ARCHITECTURAL DESIGN OF ENTERPRISE WIDE STANDARD OPERATING ENVIRONMENTS

Andrew Aupek, Manager Library IT

Macquarie University Library
Macquarie University NSW 2109
Phone: +61 2 9850 6558
Fax: + 61 2 9850 7912
Email: andrew.aupek@library.mq.edu.au

Abstract

The concept of a Standard Operating Environment (SOE) is ubiquitous throughout the IT industry when referring to a configuration deployed to desktop PC environments and large scale back-end server farms. However, the specifics of what constitutes and the components that integrate together to form a standard operating environment vary widely amongst IT professionals. To add further confusion, the contents of a standard operating environment are specific to the environment it was engineered for and will vary for each environment.

A scan of the available literature reveals articles detailing the specific implementations and challenges faced engineering a standard operating environment however available information on the high level design and engineering process is scarce. In this paper a maturity model for the level of design sophistication of SOEs is initially presented. The maturity model can be used by SOE architects in the early formation of their design. A high level architectural model for the design and engineering of a SOE is proposed. Components within each layer of the architectural model will be identified and how the components integrate together within their layer and the layers between to form the overall standard operating environment.

For each of the layers presented important design techniques will be presented that will show how IT Service Management (ITIL) processes and principles can be incorporated into the design of the standard operating environment resulting in high availability and quality assurance. Key concepts to be covered include: the standard operating environment reference model; application dependency trees, permutation matrices for application deployment and testing; the duality of open versus high availability configurations for standard operating environments and the IT operational support implications required for each.

1. Introduction

The design and build of an SOE is a resource intensive exercise that ultimately aims at reducing the total cost of ownership (TCO) for the provisioning of the IT hardware platform (e.g. server, desktop PC), the associated IT services provisioned from the resource and reducing the support requirements from users. To achieve these aims the SOE must: be consistent across the deployment profile, provide high levels of reliability, robust to withstand the demands of users and other associated unmanaged changes and provide repeatability for IT staff in deployment across different hardware platforms.

By identifying and separating the discrete layers within an SOE provides advantages during the design and engineering stages, facilitates ease of maintenance and

enables IT staff a mechanism to assess the relative impact of change requests made by users.

The increase in popularity and adoption of the IT Infrastructure Library (ITIL) [1] IT Service Management (ITSM) by organisations and their IT departments has enabled IT staff, customers and users to communicate in a common language. Using the common language of ITIL enables IT personnel to translate customer requirements and expected service levels for IT services into technical solutions that meet their needs.

Using principles from the architectural model and framing them in the common language of ITIL IT personnel can effectively engage their stakeholders when negotiating change requests for an SOE.

2. SOE Maturity Model

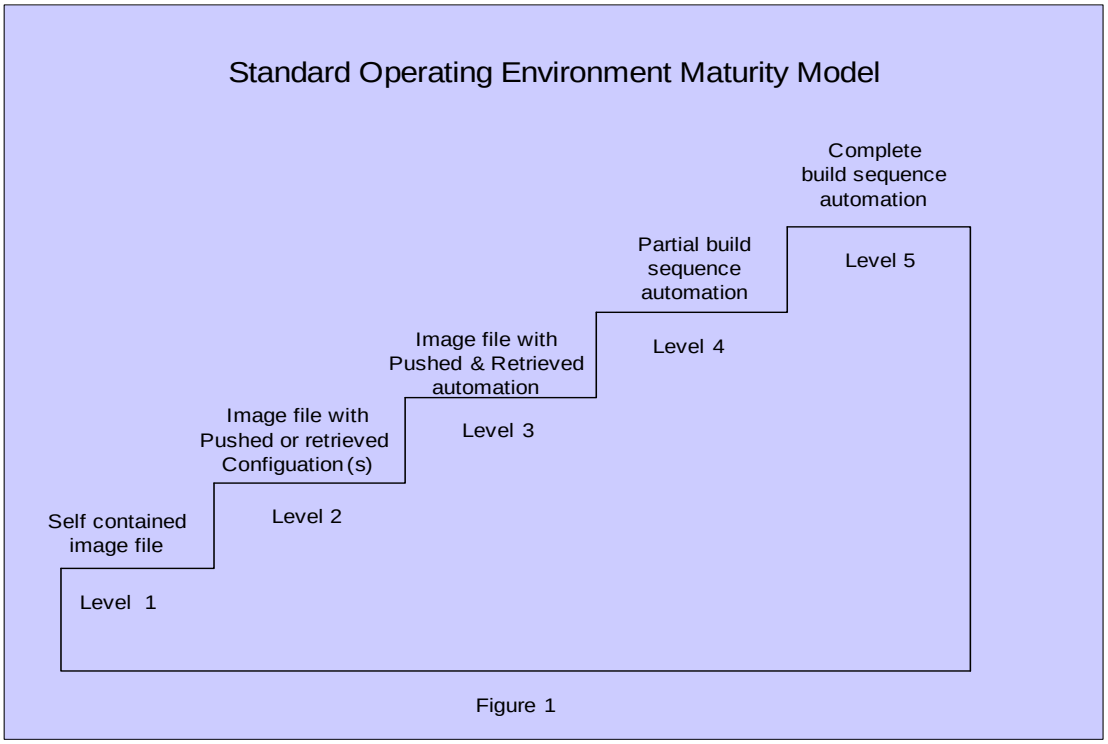

Figure 1

Applying the Software Engineering Institute's (SEI) [2] process maturity model concept figure 1 shows a maturity model for the level of automation [and design sophistication] incorporated into an SOE. The proposed architectural model for SOE design spans across the maturity spectrum. The architectural model can drive process improvements in an existing SOE or set the maturity context for a new SOE.

