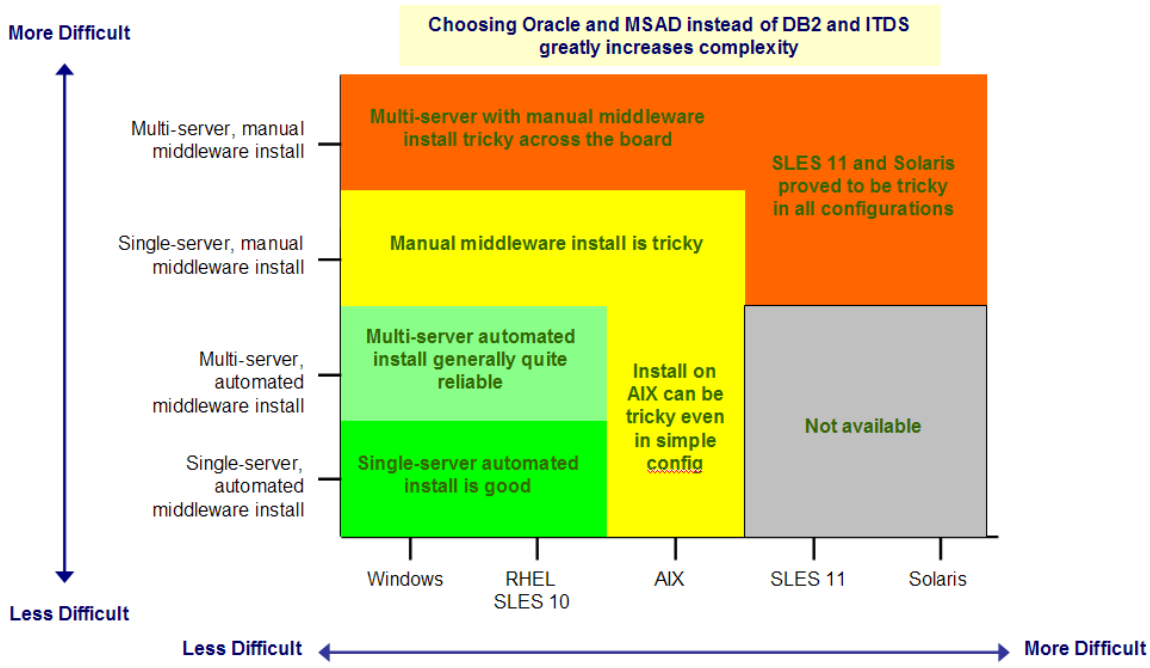

Figure 13: High-level Summary of IBM Tivoli Provisioning Manager Installation Scenarios

3.2.4 Deployment Weakness and Challenges

Complexity is the biggest weakness of the current TPM deployment process. Too many manual steps are involved and too many user inputs are required. Only a few customers can install TPM without getting support from IBM. Following the

installation guide throughout the deployment is a must; otherwise the likelihood of failure is close to 100%.

Table 3 provides some data to quantify the complexity of the TPM deployment.

Number of supported platforms	19
Number of platforms require manual middleware installation and configuration	2
Number of platforms require separate administrative workstation	7
Number of pre-installation tasks	25+
Number of panels in installer	65+
Number of user input fields / selections	320+
Number of installation guide pages	Windows 248; Unix: 294
Customers successfully install TPM without IBM support	< 5%
Time needed to get TPM installed	1 day to 2 weeks or more

Table 3: IBM Tivoli Provisioning Manager Deployment Data

Besides complexity, TPM installer is in fact quite fragile; it is very sensitive to environment settings. Many customers cannot get TPM installed because of the special configurations that customers have on their environment. Disk partitions, settings, volume settings, user settings, and security settings can all affect the stability of the deployment process.

Time taken to deploy TPM is another big issue. IBM sales typically need at least 3 days to prepare the server and install the product. For a one week proof-of-concept activity at customer's site, that only leave the sales two days to customize and demonstrate the product, which should be the main focus of the sales activity.

Problem determination of TPM deployment is also a big challenge. The messages displayed by the TPM installer are not always helpful. Users always need to check the log files for the real error message. However, there are 11 log folders and more than 1000 files generated during the deployment, it is really hard for users to tell what files should be checked. Users always need to seek help from IBM in order to identify the cause of their problem.

After the cause of deployment failure is identified, users need to recover from the failure and retry the deployment. The easiest way to recover from any TPM deployment failure is to restore a previous image of the machine; however, this option is not always available in customers' environment. The failure recovery process for TPM deployment is quite complicated. There is no single uninstallation program for all TPM components. To uninstall TPM, users must remove individual parts of the software in the opposite order from which they were installed. Users also need to perform some manual cleanup steps after the uninstallation program is done, because the uninstallation programs do not remove all files or configuration settings on the computer. Since users need to invoke different uninstallation programs for different components, and there are manual steps involved in

removing files and restoring configurations, it is often the case that users do not clean up properly, and reinstallation fail due to unclean environment.

When users or developers find bugs in the TPM installer, or find configurations that cannot be handled by the installer, TPM cannot deliver the fix for the installer to users until next major release, which is often one and a half year later. Although TPM interim fix and fix packs have a shorter release cycle, those fixes need to be applied on top of a major release, and do not include a full TPM installer, which means, TPM deployment process cannot be improved once the product is shipped.

The TPM development team has tried to address the weaknesses that I just mentioned; however, due to human resource and time to market constraints, they can never do enough to make the TPM deployment process as fast and as reliable as it should be. The following is a list of options that have been considered.

- Reduce number of supported platforms – Code can be simplified and more tests can be done on each of the supported platforms, which can improve the robustness of the installer. However, dropping supported platforms implies giving up a certain customer space; therefore this option is not desirable from marketing perspective.
- Reduce number of supported topologies – Remote server validation and configuration is the most unstable part of the installer. If multi-node topology is removed, the most complicated and most unreliable part of the

installer can be removed. However, in enterprise environment, installing TPM components on separate systems is a must as multi-node topology can promote load balancing, redundancy, reuse, security, and availability. If support of multi-node topology is removed, TPM must provide a post-installation reconfiguration tool which allows users to migrate components to separate machines.

