Visual Panels

LSARC 2007/392

Dave Powell <David.Powell@Sun.COM>
Steve Talley <Stephen.Talley@Sun.COM>
Tony Nguyen <Truong.Q.Nguyen@Sun.COM>
Justin English <Justin.English@Sun.COM>
Jaime Guerrero <Jaime.Guerrero@Sun.COM>

Visual Panels: LSARC 2007/392

by Dave Powell, Steve Talley, Tony Nguyen, Justin English, and Jaime Guerrero

Table of Contents

1. Overview 1
2. Terminology 2
3. User experience 3
    - Accessing Visual Panels 3
        - Remote access 3
    - Navigation 3
        - Categories 3
        - Services 4
        - Anomalies 5
        - Future Work 5
    - Content 5
        - Status 6
        - Objects 6
        - Global settings 7
        - Future Work 7
    - Desktop Integration 7
        - Look and Feel 7
        - Direct Access 8
        - Future Work 8
4. Architecture 9
    - Publishing Configuration 9
        - Interface technology 9
        - Mapping configuration 9
        - Deployment 11
    - Presentation 11
        - Panel Description 11
        - Panel MBeans 11
    - Access Control 12
        - Connection 12
        - Authentication 12
        - Authorization 13
5. Panels 14
    - Apache web server 14
        - Description 14
        - Configuration access 14
    - File sharing 14
        - Description 14
        - Configuration access 15
    - Internet services 15
        - Description 15
        - Configuration access 15
    - Core files 15
        - Description 15
        - Configuration access 16
    - SMF 16
        - Description 16
        - Configuration access 16
6. APIs 17
    - Panel / Components 17
        - Publishing APIs 17
        - Data access APIs 17

UI APIs 17
Other consumers 17
MBean APIs 17
JSCF 17
7. Man pages 18
vp 19
8. Interface tables 20

List of Tables

8.1. Exported Interfaces 20
8.2. Imported Interfaces 20

Chapter 1. Overview

Essential to the adoption of an operating system is the ability to easily configure that system and the services is offers. Of course, the definition of “easily” varies significantly with the experience of the user. Solaris (and therefore OpenSolaris) has enjoyed considerable success with those customers who are skilled in the ability to configure a complex Unix system. To attract newer, less specialized users, as well as to expand the capacity of existing OpenSolaris administrators, it is necessary to consider expanding the set of GUI tools available for configuring OpenSolaris.

Other operating systems have made considerable progress in this area. This isn't to say the Solaris hasn't, but our previous attempts have been limited by a number of factors:

- **Isolation from other tools.** We have created standalone tools which are launched from different locations and have different UIs. Performing multiple configuration tasks might involve switching from one tool to another, and the experience gained using one tool may not help the user in using the next.

- **Poor interface usability.** Some of our tools have a slow, tedious, or flaky user interface which discourages their use, even for those cases where the tool would be effective.

- **Lack of content.** Setting aside any subjective assessment of past configuration tools, one can objectively state that coverage over system/service configuration is far from complete. Standalone UIs don't offer a strategic foundation for increasing our coverage, and our attempts to provide open-ended frameworks leave the task of creating content to others. While allowing others to create content isn't a flawed idea, relying on it is.

- **Difficulty of producing content.** It is imperative that a framework that expects to be adopted by third-parties (either within Sun or without) not require an investment that exceeds the benefits obtained by using the framework. This should be accomplished by providing a good set of examples (see previous point) and, frankly, by providing simple interfaces.

- **High maintenance costs.** Stand-alone UIs offer no efficiencies in terms of maintenance, and systems which themselves represent a significant abstraction of the system's configuration don't either. It is essential to leverage basic system abstractions to the greatest degree possible, and to sediment new abstractions into the system where they are absent.

- **Expectation of ability.** Some UI efforts have focused on providing access to all functionality in the UI, when, in fact, one must carefully evaluate what functionality will be needed to avoid overwhelming the user and reducing the efficacy of the tool.

Visual Panels will deliver GUI configuration for OpenSolaris, while attempting to avoid the above pitfalls. Key components of Visual Panels are: an initial set of content oriented towards meeting marketing and developer audiences; a framework which will subsequently be formalized and made available to third parties for development; an underlying architecture that can be reused for other system management purposes; the ability to configure remote machines as easily as local machines; and a design which takes advantage of recent additions to OpenSolaris (e.g. SMF) to greatly reduce the amount of work required to publish content into the provided framework.