Level 1: Self contained image

Contained within this configuration of an SOE is: the operating system, all required applications and configurations. Generally no automation will exist other than, potentially, basic post SOE deployment scripting to tailor the SOE to the specific environment. This level of maturity for an SOE is applicable for environments where there is a single hardware platform and the application base is static with a minimal number of change requests. The final SOE is formed by the contents of the delivered image file.

Level 2: Image with pushed or retrieved configuration(s)

An image file containing the operating system, a standard suite of applications and configurations that applies across the deployment profile. Using a remote delivery mechanism either the deployed system can retrieve and/or the user can push further applications/configurations to their environment. The final environment is formed by the contents of the base image file and remotely retrieved applications/configurations.

Level 3: Image with push and retrieve automation

An image file containing a kernel environment consisting of the operating system that will have enough functionality to facilitate the further delivery of configurations /applications. The configurations and applications delivered are pre-packaged self contained units. The selection of configurations/applications delivered is based on one or more rule sets applied prior to their delivery. This functionality is achieved by embedded logic in the deliver mechanism. The final environment is formed by the contents of the delivered image file and the remotely delivered applications/configurations.

Level 4: Partial automation

At maturity level 4 the SOE moves away from being formed through the deployment of an image file and enters the realm of automated build sequences. At maturity level 4 the level of automation can exist either in the delivery and build of the operating system or the applications and configurations. The build sequence for the SOE automated.

The operating system contained within the SOE could be delivered via an automated installation. Once the build of the operating system is completed a layer of applications and/or configurations is pushed [or retrieved] to form the final environment. In this scenario the final environment is formed by an automated operating system installation with a pre-built layer of applications/customisations.

A second possible variation could include the downloading of the operating system with the automation provided in the installation of applications, generation and deployment of configuration items. In this scenario the final environment is formed by a pre-built operating system and automated installation of applications/configuration items.

An SOE with maturity level 4 the final build environment is formed from a pre-built portion and the automated installation of the remaining portions.

Level 5: Complete automation

At maturity level 5 complete automation exists. The operating system is customised for all possible configurations within the deployment profile and is installed via an automated installation sequence. Application installations are automated and delivered post operating system build. Configuration items are either dynamically generated and delivered to the environment or all possible configuration item combinations are accounted for and have packages developed for their delivery. The build sequence that will control the formation and delivery of the SOE is completed automated.

An SOE with maturity level 5 the final environment is formed from an automated operating system installation, automated application installation, customisations generated dynamically or all combinations generated and deployed to the environment and control of the build sequence is automated.

3. Architectural model

The architectural model for the design of an enterprise wide SOE is presented in Figure 2. The model presented is a layered architecture incorporating three layers. The model provides a framework to the engineering process and allows for the high level management of the initial engineering and ongoing support phases of an SOE. Moreover using appropriate elements of the model throughout an SOE's lifecycle brings a systematic process to effectively managing the evolution of the SOE and associated stakeholder requirements.

The layered architecture of the model enables the SOE engineering team to run development activities for each of the layers [and potentially within each of the layers] concurrently. This has the potential to reduce the development time for the SOE.

A modular approach is used in defining the requirements for each of the layers. Each module component within a layer can be analysed and the mechanism used to develop the component assessed. This approach to component building within the layers of the model provides the SOE architect with a number of options to reduce the time required to build the SOE.
Available options could include:

- Assign component modules to independent engineers or other build teams
- Procure component modules from vendors or suppliers
- Outsource the building of component modules
- Apply existing component modules

Each layer of the model successively builds on the layer below through the deployment and application of its own component objects. Once a layer has completed applying its own component objects control is handed over to the layer above to continue the build sequence. Through the interaction between the layers, the final build environment is formed.