- Reduce number of customizable parameters – Using default values can reduce user input validation and the system behavior uncertainty. However, it is very likely that the default values would violate one or more corporate IT policies, e.g. password must contains at least 2 special characters, applications cannot be installed on system drive, and certain ports are restricted.
- Reduce number of deployment steps – Combining the middleware installer, base services installer, TPM core components installer and TPM web components installer can reduce the deployment complexity and reduce the number of user input, which is the cause of deployment failure in many cases. However, combining the four installers also means eliminating componentization and reducing reusability of the installers. With current design, the middleware installer can be used by all web based applications and the base services installer can be used by all TPAE based products.
- Add more validation and be more environments tolerant – This approach improves robustness of the installer without compensating the deployment options. However, this approach also has some short comings. First, adding

more checks means adding complexity in the code. Second, some validation cannot be done without asking for more user input, e.g. checking permission of remote database backup requires root user credentials of the remote system. Third, installer code change needs to be unit tested by going through the deployment process; depending on the location of the validation added, it might require a complete run of the deployment process, it is not abnormal to spend less than a day on coding, and spend a few days on unit testing. Fourth, it is impossible to handle all configuration possibilities, and it is difficult to identify the common cases, because different customers have different preference in system configurations.

- Reduce number of manual steps involved in undeployment process – This increase the chance of getting the systems back to a stage which is ready for re-deployment. However, this approach is not trivial as components that are used by TPM might also be used by other products, e.g. the database server is used by a few applications, and the directory server is the same as the corporate directory. Sharing of components makes it really hard for TPM to automate the recovery process. It is safer to let the system administrators to decide which part of the systems and what configurations can be rolled back, and thus manual steps are necessary.

The current approach for TPM delivery is costly and complex, the TPM team needs to find a new way to deliver the product in a way which can improve the experience of developers, sales and customers.

4. Deliver TPM as Virtual Appliance

To address the weakness and challenges mentioned in previous section, rapid provisioning with virtual appliances is one of the solutions. Virtual appliance eliminates the complexities in the TPM deployment process. The simplifications are possible because TPM is delivered as a set of preconfigured, prepackaged virtual appliances without application or platform dependencies. To deploy TPM with virtual appliances, data center administrators only need to have a list of machines, a link to the template images, and some parameters for customization. Virtual appliances encapsulate the entire service environment, allowing the TPM development team to resolve execution constraints and dependencies prior to delivery. In other words, TPM components are preinstalled and validated on the virtual machines on which they run. As a result, data center administrators are not confronted by the challenges of configuring and validating their custom environments for TPM deployment.

4.1 Design Goals

The TPM virtual appliance delivery solution proposed in this paper contains the following key characteristics:

- Simple – Users no longer need to worry about the steps that are mentioned in the 248+ pages installation guide. A post-deployment tool will be provided to help users customize the solution with the topology and configurations they prefer.

- Reliable – As soon as the virtual appliance is activated, a working copy of TPM will be available.
- Serviceable – Problem determination becomes easy because point of failure and cause of problem will always be part of the error message displayed.
- Recoverable – Uses can rollback to previous configuration or continue from the point where it failed. The solution will never leave customers' systems in an unusable stage which redeployment is required.
- Extensible – New customization capability can be easily added to the solution without making changes to the framework.
- Updatable – Most updated scripts are available to customers whenever they want to reconfigure TPM.

4.2 Design Overview

To create a TPM virtual appliance, TPM first needs to be installed. Although TPM supports a large number of topology and middleware combinations, single-node topology with DB2 and IBM Tivoli Directory Server is the only combination that will be delivered in the proposed solution. An activation program and a reconfiguration tool will be added to the virtual machine such that users can reconfigure the default setup to multi-node topology using their preferred settings and middleware products. Once the activation program and the reconfiguration tool are added to the virtual machine, an OVF package can be created.

The new TPM deployment process starts with the general OVF package deployment steps, then followed by a set of optional reconfiguration steps. Figure 14 illustrates the flow of the new TPM deployment process.

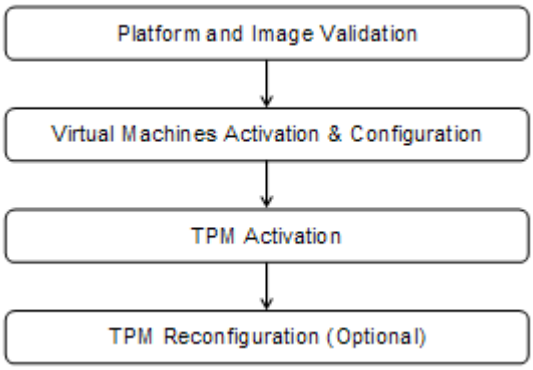

Figure 14: IBM Tivoli Provisioning Manager Virtual Appliance Deployment Process

The first step of the deployment process is to validate the virtual appliance image integrity and the host platform compatibility against the information provided in the OVF descriptor. Once the image integrity and host platform compatibility are checked, the virtual machine needs to be instantiated using the template provided in the OVF package. During virtual machine instantiation, TPM product information is displayed, licensing agreement is accepted, virtual resources are allocated and network settings are changed.

After the virtual machine is booted, the TPM activation program will be launched. The activation program is responsible for updating TPM with new network information and new product administrator password. Once the activation program is done with the basic configuration, TPM will be started and ready to be used.

Besides TPM administrator password, users might want to further customize TPM due to personal preference, corporate policies or other reasons. After TPM is activated, users can use the reconfiguration tool at any time to perform further customization. The reconfiguration tool embedded in the appliance provides different reconfiguration options. The following is a sample list of reconfiguration options.

- Change database server user credentials and instance name
- Change directory server binding credentials and distinguish names
- Configure TPM to use remote database server as data store
- Configure TPM to use corporate directory server for authentication
- Relocate TPM to another application server

4.3 Design Detail

This section provides design details on the activation program and the reconfiguration tool that are embedded in the TPM virtual appliance.