Visual Panels is written in Java, and has dependencies on the existing Cacao Common Agent Container and the upcoming Cacao webserver module. Visual Panels seeks approval for inclusion in a micro release of OpenSolaris.

Chapter 2. Terminology

The terminology used in the problem areas Visual Panels tackles is unfortunately heavily overloaded (especially in OpenSolaris). Within this document, we will attempt to consistently use these ambiguous or novel terms in a manner consistent with the following definitions:

Panel	The entire configuration UI for a particular service
Service	A service offered by the system to clients running on other machines or to clients running on the system itself
SMF service	An SMF service instance
Solaris	The Sun distribution of the OpenSolaris operating system.
Visual Panels	The Visual Panels UI and all related infrastructure
Visual Panels UI	The user interface component of Visual Panels

Chapter 3. User experience

The first impression any user interface makes is the experience it offers to the user. Visual Panels will give the user a straight path to the configuration they wish to modify, and present that configuration in a manner that balances consistency and the need to tailor the display of information for a particular purpose. Additionally, it attempts integrates both core OpenSolaris concepts and the desktop environment.

We have divided the Visual Panels user experience into four pieces: how the Visual Panels UI is invoked by the user, how the user find what they are looking for in Visual Panels, the configuration experience itself is like, and how Visual Panels can be integrated into the desktop environment.

Accessing Visual Panels

The Visual Panels UI can be started either by selecting its entry from the Administration submenu of the Gnome launch menu, or by running **/usr/bin/vp** from a shell prompt. The user will immediately be presented with a console providing whatever access they are permitted.

If the user wishes to make changes to the system's configuration in excess of their privileges or authorizations, they can request to log in as a different user (e.g. `root`). Logging in as a different user requires supplying that user's login name and password. When the user is finished operating as the alternate user, they can either close the Visual Panels UI (the Visual Panels UI *always* defaults to the user's view of the system), they can revert to the original user, or they can choose a different user to operate as.

Details about the authentication performed by Visual Panels can be found in the Security section of the Architecture chapter.

Remote access

In addition to allowing the user to configure the local machine, Visual Panels permits the user to connect to and configure remote OpenSolaris machines. In all cases, connecting to a remote machine requires providing a username and password, even if the user is logging in using the same principal used locally.

The Visual Panels UI is only able to communicate with other OpenSolaris machines running a current or previous version of the Visual Panels software.

Navigation

Before a user can begin to configure the system, they need to find the configuration they wish to change. This is as important as the configuration task itself.

Categories

The Visual Panels UI uses a category-oriented navigation scheme. Each configuration panel is assigned to a category. These categories are listed on the left of the Visual Panels UI. Only those categories that contain panels will be displayed.

Category selection

When the users selects a category, the services in that category are displayed in the area to the right of the category list. When Visual Panels starts, the first category listed will automatically be selected for the user.

Services

The services available in a particular service are represented by a vertical list of of oblong objects we call “lozenges”. Each lozenge corresponds to one and only one service, and currently:

- shows the name of the service
- has a distinguishing icon for that service
- shows a helpful status summary
- provides a shortcut to common operations on that service (e.g. enable, disable)

Clicking on a lozenge will replace the list of services with the panel for the selected service.

Service selection (“lozenges”)

Generally speaking, the intent of the lozenge is to provide — in a uniform fashion — greater context than the traditional icon/name pair. The content of the lozenge is likely to change from what is enumerated above as we learn what is most helpful for users.

Anomalies

Note

The FTP panel shown in this and the “Objects” sections is for illustrative purposes only and is itself not part of this case.

Equally important to helping the user find the service they are looking for is helping a distraught service find the user. When some Visual Panels-accessible service requires attention, we utilize several visual cues to help the user identify those parts of the system which need their attention.

When a service has less than ideal health, its lozenge will be painted in a color corresponding to the degree of the problem (yellow for a minor problem, red for a major problem). Additionally, the icon will be badged in a way that can be identified by the color blind.

A lozenge highlighted to indicate a problem

In turn, a category which contains such a service will be badged appropriately. Therefore, a user can quickly identify when something is wrong with the system, and where they should go to identify the problem.