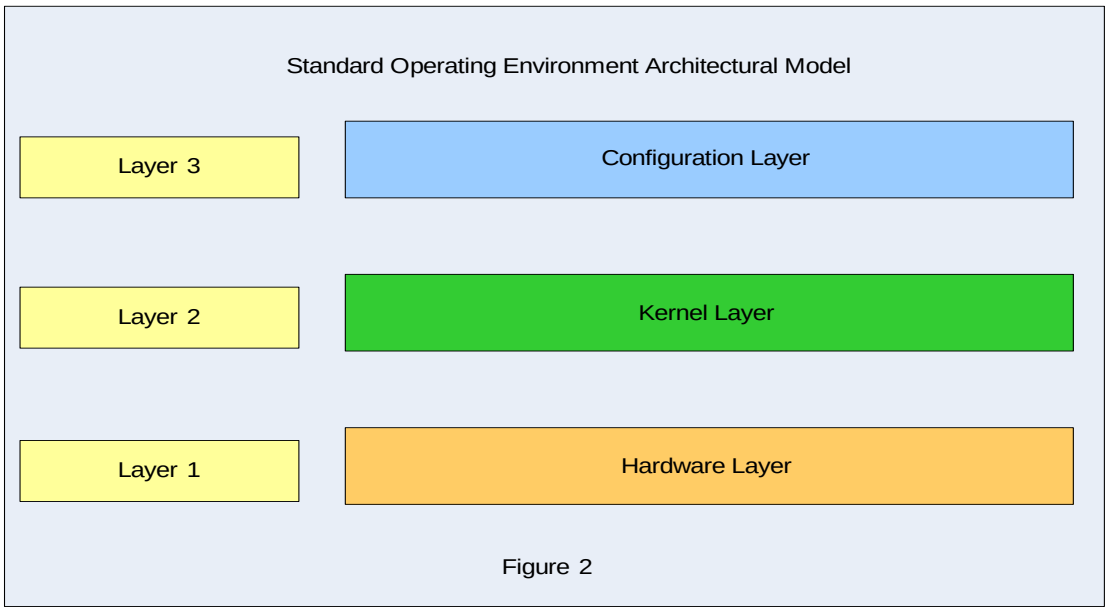

Figure 2

Layer 1: Hardware Layer

The hardware layer represents the physical hardware the SOE will execute from. The abstraction of the physical hardware to a layer within the model facilitates the inclusion of heterogeneous hardware platforms in the design process. Designing for heterogeneous hardware platforms increases the level of flexibility and adaptability of

the SOE. Moreover the life of the SOE is not limited to the life of the physical hardware platform. The underlying hardware can change with minimal changes required to the SOE.

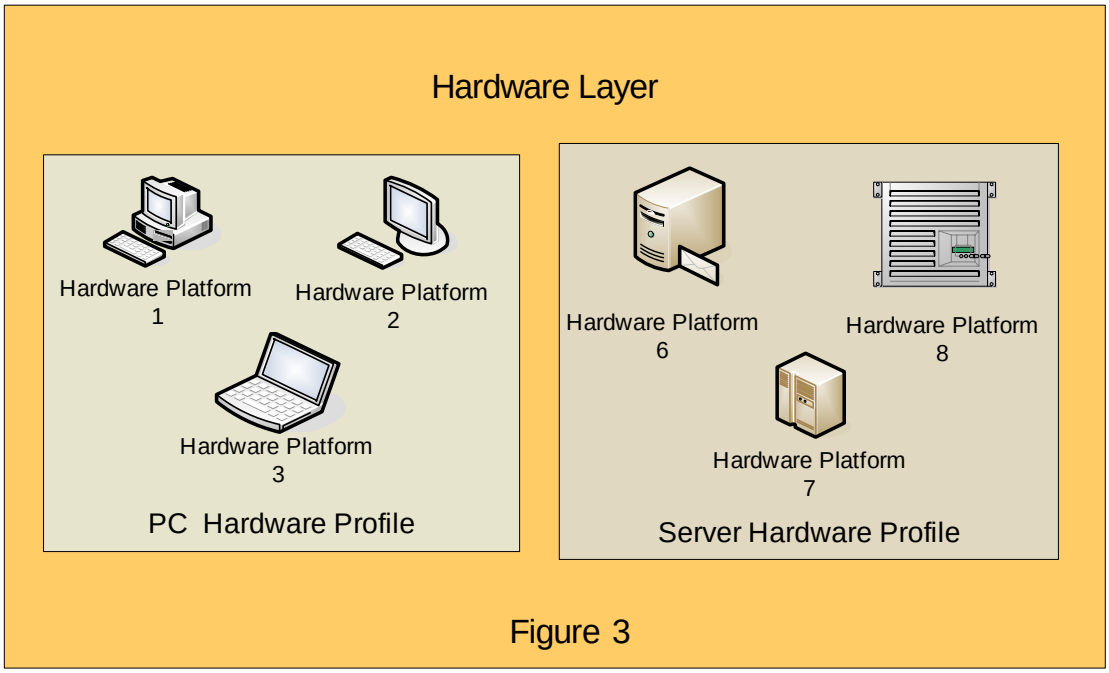

Figure 3

Figure three shows the concept of a hardware profile and a hardware platform. The term hardware profile is used to describe a collection of one or more hardware platforms an SOE will execute from. The term hardware platform is used to describe a unique hardware model an SOE will execute from.

For example in the case of a desktop PC environment a laptop would be one hardware platform and a desktop PC would be a second [and unique] hardware platform. Taking the hardware platform to a finer level of granularity, if a hardware vendor has two laptop models within a product range and both laptop models will be included within the hardware profile then each laptop model represents a hardware platform. Finally if four hardware platforms were potential deployment targets for an SOE, the hardware profile would consist of four hardware platforms.

Figure three shows two hardware profiles. Hardware profile one contains three PC hardware platforms. The second hardware profile contains three server hardware platforms. An SOE will only ever be engineered for only one hardware profile. Note the relationship between the hardware profile and the hardware platforms. One hardware profile will contain multiple hardware platforms. The concept of a hardware profile is easily transferable to server, mobile, network and communication devices.