The activation program is automatically launched once the virtual machine is booted up. The program first tries to discover the network settings of the machine by running commands like “hostname”, “nslookup” and “ping”. If the activation program fails to discover the network settings, it will request the user to fix the machine’s network configuration before activating TPM. Without networking

connection, TPM can still be started, but it will not be able to perform datacenter provisioning and management. If network settings can be discovered, the activation program will prompt the user for the product administrator password and will kick off batch scripts or shell scripts, depending on the operating system, to perform the following tasks:

1. Reconfigure the product with new hostname and IP address.
2. Change the product administrator password.

The activation program is very simple. It contains two panels and one user input. It just does enough to make TPM up and running at customer's environment. If users want to further customize the product, they need to run the reconfiguration tool.

The reconfiguration tool is embedded in the virtual machine image and can be found in the $TIO_HOME/tools/reconfig directory. When the reconfiguration tool is launched, it will first connect to the IBM support site and check for the latest update of the tool. If an update of the tool is available, it will be downloaded and will replace existing files in the $TIO_HOME/tools/reconfig/scripts folder. If Internet connection is not available, the version that comes with the appliance will be used, or users can download and update the tool manually.

Once update is done, the reconfiguration tool will load the reconfiguration options from the $TIO_HOME/tools/reconfig/scripts/config.xml file. Users can select any of

the options presented, and the tool will collect user inputs, perform validation and execute automation scripts accordingly.

The reconfiguration tool is an extensible tool which allows easy addition of new reconfiguration capabilities. Reconfiguration options and the steps associated are defined in XML. Developers can extend the tool without understanding the implementation of the tool, and the tool can be updated at anytime by replacing the configuration XML files and the scripts in the $TIO_HOME/tools/reconfig/scripts folder.

```
<!-- List of reconfiguration options -->
<reconfig>
  <option name="Change Administrator Password"
      file="ChangeAdminPwd.xml" />
</reconfig>
```

Figure 15: Sample Configuration File of the TPM Reconfiguration Tool

Figure 15 shows a sample config.xml file which consists of only one reconfiguration option. The option is for changing the administrator password of TPM, and the file that defines the steps that are involved in the password change process is ChangeAdminPwd.xml.

```
<!-- Entry point of the change password process -->
<tns:check name="Change TPM administrator password">
  <!-- Start by asking current password -->
  <tns:checkRef ref="Old Password" />
</tns:check>

<tns:checkDef name="Old Password" title="Current administrator password">
  <tns:test>
    <!-- Ask for current administrator password -->
    <tns:userInput name="AskPassword">
      <tns:question>
        Enter current password:
      </tns:question>
    </tns:userInput>
    <!-- Save user input -->
    <tns:response result="pass" test="ok"
          scope="persist" name="oldPassword">
      <tns:message>Current tioadmin password read.</tns:message>
    </tns:response>
    <tns:response result="fail" test="cancel">
      <tns:message>User canceled password change request.</tns:message>
    </tns:response>
  </tns:test>
  <!- Validate current password before proceeding -->
  <tns:checkRef ref="Validate Current Password" />
</tns:checkDef>

<tns:checkDef name="Validate Current Password">
  <tns:test>
    <tns:method>
      <!-- Invoke password validation script -->
      <tns:invoke type="execute">
        validatePwd.cmd
      </tns:invoke>
      <tns:directory>.</tns:directory>
    </tns:method>
    <!-- Check Return Code -->
    <tns:returnCode test="equals" result="pass">
      <tns:compare>0</tns:compare>
      <tns:message>tioadmin password verified.</tns:message>
    </tns:returnCode>
    <tns:returnCode test="notEquals" result="fail">
      <tns:compare>0</tns:compare>
      <tns:message>Invalid password for tioadmin</tns:message>
    </tns:returnCode>
  </tns:test>
  <!-- Ask for new password only if current password is verified -->
  <tns:checkRef ref="New Password" />
</tns:checkDef>
```

Figure 16: Sample Snippet of the Reconfiguration Process Definition

Figure 16 shows a snippet of the ChangeAdminPwd.xml, which demonstrates how user input can be gathered and how automation scripts can be run. The reconfiguration tool generates user interface and perform actions based on the

information provided in the XML. The XML schema definition (XSD) of the reconfiguration tool and the complete version of the ChangeAdminPwd.xml can be found in Appendix C.

4.4 Challenges

Although the proposed solution provides a fast and robust delivery of TPM solution to customer, there are a few challenges that need to be resolved before the solution can be rolled out to the market.

The biggest challenge related to the proposed solution is licensing issue. Each TPM virtual appliance comes with an operation system which, in most cases, is developed by other software vendors. Delivering a third-party license in an IBM solution is never easy; a lot of business negotiation and legal work need to be done. The process could take months, or even years to be completed, which delays TPM's time to market.

Some customers might be hesitant to the virtual appliance delivery mechanism due to security considerations. Many companies have a set of software needs to be installed and a set of security policies needs to be applied before any machines can be added to their environment. TPM virtual appliances do not comply with corporate security policies out of the box. IT needs to apply security settings on the TPM machine after it is deployed and connected to the network. Moreover,

customers might not be able to apply some of their security rules due to application limitations, e.g. applications are installed on paths that are not permitted, but changing the application location after installation is not supported.

5. Future Work

To address the biggest challenge, the licensing issue, of the proposed solution, JeOS is definitely one of the options that should be explored. If TPM can run on JeOS, the licensing issue would become less an issue as dealing with open software license is much easier than dealing with commercial product license.

Another area that can be explored in the future is the use of VSI. If VSI is used, TPM can deliver virtual appliances with multiple server topologies out of the box, and customers do not need to worry about the post-deployment middleware relocation steps.