A category highlighted to indicate a problem

Future Work

Search

Eventually, Visual Panels will offer the ability to find services based on their names as well as aspects of their configuration. Due to the limited amount of content in the initial version of Visual Panels, this is not included at this point in time for both economic reasons (not yet worth the engineering cost) and to cut down on visual clutter (with few panels, the distinction between search and other navigation methods can become muddled.)

Favorites

Likewise, an increase in the amount of content will make it desirable to provide the user a means of quickly accessing frequently configured services. This functionality is also not included for the reasons cited under Search.

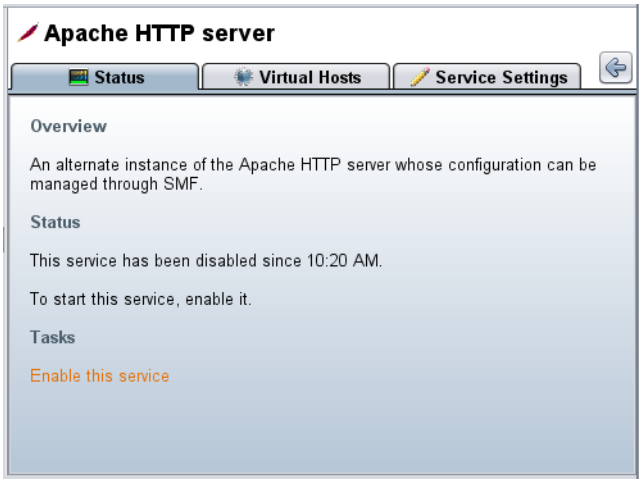
Content

When the user selects a service's lozenge, the lozenge view is replaced with the selected service's panel. There is a lot of information that needs to be conveyed by the typical service panel, so panels follow certain conventions for partitioning the presentation of that information. Specifically, the panel view is broken up into several tabs or sub-panels, each of which has a particular set of semantics. Not all panels will need all

the tabs, and the tabs might be labeled differently from panel to panel. However, the order, visual design, and task breakdown will remain consistent.

Status

The first tab will always be the Status tab. This tab gives a description of that service, a summary of the service's health, and access to any available tasks that can be performed on the service. In most cases, the tasks offered will just be to enable or disable the service. It is not required that even these be present; it is possible that a service panel will represent something that can't be turned off.

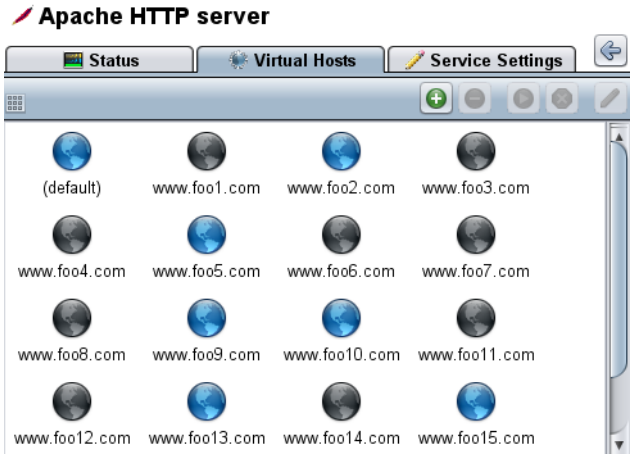


A service's status tab

Objects

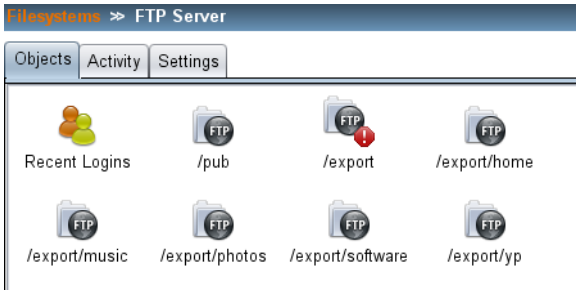
Many services contain, or have the ability to configure, a variable number of objects. Examples of services which have this relationship are file sharing (each share is an object), web serving (each virtual server is an object), and user management (each user is an object).

For such services we formalize the notion of an object and provide a tab specifically for managing the set of objects that belong to the service. This tab will provide a consistent UI for viewing and operating on a service's objects. Actions like adding an object, removing an object, and navigating to per-object configuration will behave the same from service to service.



A service's objects tab

For services which the UI has indicated are in an anomalous state, the responsible object may be highlighted by the service in its objects view.