Layer 2: Kernel Layer

Figure Four shows the sub-components of the kernel layer. The kernel layer of the SOE consists of the operating system, the operating system's patch level and a minimum subset of hardware device drivers. The packaging of the kernel layer can either be done via a pre-built generic, image file or an automated installation sequence. Primarily all hardware device drivers can reside in the configuration layer of the model except for the network interface card (NIC). The inclusion of the NIC device driver is to facilitate the delivery of the configuration layer during the SOE's build sequence.

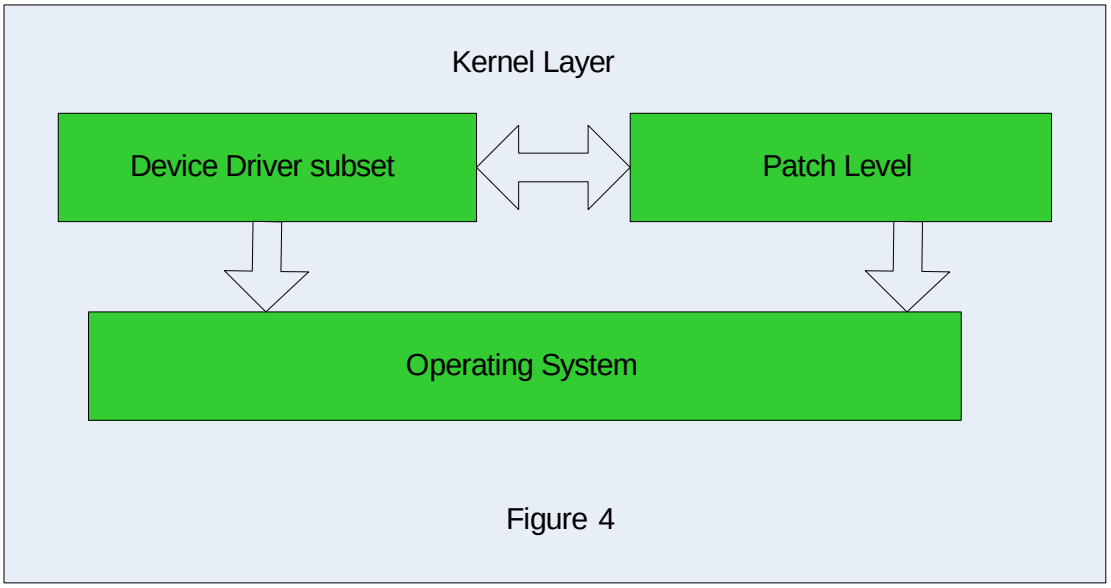

Figure 4

The arrows between the sub-components of the kernel layer indicate the inter-relationships between the components. Generally, a device driver is designed for a specific operating system. The device driver software released by a hardware manufacturer can potentially be released specifically for a required patch level of an operating system and hence the relationship between the device driver and patch level.

If the SOE hardware profile contains a diverse range of hardware platforms with each platform having a different model of NIC it is recommended the NIC drivers be installed at build/deployment time of the kernel layer. Having the NIC driver installed at deployment time will add flexibility to the solution developed for the kernel layer. The trade-off for the added flexibility is an increase in the time and resources required for the building of an automated install package for the NIC drivers [and associated quality assurance testing] during the engineering process. The development of an automated install package for the NIC drivers can be done in isolation and once completed the install package can be injected into in the driver subset module.

The time and effort required at this layer to add a new hardware platform is minimal due to the modularity of the layer. Providing the hardware platform is certified to run the designated operating system the only requirement is to develop an automated install package for the NIC drivers. Injecting the package into the kernel layer solution facilitates a reduction in the number of changes required at this layer resulting in reduced test and development times and change management requirements.

Layer 3: Configuration Layer

Figure five shows the three different component object types available within the configuration layer: application packages, configuration items and customisations.

A component object is a self contained package. Each component object can be delivered to the kernel layer in singularity or if dependencies exist between the component objects all required objects will be delivered in a pre-determined sequence.

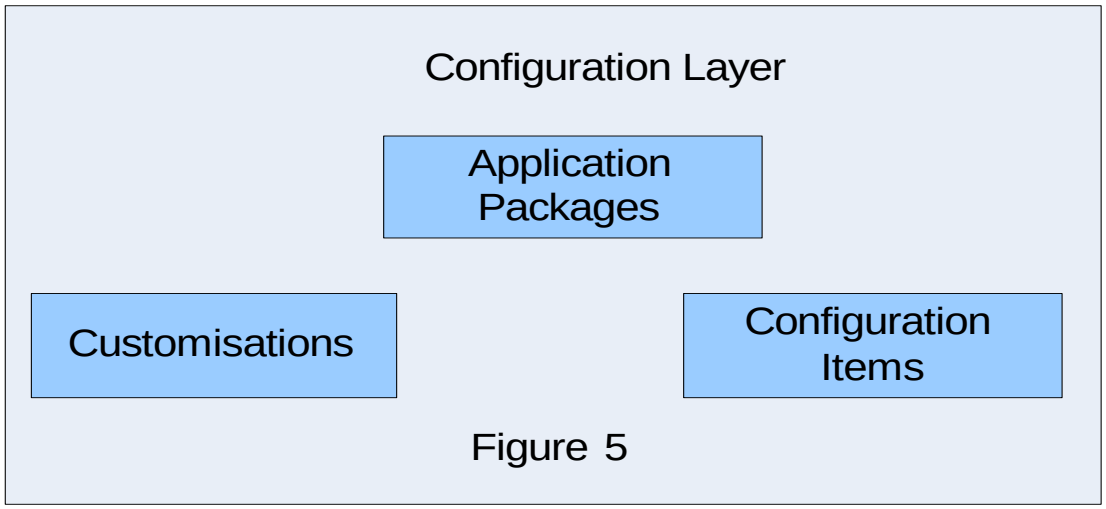

Figure 5

Application packages contain a single application that is required to be made available as a component of the SOE. The ideal configuration of an application package is the complete automated installation of an application with only the application files, configuration files [to configure the application during the installation] and any associated script(s) to initiate and guide the installation sequence. Using this method for application packaging provides numerous benefits including:

- A high level of flexibility in the delivery of the application. As the application package is installed as part of delivery it is not specific to a particular operating system. Therefore, the package can be used across a number of different operating system types.
- Ease in maintaining the application and associated upgrades. Numerous software vendors now allow for the applying of patches directly to their applications through updating the installation files. If a patch can be applied to the application package all that is required is the redeployment of the application package instead of the generation of a new a package. Moreover the updated application will only require a single delivery sequence.
- Through the use of configuration files, multiple different installation configurations can be made available using the common installation files.
- The installation package, if delivered using certain technologies, has the ability to self heal if it application becomes corrupted or damaged.
- Removing the dependency between the application package and delivery mechanism allows for greater flexibility. The delivery mechanism can be changed or upgraded without an impact on the application package.

Examples of application packages include device driver install sets, application plug-ins and application programs.

Customisations are generally some form of tailoring an application, operating system or a feature set of the SOE. Customisation packages are specific to [targeted] user groups and allow for the presentation of a unique view of the SOE.

Configuration items provide necessary details for the tailoring of, enabling or disabling of feature sets. Configuration items can also control the environment presented to the user. For example in a back-end server farm a configuration item could be used to remove or disable a service within SOE.

Dependency relationships can exist between the three difference component objects within the configuration layer.

4. Build Sequence of the kernel and configuration layers

The term build sequence refers to the sequence of events for the delivery of the kernel and configuration layers. The build sequence controls the sequencing of events. The build sequence for the delivery of the kernel and configuration layer follows a bottom up approach. At the hardware layer, a hardware platform first receives either an image file or an automated operating system install from the kernel layer. The kernel layer performs all necessary tasks to complete phase one of the deployment. This may include automated scripting to perform preliminary customisation and the delivery of a core device driver set to enable specific hardware devices required for the next step in the build sequence. The second phase of the build sequence is the delivery of component objects from the configuration layer. During this phase of the SOE build one or more component objects are delivered to the hardware platform.

The delivery of component objects can either be initiated by an automated sequence contained within the post kernel layer delivery tasks, a management application, or manually retrieve by the end-user. The delivery of the component objects can be completed either at the initial build of the SOE or post build with the SOE in live production.

5. Application Dependency Trees

Within the proposed model the majority of components are delivered to the build from the configuration layer. Depending on the number of required component objects a high level of complexity can exist within the configuration layer. There are two types of complexity that can exist: vertically complexity and horizontally complexity.

Vertical complexity in the configuration layer occurs when specific relationships exist between two or more application component objects and the level of operational effectiveness is determined by the delivery sequence of the objects.

Horizontal complexity in the configuration layer occurs when specific relationships exist between two or more component objects and the delivery sequence determines the level of integration between the component objects.

To aid in the control and management of the vertical complexity a systematic approach to the mapping of the vertical relationships between component objects is required. Application dependency trees are a technique developed to map the vertical relationships. Once the relationships are mapped the design team can determine the correct delivery sequence for the objects. [The mechanism to resolve horizontal complexity is presented in the permutation matrices section below.]

Let us assume you have twenty component objects that are delivered from the configuration layer. How does one determine the sequence in which these objects should be delivered? A closer study of the problem reveals two possible scenarios. The first scenario is where an object is deemed to be standalone. It is unique and does not depend on any other object. Therefore no relationship exists for this object. The second possibility is before an object can either be installed or executed another object needs to be installed. This situation is categorised as the object having a dependency on another object.

Consider the following scenario to further demonstrate a dependency relationship. A personnel management system client application is used to update employee records on a database. The client application remotely accesses and updates a database

through an ODBC connection. For the ODBC connection to be established the database vendor's client software needs to be installed and the ODBC data source configured. In the context of the configuration layer these three requirements would translate into two application packages (client access software for the personnel system and the database vendor's client) and one configuration item (ODBC data source configuration). The specific relationship that exists between the three configuration objects determines the delivery sequence:

1. Database vendor's client software
2. ODBC data source configuration is setup
3. Personnel management system client can be installed

Figure 6 shows the mapping of the delivery sequence.