6. Conclusion

In this paper, a survey of current research in the area of virtual appliances was provided. The definition, benefits and architecture of virtual appliances were discussed in the survey. Thereafter, a commercial product called IBM Tivoli Provisioning Manager was introduced, and its current deployment mechanism was analyzed. To address the weakness and challenges of the current IBM Tivoli Provisioning Manager deployment process, a proposal to deliver the product as virtual appliance was provided. The proposed solution provides a fast and robust delivery of TPM solution to customer, as it eliminates many error-prone manual steps that are involved in current deployment process.

Appendix A: Sample OVF Descriptor

The following is a complete OVF descriptor for a typical single virtual machine appliance [10].

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

   <!-- 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 -->
   <DiskSection>
      <Info>Describes the set of virtual disks</Info>
      <Disk ovf:diskId="vmdisk1" ovf:fileRef="file1" ovf:capacity="4294967296"
            ovf:format="http://www.vmware.com/interfaces/specifications/vmdk.ht
                ml#sparse"/>
   </DiskSection>

   <!-- Describes all networks used in the package -->
   <NetworkSection>
      <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>
   </NetworkSection>
   <VirtualSystem ovf:id="vm">
      <Info>Describes a virtual machine</Info>
      <Name>Virtual Appliance One</Name>
      <ProductSection>
         <Info>Describes product information for the appliance</Info>
         <Product>The Great Appliance</Product>
         <Vendor>Some Great Corporation</Vendor>
         <Version>13.00</Version>
         <FullVersion>13.00-b5</FullVersion>
         <ProductUrl>
            http://www.somegreatcorporation.com/greatappliance
         </ProductUrl>
         <VendorUrl>http://www.somegreatcorporation.com/</VendorUrl>
         <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>
      </ProductSection>
      <AnnotationSection 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>
      </AnnotationSection>
      <EulaSection>
         <Info>License information for the appliance</Info>
         <License>Insert your favorite license here</License>
      </EulaSection>
      <VirtualHardwareSection>
         <Info>256MB, 1 CPU, 1 disk, 1 nic</Info>
         <Item> Version 1.0.0 Page 11
            <rasd:Description>Number of virtual CPUs</rasd:Description>
            <rasd:ElementName>1 virtual CPU</rasd:ElementName>
            <rasd:InstanceID>1</rasd:InstanceID>
            <rasd:ResourceType>3</rasd:ResourceType>
            <rasd:VirtualQuantity>1</rasd:VirtualQuantity>
         </Item>
         <Item>
            <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
            <rasd:Description>Memory Size</rasd:Description>
            <rasd:ElementName>256 MB of memory</rasd:ElementName>
            <rasd:InstanceID>2</rasd:InstanceID>
            <rasd:ResourceType>4</rasd:ResourceType>
            <rasd:VirtualQuantity>256</rasd:VirtualQuantity>
         </Item>
         <Item>
            <rasd:AutomaticAllocation>true</rasd:AutomaticAllocation>
            <rasd:Connection>VM Network</rasd:Connection>
            <rasd:ElementName>
               Ethernet adapter on "VM Network"
            </rasd:ElementName>
            <rasd:InstanceID>4000</rasd:InstanceID>
            <rasd:ResourceType>10</rasd:ResourceType>
         </Item>
         <Item>
            <rasd:ElementName>Harddisk 1</rasd:ElementName>
            <rasd:HostResource>ovf:/disk/vmdisk1</rasd:HostResource>
            <rasd:InstanceID>22001</rasd:InstanceID>
            <rasd:ResourceType>17</rasd:ResourceType>
         </Item>
      </VirtualHardwareSection>
      <OperatingSystemSection ovf:id="58" ovf:required="false">
         <Info>Guest Operating System</Info>
         <Description>Windows 2000 Advanced Server</Description>
      </OperatingSystemSection>
   </VirtualSystem>
</Envelope>
```

Appendix B: IBM Tivoli Provisioning Manager Pre-installation Tasks

1. Run prerequisites scanner
2. Read release notes
3. Verify supported operating system and middleware combination
4. Perform operating system pre-installation tasks

   Windows:

   a. Verify 8.3 file format is enabled
   b. Disable DNS client services
   c. Ensure Remote Registry service is enabled
   d. Ensure Windows Management Instrumentation service is started
   e. Check if Terminal Server is installed and verify its settings
   f. Check if Windows Scripting Host is enabled
   g. Verify NetBIOS is enabled
   h. Verify required user rights
   i. Disable automatic updates
   j. Uninstall Global Secure ToolKit

   AIX:

   a. Check if the computer is running in 64-bit kernel mode
   b. Disable AIX automountd daemon
   c. Set default paging space
   d. Set maximum number of processes per user to at least 2048
   e. Enable large file sizes in file system
   f. Set user limits
   g. Set environment variable if Compiz is used
   h. Set sticky bit of /tmp directory
   i. Verify command prompt requirements

   Linux:

   a. Verify kernel version

b. Set SELinux to permissive or disabled
c. Disable automatic cleanup of /tmp directory
d. Set swap size
e. Set kernel parameters for DB2
f. Set user limits
g. Set environment variable if Compiz is used
h. Set sticky bit of /tmp directory
i. Verify command prompt requirements

Solaris:

a. Verify root user login shell
b. Verify CONSOLE, PATH and SUPATH environment variables
c. Update PATH variable with the browser directory
d. Set environment variable if Compiz is used
e. Set sticky bit of /tmp directory
f. Create symbolic link to expect
g. Modify sshd_config
h. Set user limits
i. Set swap size
j. Verify command prompt requirements

5. Ensure all UNIX and Linux required packages are installed
6. Verify database server requirements
    a. Verify table space disk space requirements
    b. Ensure asynchronous I/O is enabled
7. Verify directory server requirements
8. Ensure supported browser is installed
9. Allocate appropriate hardware
    a. Ensure minimum memory requirement is met
    b. Ensure enough disk space is allocated
10. Verify requirements for user name and password
11. Verify requirements for database name
12. Ensure fully qualified domain name is configured

13. Ensure static IP address is configured
14. Ensure host name can be resolved either using a DNS server or a hosts file
15. Configure Network Information Service (NIS) if it is installed
16. Enable remote access protocol on each computer involved in multi-server topologies
17. Verify network media speed settings
18. Check ports availability
19. Stop or reschedule anti-virus software
20. Stop or reschedule other process-intensive software
21. Download and extract installation images

Appendix C: TPM Virtual Appliance Reconfiguration Tool

C.1 Reconfiguration XML Definition Schema – reconfig.xsd