An objects tab showing the specific source of a problem

Not all services have such a complex configuration. For such services, there will be no objects tab.

Finally, while the user's interaction with different services' objects may be very similar, the user's perception of these objects is almost guaranteed to differ. The objects tab will always be given a label which is specific to the purpose of the service. Using the above examples, the objects tab for file sharing would probably be called “Shares”, while the objects tab for a user management panel would most likely be named “Users”.

Global settings

For services which have configuration which spans the objects they manage, or for services which don't manage objects at all, Visual Panels provide a global settings tab. It is in this tab that normal “flat” configuration takes place. For those services which don't have a objects tab, the global settings tab will probably be labeled just “Settings”.

Future Work

Activity

Alongside status and configuration, it would be nice to also provide a place to observe service activity. This can include things like statistics on historical events, graphs of real-time behavior, and summaries of access logs. In part due to the lack of existing infrastructure for gathering and accessing historical data, and in part due to a desire to focus on the configuration problem, the initial version of Visual Panels won't be providing much by way of behavioral observability.

Nonetheless, we have put considerable thought into how this would fit into the Visual Panels UI and what form it would take for different services. We envision providing an “Activity” tab that would be the common location where a user could find such information.

Desktop Integration

Key to the success of a configuration UI is a feeling of being integrated with the system it's being used on. This doesn't stop with being accessible from the launch menu; appearance and other behaviors are also important.

Look and Feel

The Visual Panels UI instructs Java to use the native look and feel, so on the Gnome desktop it should look like a native Gnome application.

Direct Access

If an application requires the user to make a change to the system's configuration, it should be able to take the user directly to the configuration that needs to be changed. We should avoid instructing the user to do thing that the system can do for them whenever possible. To this end, it is possible to direct Visual Panels UI to open to a particular service's configuration automatically, without requiring the user to navigate to that panel.

Future Work

Notification Area

One of the key features of Visual Panels is how it leverages the SMF infrastructure to manage configuration as well as monitor the health of the system. As described above, anomalous system state will be communicated to a user who has the Visual Panels UI open. While this eliminates many of the manual steps from today's recovery process, the system still relies on the user to notice that something has gone wrong, or on the user fortuitously be using the Visual Panels UI at the right time.

One thing we would like to do with the Visual Panels UI is to give it a presence in the Gnome notification area. From here, we could notify the user of problems when they occur, thereby eliminating this last step.

Additionally, there are system components or services system which may have specific needs that warrant being present in the notification area. If these components are being managed by Visual Panels, we need a means of permitting them access to the notification area.

Chapter 4. Architecture

The primary purpose of Visual Panels is to expose OpenSolaris configuration to users. Naturally, care needs to be placed in how configuration data is managed to minimize the work required to publish configuration data, while offering flexible consumption of published configuration data.

For example, the Visual Panels UI is intended for a OpenSolaris end user. To serve that user well, we will be making user interface decisions that may be poor, for instance, for the administrator trying to manage a grid of machines. The configuration of the machine, however, remains a constant between these systems. It would be wasteful for us to spend time exposing OpenSolaris configuration and not also make it available to consumers who want to present it in a different form.

Publishing Configuration

The act of publishing configuration consists of several steps. First, a versatile interface needs to be chosen or developed. Second, a mapping needs to be created between this interface and the underlying configuration. Third, an implementation of these first two pieces needs to be deployed and made available to consumers.

Interface technology

The primary interface for accessing Visual Panels configuration is through Java Management Extensions (JMX) MBeans. MBeans:

- are a standardized Java technology (JSR 160) and are part of J2SE
- provide a means of defining system management interfaces
- provide an infrastructure for accessing those interfaces with custom protocols, independent of the interfaces themselves
- can be used to communicate within a single JVM as well as to JVMs *or other technologies* running on remote machines
- don't prescribe a set of standard interfaces into which OpenSolaris's needs must be fit

By using an MBean-based interface, Visual Panels is able to configure remote machines as easily as it can configure local machines. Additionally, configuration published by Visual Panels will be easily consumed by other management infrastructures (pending future ARC cases).

Mapping configuration

Categorizing configuration by its storage and usage attributes is convenient for both discussion and implementation. Each category of configuration data (not to be confused with the categories in the Visual Panels UI) is represented by an MBean interface, with each configurable element being an instance of that interface.