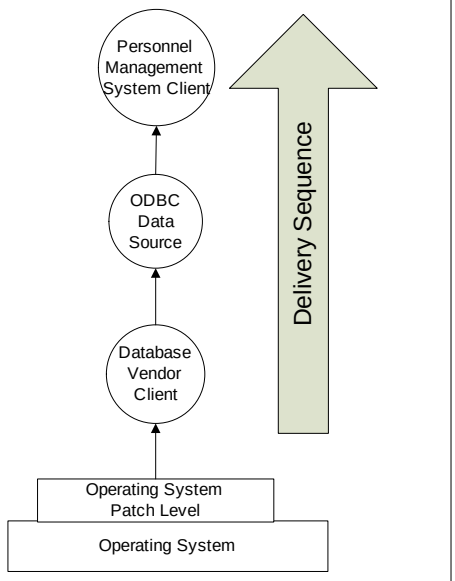

Figure 6

Figure 6 is a simple example of a dependency tree. The example presented shows a linear relationship between three configuration objects. One of the key points to identify is at the root of the dependency tree is the SOE's operating system and associated patch level. These two components are the foundation for all configuration objects and are at the root of all dependency trees.

Figure seven shows a small scale dependency tree for an SOE. Within the tree five of the most reoccurring dependency relationships are shown (VR1 through to VR5). Working through the five relationships in figure seven from left to right each of the relationships will be discussed. An example of dependency one (VR1) was shown in figure six. This relationship is the simplest dependency relationship that can exist and is the easiest to implement, maintain, manage and support over the life of the SOE. The linear relationship enables the impact of changes made to objects within the relationship easy to asses and diagnose.

Dependency relationship two (VR2) is an example of where multiple dependencies exist at more than one layer within the dependency tree. To add further complexity there are three sub-relationships contained within the overall dependency:

- Two burst dependencies (the first at VR2, AM7 & AM8 and the second at AM7, A2 & A3).
- Convergent dependency at AM7, AM8 & AM3.

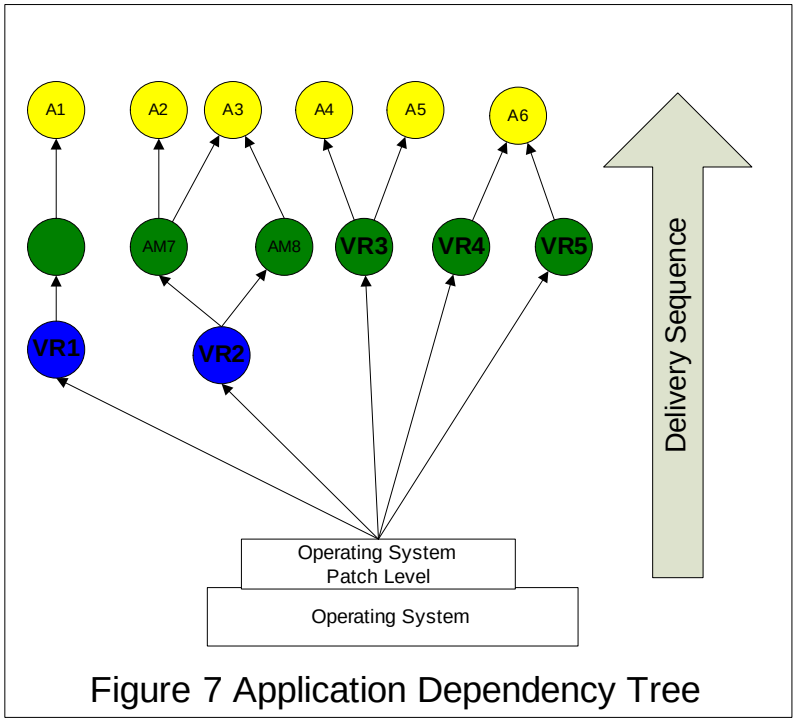

Figure 7 Application Dependency Tree

A burst dependency occurs where two or more applications are dependent on a singular parent application. A convergent dependency occurs where a dependent application is dependent on two or more parent applications.

Consider the following example to further explain the VR2 dependency. Application three (A3) represents a web browser that must be used to access a corporate website application. Application two (A2) is an image capturing application used to access and scan documents from a personal scanner. The green application on the left (AM7) is an application that generates portable document format (PDF) files with A2 integrating into AM7 to allow for the scanning of images directly into AM7. The green application on the right (AM8) is a bibliographic management application. Finally the application at the root of the dependency (VR2) is a suite of office applications (e.g. MS Office 2003).

VR2 is required to be installed prior to any of the dependent applications. At install time applications AM7 & AM8 detect the presence of VR2 and install required components that integrate into VR2. Finally once VR2, AM7 and AM8 are installed applications A2 and A3 can install and integrate with their respective parent applications to provide the required level of functionality.

Dependency relationship three (VR3) is an example of a burst relationship where applications A4 & A5 are dependent on application VR3. An example of this relationship would exist where two client applications; an E.R.P. client and a Library management system client, require the same database vendor client.

Dependency relationship four & five (VR4 & VR5) are an example of a convergent dependency. An example of a convergent dependency was covered as a sub-dependency in dependency relationship VR2.

6. Permutation matrices for application testing

The technique of mapping vertical application relationships using a dependency trees [covered in section 5] requires the dependencies between component objects to be

known. Dependencies are not always known for a variety of reasons or a direct dependency between component objects may not exist however they may share a common O.S. resource. In either of these two scenarios another technique is required to ensure effective delivery and operation of component objects.