```
<xsd:schema targetNamespace="TPMReconfig.xsd" xmlns:tns="TPMReconfig.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ref="http://ws-
i.org/profiles/basic/1.1/xsd"
    elementFormDefault="qualified">

    <!-- root element for the reconfiguration options file -->
    <xsd:element name="reconfig" type="tns:reconfigType" />

    <xsd:complexType name="reconfigType">
        <xsd:sequence>
            <xsd:element name="option" maxOccurs="unbounded" type="tns:optionType" />
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="optionType">
        <xsd:attribute name="name" use="required" />
        <xsd:attribute name="file" use="required" />
    </xsd:complexType>

    <!-- root element for each reconfiguration option -->
    <xsd:element name="check" type="tns:checkType" />

    <xsd:complexType name="reference">
        <xsd:attribute name="ref" use="required" />
        <xsd:attribute name="default" use="optional" />
    </xsd:complexType>

    <xsd:complexType name="checker">
        <xsd:choice>
            <xsd:element name="check" type="tns:checkType" />
            <xsd:element name="checkRef" type="tns:reference" />
        </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="remediator">
        <xsd:choice>
            <xsd:element name="method" type="tns:methodType" />
            <xsd:element name="methodRef" type="tns:reference" />
        </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="checkType">
        <xsd:sequence>
            <xsd:element name="applies" type="tns:checker" minOccurs="0" />
            <xsd:element name="test" type="tns:testType" minOccurs="0"
                maxOccurs="unbounded" />
            <xsd:element name="remediation" type="tns:remediator"
                minOccurs="0" />
            <!-- import an external check file -->
            <!-- for nested checks -->
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="include" type="tns:includeType" />
                <xsd:element name="check" type="tns:checkType" />
                <!-- check reference (evaluated like a check) -->
                <xsd:element name="checkRef" type="tns:reference" />
                <!-- defs w/o execution for later use/reference -->
                <xsd:element name="checkDef" type="tns:checkType" />
                <xsd:element name="choicesDef" type="tns:choicesType" />
                <xsd:element name="userInputDef" type="tns:userInputType" />
                <xsd:element name="propertyDef" type="tns:propertyType" />
                <xsd:element name="methodDef" type="tns:methodType" />
                <xsd:element name="userQueryDef" type="tns:userQueryType" />
            </xsd:choice>
```

```
    </xsd:sequence>
    <xsd:attribute name="name" use="required" type="xsd:string" />
    <xsd:attribute name="title" use="optional" type="xsd:string" />
    <xsd:attribute name="default" use="optional" type="tns:checkAttrType"
        default="pass" />
</xsd:complexType>

<!--
    import an external file or class defining additional checks the
    external file must comply with this XSD, the internal class must
    implement the interface com.ibm.tpm.prereqs.Check
-->
<xsd:complexType name="includeType">
    <xsd:complexContent>
        <xsd:extension base="tns:expressionType">
            <xsd:attribute name="type" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="file" />  <!-- from a file -->
                        <xsd:enumeration value="java" />  <!-- java implementation -->
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- a name/value pair in the current context -->
<xsd:complexType name="propertyType">
    <xsd:complexContent>
        <xsd:extension base="tns:expressionType">
            <xsd:attribute name="name" use="required" type="xsd:string" />
            <xsd:attribute name="scope" use="optional" type="tns:scopeAttrType" />
            <xsd:attribute name="context" use="optional" type="xsd:string" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="choicesType">
    <xsd:sequence>
        <xsd:element name="choice" maxOccurs="unbounded" type="tns:expressionType" />
    </xsd:sequence>
    <xsd:attribute name="name" use="required" type="xsd:string" />
</xsd:complexType>

<!--
    a credential, substitution is the credential password/key
-->
<xsd:complexType name="credentialType">
    <xsd:attribute name="name" use="required" type="xsd:string" />
    <xsd:attribute name="type" use="optional" default="decrypted">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="decrypted" />
                <xsd:enumeration value="encrypted" />
                <xsd:enumeration value="basicHttp" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>

<!-- string with substitutions from the dictionary -->
<xsd:complexType name="expressionType" mixed="true">
    <xsd:sequence mixed="true">
        <xsd:choice minOccurs="0" maxOccurs="unbounded" mixed="true">
            <xsd:element name="property">
                <xsd:complexType>
                    <xsd:complexContent>
                        <xsd:extension base="tns:reference">
                            <xsd:attribute name="scope" use="optional">
                                <xsd:simpleType>
```

```
                                    <xsd:restriction base="xsd:string">
                                        <xsd:enumeration value="session" />
                                        <xsd:enumeration value="persisted" />
                                    </xsd:restriction>
                                </xsd:simpleType>
                            </xsd:attribute>
                            <xsd:attribute name="context" use="optional"
type="xsd:string" />
                        </xsd:extension>
                    </xsd:complexContent>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="methodRef" type="tns:reference" />
            <xsd:element name="userInputRef" type="tns:reference" />
            <xsd:element name="credential" type="tns:credentialType" />
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="testType">
    <xsd:choice minOccurs="0">
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="method" type="tns:methodType" />
                <xsd:element name="methodRef" type="tns:reference" />
            </xsd:choice>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="output" type="tns:valueResult" />
                <xsd:element name="returnCode" type="tns:returnCodeResult" />
            </xsd:choice>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="location" type="tns:locationType" />
                <xsd:element name="setting" type="tns:settingType" />
            </xsd:choice>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="contents" type="tns:valueResult" />
                <xsd:element name="existence" type="tns:existenceResult" />
            </xsd:choice>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="fileInput" type="tns:fileInputType" />
                <xsd:element name="directoryInput" type="tns:directoryInputType" />
            </xsd:choice>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="contents" type="tns:valueResult" />
                <xsd:element name="existence" type="tns:existenceResult" />
                <xsd:element name="response" type="tns:userResponseResult" />
            </xsd:choice>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="userInput" type="tns:userInputType" />
                <xsd:element name="choicesInput" type="tns:choicesInputType" />
            </xsd:choice>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="value" type="tns:valueResult" />
                <xsd:element name="response" type="tns:userResponseResult" />
            </xsd:choice>
        </xsd:sequence>
        <xsd:sequence>
            <xsd:element name="userQuery" type="tns:userQueryType" />
            <xsd:element name="response" type="tns:userQueryResult"
                maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="check" type="tns:checkType" />
                <xsd:element name="checkRef" type="tns:reference" />
```