Service Configuration Facility (SCF)

The Service Management Facility (PSARC/2002/547) introduced SCF, the Service Configuration Facility, as a means of storing system configuration data. SCF offers typed, transactional configuration storage, and, when combined with SMF, also defines a lifecycle for service configuration. Moreover, as SCF is

improved, customers are often able to reap additional benefits without the owners of the configuration needing to provide explicit support for those benefits. All new OpenSolaris configuration is expected to reside in SCF per the SMF Usage Policy [http://opensolaris.org/os/community/arc/policies/SMF-policy/].

Because of the leverage it offers, its large and growing consumer base, and the generality of its interfaces, SCF is the preferred means of exposing configuration to the Visual Panels UI. Each SMF instance will be automatically published as an instance of an SMF MBean. Configuration stored in SCF will, in turn, be available to Visual Panels with no additional effort required.

Access to configuration stored in SCF is achieved through a package of Java routines that will eventually be independently ARCed, but will initially be project private.

Naming/Directory services

Unfortunately, not all configuration can be stored in the SCF. A prime example of this is information stored in or obtained from a directory service such as NIS or LDAP, or in a local database (e.g. `/etc/passwd`). Though the initial version of Visual Panels will not be offering the ability to modify the content of such data stores, accessing things like account data is required to provide a complete experience for our initial set of panels.

A Naming MBean is provided to publish this configuration data. As our initial demands on this configuration source are few, so will be the operations supported by this MBean.

Access to naming configuration is currently accomplished, in what is perhaps a misguided attempt to reuse existing code, through the same (private?) classes used by SMC. It is likely that we will implement something simpler, stabler, and Visual Panels -specific for commitment.

Other configuration

Some things, e.g. ZFS, are similar to Naming in that the configuration is represented in some medium that isn't easily mapped to SCF. Such configuration will most likely need to have MBeans written for them. Fortunately subsystems like ZFS and Naming are limited in quantity.

Another class of configuration is the application with its own complex configuration. Representing its entire configuration in SCF is either untenable due to its size or unmaintainable due to the stability of the (possibly third-party) application. In such cases we will need to examine use of the application, find a useful subset of the application's configuration, and concoct an SCF representation of a user's intent that spans those configuration areas. Such configuration will ultimately be made available through the standard SMF MBean.

One last class of configuration is that which is currently stored in SCF, but has a representation that is either very complex or private to the configuration owner's implementation. In such cases, it is a judgement call whether we expose that configuration via the SMF MBean or through a custom MBean. The former has the advantage of adding one less component that requires maintenance, whereas the latter can simply implementation and ease the maintenance of consumers of the configuration. Clearly, in cases where we are interested in creating an interface to be consumed by customers or third parties, we have no choice but to take the latter option.

Visual Panels MBean

Though MBeans have a name that can be used to access them if their existence is known *a priori*, in most cases it would be preferable to have some other mechanism to act as a directory for the MBeans described above (as well as the MBeans not yet covered). To satisfy this need, We publish a Visual Panels MBean that provides access to this meta-configuration.

Deployment

While configuration UIs will simply be consuming MBean interfaces, the MBeans implementing those interfaces are run in a server called an agent. Rather than create its own new system daemon, Visual Panels leverages existing infrastructure and runs its MBeans in the Cacao Common Agent Container (LSARC 2006/004). This reduces the impact Visual Panels has on the machine by — among other things — not running a separate JVM, and also offers economies as far as Visual Panels implementation is concerned.

Some of the features of Cacao that Visual Panels will be taking advantage of are:

- support for securing connections with TLS
- support for PAM authentication
- APIs for simplifying MBean development
- a webserver module for bulk data transfer (e.g. class files, images)

Cacao is currently shipped off by default. Visual Panels will require enabling Cacao.

Presentation

Regardless of how easy it is for Visual Panels to access configuration, some amount of manual effort will always be required to direct how that configuration is presented to the user. As discussed in the previous chapter, configuration is divided into panels which represent a cohesive unit of system or service configuration.

Panel Description

To reduce the coordination problems that arise when partitioning configuration and presentation, as well as when supporting remote configuration, the Visual Panels UI itself will not contain hard-coded implementations of these panels, or even have knowledge of their existence. Instead, it will read a set of panel descriptions that inform the UI what panels exist and how each presents configuration to the user.