A comprehensive test schedule that includes sequencing the delivery of component objects to the build environment can assist in revealing hidden dependencies. The test schedule should allow for all plausible delivery sequences of component objects to the SOE. The result of a completed test schedule will reveal hidden/unknown dependencies or certified component objects [and their level of integration] to SOE. The requirements for the test schedule include:

- Test the delivery of all plausible component object combinations.
- Test the delivery sequence of component objects.
- Capable of identifying hidden/unknown dependency relationships between component objects

Definition: A permutation is an arrangement of distinct objects in a definite order [3].

Formula for a Permutation of *n* distinct objects taken *r* at a time [3]

$$P(n,r) = \frac{n!}{(n-r)!}$$

n is the number of objects that are available to be chosen
r is the number of objects that are chosen.
n! is n factorial

Example 1.	**Example 2.**
n := 4 r := 4 and so $P(4,4) = \frac{4!}{(4-4)!}$ $= \frac{4!}{(0)!}$ $= 4! = 24$ There are **24** permutations of 4 objects taken 4 at a time.	n := 20 r := 20 and so $P(20,20) = \frac{20!}{(20-20)!}$ $= \frac{20!}{(0)!}$ $= 20! = 2.4 \times 10^{18}$ There are **2.4×10^{18}** permutations of 20 objects taken 20 at a time.

Once the design team has a defined list of all required component objects, their vertical relationships are determined and delivery packages are developed using permutations a test schedule can now be developed.

Assume a design team has 20 component objects, accounting for vertical dependencies between the objects the component objects the twenty component objects form four distinct delivery packages. Table 1 lists the applications and associated dependencies and Figure 8 shows the dependency tree.

Application Name:	Dependency to	Application Name:	Dependency to:
Application 1 (A1)	-	Application 11 (A11)	-
Application 2 (A2)	A1	Application 12 (A12)	A11
Application 3 (A3)	A1	Application 13 (A13)	A11
Application 4 (A4)	A2	Application 14 (A14)	A13
Application 5 (A5)	A3	Application 15 (A15)	A12,A13
Application 6 (A6)	-	Application 16 (A16)	-
Application 7 (A7)	A6	Application 17 (A17)	A16
Application 8 (A8)	A7	Application 18 (A18)	A16
Application 9 (A9)	A6	Application 19 (A19)	A17, A18
Application 10 (A10)	A9	Application 20 (A20)	A16
Table 1: Application and dependency list.			

From Figure 8 we can see the four delivery packages (A1, A6, A11 & A16) form the main branches within the dependency tree.

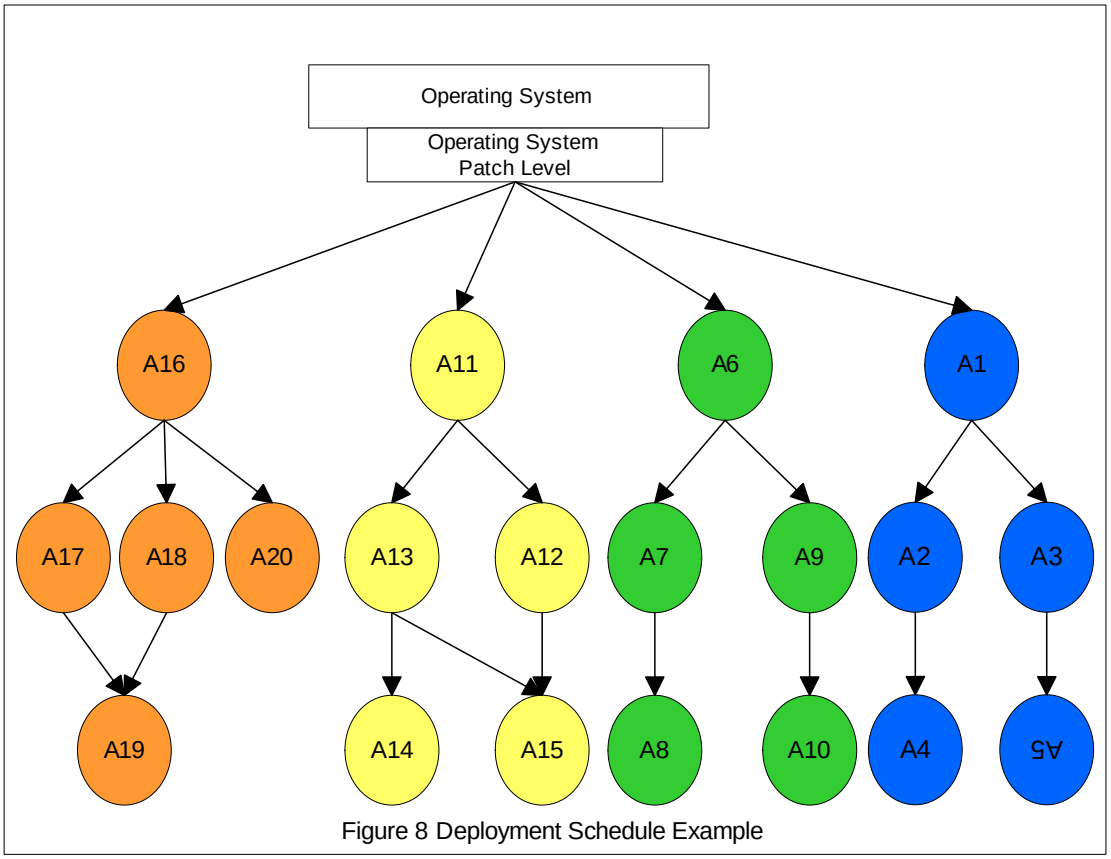

Figure 8 Deployment Schedule Example

To determine the required number of test sequences for the four delivery packages using the formula for permutations from above we have:

$n = 4$

$r = 4$

$$P(4,4) = \frac{4!}{(4-4)!} \Rightarrow \frac{4!}{(0)!} = 4! = 24$$

The test schedule will have **24** distinct test scenarios. The test schedule for the example in figure 8 is presented in table 2.