```
            </xsd:choice>
            <xsd:element name="result" type="tns:checkResult"
                maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:choice>
</xsd:complexType>

<!--
    note that OS commands (esp on windows are different that programs or
    scripts to execute
-->
<xsd:complexType name="methodType">
    <xsd:sequence>
        <xsd:element name="invoke">
            <xsd:complexType>
                <xsd:complexContent>
                    <xsd:extension base="tns:expressionType">
                        <xsd:attribute name="type" use="required">
                            <xsd:simpleType>
                                <xsd:restriction base="xsd:string">
                                    <!-- execute script or application -->
                                    <xsd:enumeration value="execute" />
                                    <!-- execute system command -->
                                    <xsd:enumeration value="command" />
                                    <!-- execute java class with static main method -->
                                    <xsd:enumeration value="java" />
                                </xsd:restriction>
                            </xsd:simpleType>
                        </xsd:attribute>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="directory" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="environment" type="tns:propertyType"
            minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="name" use="optional" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="locationType">
    <xsd:complexContent>
        <xsd:extension base="tns:expressionType">
            <xsd:attribute name="type" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="file" /> <!-- get file -->
                        <xsd:enumeration value="directory" /> <!-- check directory -->
                        <xsd:enumeration value="url" />  <!-- get URL -->
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="settingType">
    <xsd:complexContent>
        <xsd:extension base="tns:expressionType">
            <xsd:attribute name="type" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <!-- this session context -->
                        <xsd:enumeration value="session" />
                        <!-- persisted across this session context -->
                        <xsd:enumeration value="persisted" />
                        <!-- environment variables -->
                        <xsd:enumeration value="environment" />
                        <!-- java System.properties environment variables -->
                        <xsd:enumeration value="properties" />
```

```
                        <!-- registry value (windows) -->
                        <xsd:enumeration value="registry" />
                        <!-- user -->
                        <xsd:enumeration value="login" />
                        <!-- user permission -->
                        <xsd:enumeration value="permission" />
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="choicesInputType">
    <xsd:sequence>
        <xsd:element name="question" type="tns:expressionType" />
        <xsd:element name="default" type="tns:expressionType"
            minOccurs="0" />
        <xsd:choice>
            <xsd:element name="choices" type="tns:choicesType" />
            <xsd:element name="choicesRef" type="tns:reference" />
        </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="name" use="optional" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="userInputType">
    <xsd:sequence>
        <xsd:element name="question" type="tns:expressionType" />
        <xsd:element name="default" type="tns:expressionType"
            minOccurs="0" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="fileInputType">
    <xsd:sequence>
        <xsd:element name="question" type="tns:expressionType" />
        <xsd:element name="default" type="tns:expressionType"
            minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="extensions" use="optional" type="xsd:list" />
    <xsd:attribute name="name" use="optional" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="directoryInputType">
    <xsd:sequence>
        <xsd:element name="question" type="tns:expressionType" />
        <xsd:element name="default" type="tns:expressionType"
            minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" use="optional" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="userQueryType">
    <xsd:sequence>
        <xsd:element name="question" type="tns:expressionType" />
    </xsd:sequence>
    <xsd:attribute name="name" use="optional" type="xsd:string" />
    <xsd:attribute name="type" use="required" type="tns:buttonsAttrType" />
</xsd:complexType>

<xsd:attributeGroup name="resultAttrGroup">
    <xsd:attribute name="result" use="required" type="tns:resultAttrType" />
    <xsd:attribute name="scope" use="optional" type="tns:scopeAttrType" />
    <xsd:attribute name="name" use="optional" type="xsd:string" />
    <xsd:attribute name="context" use="optional" type="xsd:string" />
</xsd:attributeGroup>

<xsd:complexType name="returnCodeResult">
    <xsd:sequence>
        <xsd:element name="compare" type="tns:expressionType" />
```

```
        <xsd:element name="message" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="propertyDef" type="tns:propertyType"
            minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:resultAttrGroup" />
    <xsd:attribute name="test" use="required">
        <xsd:simpleType>
            <xsd:union
                memberTypes="tns:baseAttrType
                                tns:valueAttrType
                                tns:numericAttrType" />
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="valueResult">
    <xsd:sequence>
        <xsd:element name="compare" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="message" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="propertyDef" type="tns:propertyType"
            minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:resultAttrGroup" />
    <xsd:attribute name="test" use="required">
        <xsd:simpleType>
            <xsd:union memberTypes="tns:baseAttrType
                tns:valueAttrType" />
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="existenceResult">
    <xsd:sequence>
        <xsd:element name="message" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="propertyDef" type="tns:propertyType"
            minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:resultAttrGroup" />
    <xsd:attribute name="test" use="required">
        <xsd:simpleType>
            <xsd:union memberTypes="tns:baseAttrType
                                tns:existsAttrType" />
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="userResponseResult">
    <xsd:sequence>
        <xsd:element name="message" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="propertyDef" type="tns:propertyType"
            minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:resultAttrGroup" />
    <xsd:attribute name="test" use="required">
        <xsd:simpleType>
            <xsd:union memberTypes="tns:baseAttrType tns:okCancelAttrType" />
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="userQueryResult">
    <xsd:sequence>
        <xsd:element name="message" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="propertyDef" type="tns:propertyType"
            minOccurs="0" maxOccurs="unbounded" />
```