A panel description contains the localized name of the panel, the category the panel belongs to, an icon, a set of associated SMF services, and most importantly a set of descriptors that define the appearance of the panel. We allow for a set of descriptors so that different versions of the panel can be provided for different environments.

There are two primary kinds of descriptor. The first is a “property sheet” descriptor, which instructs the UI to simply present the specified SMF service instance along with a localized form allowing modification of its properties. This is intended mainly for development purposes, and for simplified exposure of site-local service configuration within a site. The second is a “custom” descriptor, which provides a set of Java classes that implement a UI for a particular environment.

All content provided by this project will take the form of custom panels that deliver UIs implemented to the Swing toolkit. Property sheet panels, though a useful tool, do not provide an experience we think is appropriate for any content delivered by OpenSolaris itself.

Panel MBeans

Panel definitions, like any other piece of configuration, needs to be stored somewhere. In our case, each panel is represented by an SMF service instance, with its configuration stored as a set of property groups

and properties of that instance. Additional data files (e.g. classes, images, `.xml` files) can be referenced by the panel definition.

While this configuration could be manually obtained by inspecting a set of SMF services, this is a case where it is cleaner to provide a special purpose MBean. For each panel supported by the system, the system publishes a Panel MBean that can be queried to obtain the panel's description.

It is possible that a custom MBean definition will require reading larger amounts of data than can be efficiently transferred using the MBean transport mechanism. In such cases, the Panel MBean returns URLs that map directly to those static resources. In the case of local access, the URLs will point directly to the files, while in the case of remotely configuring another machine, these URLs will point to Cacao's web server module.

Access Control

As described above, configuration is published by a set of MBeans running in the system's Cacao Common Agent Container. An important part of Visual Panels is managing what access users have to that configuration. There are three parts to this problem: connecting to Cacao, authenticating the user, and determining what that user is permitted to do.

Connection

The primary connection mechanism provided by Cacao is a TLS socket connection, providing both a private communications channel and the ability for clients to authenticate the daemon. The Cacao daemon creates a self-signed certificate at install time. The public certificate is published in the file system so that local connections can verify its authenticity. While this might offer some amount of comfort to remote users as well, a secure connection depends on either propagating this certificate to client hosts or obtaining a proper certificate.

The Java Management Extensions infrastructure is very flexible, and among other things allows consumers to plug in custom connection types. (It is through this mechanism that Cacao provides its TLS connector). To facilitate a more seamless local authentication process (discussed in the next section), Visual Panels will be delivering its own connection type based on Unix Domain Sockets. With an endpoint in `/var/run`, clients can be assured they are talking to the appropriate daemon and that their communications won't be intercepted.

Authentication

Having established a connection with and trust of a Cacao container, the Visual Panels client must authenticate itself. There are two authentication cases that need to be considered.

PAM authentication

The more general authentication case is that of a user attempting to authenticate as an arbitrary user, be it with the local system or a remote host. In this case, Visual Panels uses the PAM authentication mechanism provided by Cacao. The user must supply a username and password. Visual Panels passes these to Cacao, and Cacao passes these to the OpenSolaris PAM authentication framework.

Unfortunately, Cacao's PAM authentication process assumes the default PAM stack configuration. The lack of Java interfaces to the OpenSolaris PAM stack makes a fix for this unlikely in the near term. It should be noted that existing SMC and Lockhart management frameworks have the same limitation (in fact, their implementations are based on the same code).

Total Information Awareness, Unix style

While the above provides adequate authentication, it suffers greatly as far as usability is concerned. In the common case, users will be configuring local machines as themselves. Forcing a user to reauthenticate with Visual Panels when they have already authenticated with the desktop is extremely hostile.

When Visual Panels connects to a local Cacao daemon, it will default to using the Unix Domain Socket connector. In addition to the advantages listed above, connecting in this fashion allows the daemon to determine the credentials of the client through the use of the `getpeerucred(3c)` interface. As a result, local access to Visual Panels requires no additional authentication. This has the added benefit of side-stepping any limitations in Cacao's PAM support (in this case).

Authorization

Authorization is handled entirely by using existing OpenSolaris security mechanisms, primarily RBAC authorizations. As most configuration changes are driven through SCF, Visual Panels MBeans will respect the authorizations specified in the SCF respository. In those cases where the configuration isn't stored in SCF, will likewise be permitted in accordance with the underlying technology.