	Test Schedule #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Delivery order	1	A1	A1	A1	A1	A1	A1	A2	A3	A4	A5	A6	A7	A3	A3	A3	A3	A3	A3	A4	A4	A4	A4	A4	A4
	2	A2	A2	A3	A3	A4	A4	A1	A1	A3	A3	A4	A4	A1	A1	A2	A2	A4	A4	A1	A1	A2	A2	A3	A3
	3	A3	A4	A4	A2	A2	A3	A3	A4	A1	A4	A3	A1	A2	A4	A1	A4	A1	A2	A2	A3	A3	A1	A2	A1
	4	A4	A3	A2	A4	A3	A2	A4	A3	A4	A1	A1	A3	A4	A2	A4	A1	A2	A1	A3	A2	A1	A3	A1	A2

Table 2: Application testing permutation matrix

7. Dependency Trees, Permutation Matrices & ITIL

Within the ITIL framework there are two process groups: Service Support processes and Service Delivery processes. Change and Release management processes are two of the five processes with the Service Support process group.

One of the aims of the change management process is to minimise the adverse impacts of changes introduced into the live production environment. The release management process aims to manage the introduction of changes into the live production environment. The following sub-processes are within the release management process: the planning of release packages; build and configure release packages, development and implementation of test schedules for release packages.

Application dependency trees and permutation matrices are tools that support the change and release management processes of ITIL. Through analysis of dependency trees SOE engineers are able to asses the potential impact change requests will have for an SOE. Rollback strategies for unsuccessful changes can easily be developed using dependency trees. Permutation matrices provide a toolset for the development of comprehensive test schedules that can be used within the sub-processes of the release management process.

Using application dependency trees & permutation matrices and SOE team can effectively support the implementation of ITIL processes Change & Release management for the effective management and support of an SOE.

8 High availability versus Open configuration SOEs

A high availability SOE [also referred to as a locked down SOE] is one where the available functionality to the user is limited. The user of a highly available SOE configuration is presented with a predefined number of applications and has no ability to install, remove or upgrade the application base. The SOE's configuration will limit the level of access the user has to the O.S. and their ability to make changes. The file system, generally, is protected through the application of access control lists and the user is unable to save, add, delete or modify.

An open SOE is one where the user has the ability to install, remove or upgrade the applications contained within. The user has the ability to make modifications to the O.S. and associated settings. The user has the ability to add, delete or modify the file system and to store data locally.

A high availability SOE configuration requires a large investment in time and resources to engineer. The level of investment required centres around the need to resolve technical issues when applications are installed into the lockdown environment and the required level of testing to ensure all required feature sets work as per stakeholder requirements. The return on investment (ROI) for the time, effort and resources made during the engineering phase of the SOE's lifecycle is a significant reduction in 'break-fix' support required from IT staff post implementation.

An open configuration SOE requires a reduced level of investment in time, effort and resources, when compared to a high availability configuration, to engineer. The reduced level of investment is achieved through a reduction in the number of technical issues that are required to be resolved throughout the engineer process. The level of testing required to ensure operational effectiveness is reduced. An open configuration SOE requires a shorter development time. The savings made in developing an open configuration SOE are counterweighted by an increased level of support required from IT staff to resolve break-fix support requests from users post implementation.

Due to the open nature of the environment users of the SOE are able to make untested changes to the environment that potentially have adverse effects and require IT support staff to resolve.

User requirements and the IT resources available to support an SOE ultimately determine how open an SOE's configuration is made. The level of openness an SOE is configured for will determine the level of IT support resources required to maintain the environment in live production.

The level of openness and associated IT support requirements for an SOE ultimately determines the SOE's TCO and ROI.

9 Conclusion

The design and engineering of an enterprise wide standard operating environment is an intensive exercise that requires a strong commitment from all levels of the organisation. The design phase is the most crucial of phases in a SOE's lifecycle. During the design phase and the engineer's ability to able to listen to stakeholder requirements, translate requirements into complex technical solutions that are operationally supportable by IT staff, understand and assess the financial impact of design decisions will ultimately determine the longevity and sustainability of an SOE.

Using the maturity model, architectural design model and testing tools an SOE architect will be able to design an SOE that will meet end-user, IT support staff and business owner requirements while delivering an acceptable ROI and TCO.

References

[1] Office of Government Commerce 2006, Information Technology Infrastructure Library. Retrieved November 4, 2006 from http://www.itil.co.uk/

[2] Carnegie Mellon Software Engineering Institue 2006, Capability Maturity Model Integration (CMMI). Retrieved November 4, 2006 from http://www.sei.cmu.edu/cmm/

[3] Aufmann, RN, Barker, VC, Nation Jr, RD 1991, PRECALCULUS, Houghton Mifflin Company, Boston, p. 601