```
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:resultAttrGroup" />
    <xsd:attribute name="test" use="required">
        <xsd:simpleType>
            <xsd:union memberTypes="tns:baseAttrType tns:yesNoAttrType" />
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="checkResult">
    <xsd:sequence>
        <xsd:element name="message" type="tns:expressionType"
            minOccurs="0" />
        <xsd:element name="propertyDef" type="tns:propertyType"
            minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attributeGroup ref="tns:resultAttrGroup" />
    <xsd:attribute name="test" use="required">
        <xsd:simpleType>
            <xsd:union
                memberTypes="tns:baseAttrType
                             tns:checkAttrType
                             tns:checkedAttrType" />
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>

<xsd:simpleType name="baseAttrType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="always" /> <!-- always pass -->
        <xsd:enumeration value="never" /> <!-- never pass -->
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="interactiveAttrType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="approve" />   <!-- interactive user -->
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="okCancelAttrType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ok" />
        <xsd:enumeration value="cancel" />
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="yesNoAttrType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="yes" /> <!-- returns true|yes|ok -->
        <xsd:enumeration value="no" /> <!-- returned false|no|cancel -->
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="existsAttrType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="exists" /> <!-- target exists -->
        <xsd:enumeration value="notExists" /> <!-- target does not exist -->
        <xsd:enumeration value="empty" /> <!-- no value -->
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="valueAttrType">
    <xsd:restriction base="xsd:string">
        <!-- stdout/content -->
        <xsd:enumeration value="contains" />
        <xsd:enumeration value="notContains" />
        <xsd:enumeration value="equals" />
        <xsd:enumeration value="notEquals" />
        <xsd:enumeration value="matches" /> <!-- matches regex -->
        <xsd:enumeration value="notMatches" /> <!-- does not match regex -->
```

```
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="numericAttrType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="lessThan" />
            <xsd:enumeration value="greaterThan" />
            <xsd:enumeration value="range" /> <!-- x,y where x < value <= y -->
            <xsd:enumeration value="isIn" /> <!-- a,b,...z where value is in set -->
            <xsd:enumeration value="isNotIn" /> <!-- a,b,...z where value is in set -->
        </xsd:restriction>
    </xsd:simpleType>


    <xsd:simpleType name="checkAttrType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="none" />
            <xsd:enumeration value="pass" />
            <xsd:enumeration value="info" />
            <xsd:enumeration value="warn" />
            <xsd:enumeration value="fail" />
            <xsd:enumeration value="error" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="resultAttrType">
        <xsd:union memberTypes="tns:checkAttrType
                                tns:interactiveAttrType" />
    </xsd:simpleType>

    <xsd:simpleType name="checkedAttrType">
        <xsd:restriction base="xsd:string">
            <!-- any passed value -->
            <xsd:enumeration value="passed" />
            <!-- any failed value -->
            <xsd:enumeration value="failed" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="buttonsAttrType">
        <xsd:restriction base="xsd:string">
            <!-- button types -->
            <xsd:enumeration value="ok|cancel" />
            <xsd:enumeration value="yes|no" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="scopeAttrType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="session" />
            <xsd:enumeration value="persist" />
            <xsd:enumeration value="unpersist" />
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

C.2 Sample reconfiguration configuration – ChangeAdminPwd.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:check name="Change Tivoli Provisioning Manager Administrator Password"
    xmlns:tns="TPMReconfig.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="PrereqCheck.xsd">

    <!-- Entry point of the change password process -->
    <tns:check name="Change TPM administrator (tioadmin) password">
        <!-- Start by asking current password -->
        <tns:checkRef ref="Old Password" />
    </tns:check>

    <tns:checkDef name="Old Password" title="Current administrator password">
        <tns:test>
            <!-- Ask for current administrator password -->
            <tns:userInput name="AskPassword">
                <tns:question>
                    Enter current password:
                </tns:question>
            </tns:userInput>
            <!-- Save user input -->
            <tns:response result="pass" test="ok" scope="persist" name="oldPassword">
                <tns:message>Current tioadmin password read.</tns:message>
            </tns:response>
            <tns:response result="fail" test="cancel">
                <tns:message>User canceled password change request.</tns:message>
            </tns:response>
        </tns:test>
        <tns:checkRef ref="Validate Current Password" />
    </tns:checkDef>

    <tns:checkDef name="Validate Current Password">
        <tns:test>
            <tns:method>
                <!-- Invoke password validation script -->
                <tns:invoke type="execute">
                    validatePwd.cmd
                </tns:invoke>
                <tns:directory>.</tns:directory>
            </tns:method>
            <!-- Check Return Code -->
            <tns:returnCode test="equals" result="pass">
                <tns:compare>0</tns:compare>
                <tns:message>tioadmin password verified.</tns:message>
            </tns:returnCode>
            <tns:returnCode test="notEquals" result="fail">
                <tns:compare>0</tns:compare>
                <tns:message>Invalid password for tioadmin</tns:message>
            </tns:returnCode>
        </tns:test>
        <!-- Proceed only if current password is verified -->
        <tns:checkRef ref="New Password" />
    </tns:checkDef>

    <tns:checkDef name="New Password" title="New administrator password">
        <tns:test>
            <!-- Ask for new administrator password -->
            <tns:userInput name="AskPassword">
                <tns:question>
                    Enter new password:
                </tns:question>
            </tns:userInput>
            <!-- Save user input -->
            <tns:response result="pass" test="ok" scope="persist" name="newPassword">
                <tns:message>New tioadmin password read.</tns:message>
            </tns:response>
            <tns:response result="fail" test="cancel">
                <tns:message>User canceled password change request.</tns:message>
```

```
        </tns:response>
    </tns:test>
    <!-- Ask for password confirmation before proceeding -->
    <tns:checkRef ref="New Password Again" />
</tns:checkDef>