There is a distinction, though, between individual pieces of configuration and the way in which that configuration is presented to the user. To simplify the user experience, the Visual Panels UI will only permit the user to make changes in a panel if it can determine that the user has the union of all authorizations required by that panel. The ability to make changes at a finer granularity is deferred to a future version of Visual Panels.

It should be noted that as the ability to actually make a change to the system's configuration is evaluated on a case-by-case basis by the MBean publishing the configuration (or the underlying technology itself). A failure of the client to evaluate the user's ability to change the configuration managed by a panel will result in either the user being prevented from making changes they are allowed to make, or being given only the *appearance* of permission to make a change that will ultimately fail. At no time is the user able to make a change contrary to what the underlying configuration allows.

Chapter 5. Panels

A key part of Visual Panels is the inclusion of content in the project's scope. In the initial version, we are planning on include the following services. They are chosen based on their presence in Solaris Marketing's PRD (Product Requirements Document), their exposure of useful OpenSolaris technology, and their usefulness in giving to the user a complete picture of an OpenSolaris system.

Apache web server

Description

Sharing content via the web is an extraordinarily common goal for users. To close the gap between system administration and those who simply have content they want to publish, Visual Panels will deliver a panel allowing configuration of the bundled Apache 2 web server. Unfortunately, providing support for the entire configuration space of Apache would be a project in and of itself (and a never-ending one at that). Visual Panels panel will expose only a commonly used subset of Apache's configuration.

The Apach panel will support virtual servers, and these will be represented as objects in the panel's object view. Each virtual server can have its own document root, exposure of user content (i.e. `http://hostname/~user`), SSL settings, MIME types, and module settings.

There will always be users who fall outside the set of configuration we choose to support. To prevent Visual Panels from becoming an obstruction to these people, we will provide the option of specifying a custom user-provided configuration file that takes the place of all Apache configuration performed through Visual Panels.

Configuration access

Apache's configuration is stored in a text file or set of text files. There are two ways of providing access to this configuration: a special-purpose MBean or by finding a way to represent Apache's configuration in SCF. Any piece of code that we create that is capable of making configuration changes to a large, complicated third-party configuration file will have high maintenance costs. Instead, we will take the latter approach and store in SCF the subset of Apache's configuration we are exposing to the user. When the Apache service is started or refreshed, we will map this configuration — the user's intent — to its Apache configuration file representation.

To accomplish this, Visual Panels will be delivering either modifications to the existing Apache SMF server or a new Apache service. In the case of the former, we will run a separate PSARC fast track to cover the changes. If we deliver a new service, it will be a project private interface of this case.

File sharing

Description

Another common user goal is sharing filesystems to other machines. Visual Panels will be delivering a panel that allows users to configure what filesystems are shared and how. To our benefit, the recent `sharemgr` work made great improvements in how we think of sharing filesystems on OpenSolaris. The Visual Panels file sharing panel will be exposing an identical model. Each object in the objects view will represent a sharemgr share group.

Configuration access

In addition to rationalizing our file sharing design, sharemgr also rationalized how we store our configuration. Most file sharing configuration is now stored in SCF. That said, the representation of that configuration is project private. We are currently evaluating whether it is more appropriate to create a separate MBean that talks to libshare or to request a contract with sharemgr and access its configuration through the SMF MBean.

One benefit the former has is that it insulates us from any changes made to the way sharemgr stores its configuration. As ZFS has added considerable complications to the model, which both makes our job more difficult and the likelihood of change greater, this approach may have more than the usual set of merits.

Internet services

Description

The internet services panel will provide a cross-sectional view into the system's configuration, showing which network-facing services are available, offering the ability to enable/disable or firewall those services, and providing a means of navigating directly to the panel (if available) for each service. Additionally, for those services which are under the control of inetd, inetd-specific configuration options will be configurable.

The set of services shown on this panel is the union of the set of services supported by inetd and the services that have their own panels. As there could be services outside these two sets, the notion of “firewalling” would be qualified as either blocking all access except to the specified services, or allowing all access except to the specified services. While this doesn't cover the entire space of possible configurations, it is noteworthy improvement over the current situation and can be evolved to add richer support over time.

One outstanding question is that of how firewalling is to be achieved. We are currently evaluating the relative benefits of the many network access control mechanisms available on OpenSolaris. Using IPFilter exclusively is appealing because of how general purpose and flexible it is, but handling services (e.g. X11) that can have a variable number of ports open can be tricky. On the other hand, engaging some of the infrastructure put into place by the Secure-by-default work gives us more complete control over individual services, but offers significantly reduced coverage. Ultimately, we may adopt a combination of the two.

Configuration access

All configuration managed by the internet services panel will be accessed through SCF. If we need to effect changes to IPFilter configuration, we will adopt a scheme similar to that used for the Apache configuration.

Core files

Description

A common post-install task for developers as well as deployers is to set aside a global location to capture core files from failed applications. While it is tempting to set up something of this nature by default (much as was done in 2.6 with crash dumps), the fallibility of workloads running on our customers' system isn't under our control. Visual Panels will be delivering a panel that provides easy access to this functionality.

All the coreadm configuration will be made available. As there is only a single set of configuration for the entire system, there will be no objects view for the core file panel.

Configuration access

There is currently a RFE in flight to move coreadm configuration to SCF. Visual Panels will be consuming that configuration directly.

SMF

Description

Finally, Visual Panels will be delivering an SMF panel. While the purpose of Visual Panels isn't to be an SMF UI, SMF plays a large role in the functioning of a OpenSolaris system and therefore deserves representation. Moreover, giving SMF a place in the Visual Panels UI allows us to provide an administrator greater insight into the failings of higher level services, as well as point the administrator in the right direction if they are approaching a configuration problem from too low a level.

The functionality offered by the SMF panel will be a straightforward exposure of the SMF model. Services will be represented as objects. Operations (e.g. enable, disable, clear maintenance) will be applicable to to services, and the application-specific properties of services that have them will be displayed and/or editable as their templates allow. In those cases where there exists a panel that, as part of its implementation, modifies an SMF service, that service will provide (and in fact, encourage) direct navigation to that panel.

Configuration access

Obviously, access to SMF configuration will provided by the SMF MBean.

Chapter 6. APIs

As Visual Panels is structured for reuse, there are many places where APIs will form. It is unlikely that any of these APIs will be published with the initial release, however, as our focus will be on the user experience and not the stability of these interfaces. Instead, these proto-APIs will be considered either project private interfaces or implementation details, depending on how far they have matured. For the purposes of inception, we merely describe what these different areas are.

Panel / Components

One body of APIs are those consumed in the creation of a panel for the Visual Panels UI.

Publishing APIs

We will have a set of APIs defining how a panel is defined and how its various parts are delivered for consumption by Visual Panels.

Data access APIs

Our data access APIs will be those interfaces used by a panel to access the configuration data it exposes. Specifically, they will be a union of the MBean interfaces themselves and any common data management operations that are best implemented on the client side.

UI APIs

We could theoretically require a panel creator to take complete control over a rectangular region of pixels, but doing so would eliminate any possibility inter-panel consistency. We will be providing APIs for common data representation and UI tasks. The primary implementation interface will still be Swing, but a panel will interfaces that help them look like other panels.

Other consumers

A second body of APIs are those which are intended for external consumers of Visual Panels configuration.

MBean APIs

The primary interface we will offer to alternate consumers are the MBean interfaces themselves. This would give Sun and third-party management software the ability to reuse much of the configuration management work done by Visual Panels.

JSCF

More an interface into SMF than Visual Panels, it is likely that we will eventually want to commit the Java SCF interfaces Visual Panels uses internally. We have already seen considerable interest from people who have been experimenting with the prototype published through OpenSolaaris.

Chapter 7. Man pages

Name

vp -- starts the Visual Panels UI

`/usr/bin/vp` [panel]

Description

vp starts the Visual Panels UI.

Operands

The following operand is supported:

`panel`
: `panel`
The name of a panel to open when the Visual Panels UI is started.

Chapter 8. Interface tables

Table 8.1. Exported Interfaces

Interface	Stability
Visual Panels option in Gnome launch menu	Uncommitted
/usr/bin/vp	Committed
`/var/run/vpanels`	Project Private
`svc:/application/management/vpanels:*`	Project Private
org.opensolaris.os MBeans	Project Private

Table 8.2. Imported Interfaces

Interface	Stability
Apache 2 configuration file format	External
sharemgr configuration/libshare	Contract
Cacao Common Agent Container	