<tns:checkDef name="New Password Again" title="New administrator password">
    <tns:test>
        <!-- Ask for new administrator password again -->
        <tns:userInput name="AskPassword">
            <tns:question>
                Enter new password again:
            </tns:question>
        </tns:userInput>
        <!-- Checks if password entered matches previous entry -->
        <tns:value result="pass" test="equals">
            <tns:compare>
                <tns:property ref="newPassword" />
            </tns:compare>
            <tns:message>New password entries match.</tns:message>
        </tns:value>
        <tns:value result="fail" test="notEquals">
            <tns:compare>
                <tns:property ref="newPassword" />
            </tns:compare>
            <tns:message>New password entries do not match.</tns:message>
        </tns:value>
        <tns:response result="fail" test="cancel">
            <tns:message>User canceled password change request.</tns:message>
        </tns:response>
</tns:test>
<!-- Ask for user confirmation -->
    <tns:checkRef ref="Confirmation" />
</tns:checkDef>

<tns:checkDef name="Confirmation">
    <tns:test>
        <tns:userQuery type="yes|no">
            <tns:question>
                Are you sure you want to change the password for tioadmin?
            </tns:question>
        </tns:userQuery>
        <tns:response result="pass" test="yes">
            <tns:message>User confirmed password change</tns:message>
        </tns:response>
        <tns:response result="fail" test="no">
            <tns:message>User canceled password change</tns:message>
        </tns:response>
    </tns:test>
    <!-- Proceed with password change -->
    <tns:checkRef ref="Change tioadmin Password" />
</tns:checkDef>

<tns:checkDef name="Change tioadmin Password">
    <tns:test>
        <tns:method>
            <!-- Invoke password validation script -->
            <tns:invoke type="execute">
                changePwd.cmd
            </tns:invoke>
            <tns:directory>.</tns:directory>
        </tns:method>
        <!-- Check Reutnr message and return Code -->
        <tns:output result="none" test="contains" scope="persist" name="errorMessage">
            <tns:compare>ERROR</tns:compare>
        </tns:output>
        <tns:returnCode test="equals" result="pass">
            <tns:compare>0</tns:compare>
            <tns:message>tioadmin password changed successfully.</tns:message>
        </tns:returnCode>
        <tns:returnCode test="notEquals" result="fail">
```

```
                <tns:compare>0</tns:compare>
                <tns:message>
                    Failed to change tioadmin password. Error:
                    <tns:property scope="session" ref="errorMessage" />
                </tns:message>
            </tns:returnCode>
        </tns:test>
    </tns:checkDef>
</tns:check>
```

Appendix D: Acronyms

- APDE - Automation Package Development Environment
- API - Application Programming Interface
- BSI - Base Services Installer
- DMTF - Distributed Management Task Force
- EULA - End-User Licensing Agreement
- ISM - IBM Service Management
- JeOS - Just Enough Operating System
- LWI - Lightweight Infrastructure
- NIS - Network Information Service
- OVF – Open Virtualization Format
- PSI - Process Solution Installer
- REST - REepresentational State Transfer
- SMP - Service Management Products
- TAR - TapeARchive
- TPAE - Tivoli Process Automation Engine
- TPM - IBM Tivoli Provisioning Manager
- VSI - Virtualization Integrator
- WAS - WebSphere Application Server
- XSD - XML Schema Definition

References

[1] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual Appliances for Deploying and Maintaining Software", Proceedings of USENIX conference on System administration (LISA), 2003, pp. 181–194.

[2] C. Sapuntzakis and M. S. Lam, "Virtual Appliances in the Collective: A Road to Hassle-Free Computing", Proceedings of Workshop on Hot Topics in Operating Systems, 2003.

[3] C. Sun, L. He, Q. Wang, and R. Willenborg, "Simplifying Service Deployment with Virtual Appliances", Proceedings of the IEEE International Conference on Services Computing (SCC), Hawaii, 2008, pp. 265–272.

[4] VMware, "Virtual Appliances: A New Paradigm for Software Delivery", White Paper, 2008.
http://www.vmware.com/files/pdf/vam/VMware_Virtual_Appliance_Solutions_White_Paper_08Q3.pdf.

[5] VMware, "Virtual Appliance Marketplace",
http://www.vmware.com/appliances/

[6]Laadan, O., Nieh, J., and Viennot, N., "Teaching Operating Systems Using Virtual Appliances and Distributed Version Control", Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE), Milwaukee, March 2010.

[7] R. Willenborg, Q. Wang, D. Gilgen, and S. Smith, "Using Virtual Image Templates to Deploy Websphere Application Server", IBM WebSphere Developer Technical Journal, vol. 5, May 2007.

[8] L. He, S. Smith, R. Willenborg, and Q. Wang, "Automating Deployment and Activation of Virtual Images", IBM WebSphere Developer Technical Journal, vol. 8, Aug. 2007.

[9] Distributed Management Task Force, "Open Virtualization Format Specification," February, 2009.
http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf

[10] Distributed Management Task Force, "Open Virtualization Format White Paper," February, 2009.
http://www.dmtf.org/standards/published_documents/DSP2017_1.0.0.pdf

[11] X. Jin, R. Willenborg, Y. Zhao, C. Sun, L. He, Z. Chen, Y. Chen, Q. Wang, "Reinventing Virtual Appliances", IBM Journal of Research and Development, vol. 53, issue 4, July 2009.

[12] "Just Enough Operating System", June 2010, http://en.wikipedia.org/wiki/Just_enough_operating_system

[13] "SUSE Linux Enterprise Just Enough Operating System", http://www.novell.com/products/jeos/

[14] VMware, "OVF Tool User Guide", 2009, http://www.vmware.com/support/developer/ovf/ovf10/ovftool_10_userguide.pdf

[15] Citrix, "XenConvert 2.1.1", April 2010, http://citrix.com/english/ss/downloads/details.asp?downloadID=1862307

[16] VMware, "Best Practices for Building Virtual Appliances", White Paper, Nov 2007. http://www.vmware.com/files/pdf/Best_Practices_Building_Virtual_Appliances.pdf

[17] IBM, "IBM Tivoli Provisioning Manager Information Center, Version 7.2", July 2010. http://publib.boulder.ibm.com/infocenter/tivihelp/v38r1/index.jsp