

# Caiman Architecture

V1.1

Sarah Jelinek ([Sarah.Jelinek@sun.com](mailto:Sarah.Jelinek@sun.com))

Sanjay Nadkarni ([Sanjay.Nadkarni@sun.com](mailto:Sanjay.Nadkarni@sun.com))

# Table of Contents

1.Introduction.....	3
2.Orchestrator service.....	7
3.Channel Service.....	18
4.Customization Service.....	19
5.Driver Verification Service.....	20
6.Essential Systems Configuration Service.....	22
7.GUI Service.....	23
8.Install Time Update Service (ITU).....	24
9.Jumpstart Profile Service.....	25
10.Logging Service.....	27
11.Metadata Service.....	29
12.Patching Service.....	30
13.Post-Install Service.....	31
14.Software Repository Service.....	32
15.Software Selection Service.....	33
16.Target Detection Service.....	35
17.Target Instantiation Service.....	36
18.Transfer Service.....	37
Appendix A: Caiman requirements mappings.....	39
Appendix B: Glossary.....	42
Appendix C: Revision History.....	43

# 1. Introduction

Over the past few years core Solaris has been significantly enhanced to include world class features, such as Zones, SMF, FMA, ZFS and others. The enhancements for Solaris install have been made primarily in the non-interactive arena with features like Flash. These enhancements typically address the needs of enterprise class users. Functionality for desktop users, who typically use interactive install, has lagged significantly.

To address this problem an OpenSolaris project called Caiman was launched. A survey of the problems that exist with Solaris install can be found at [http://www.opensolaris.org/os/community/install/files/install\\_strategy.pdf](http://www.opensolaris.org/os/community/install/files/install_strategy.pdf). The paper also defines a set of requirements for the new install.

The Caiman architecture came about after an extensive investigation of other installer technologies. The technologies investigated were:

- Sun's Purple Haze engine technology. This is currently a Sun internal project that is being developed in support of the Java Enterprise Software product set. The intent of this engine technology is to be a multi-purpose install engine.
- Solaris installer engine technology. This is not currently open sourced but we are in the process of making this available.
- The Fedora Core 5 Anaconda install engine technology. The data about this is listed in the reference section.

Each one of these engine technologies offered some of the functionality needed to meet Caiman requirements but none could offer the complete functionality required. This necessitated defining the Caiman architecture.

The investigation documents can be found at:

<http://www.opensolaris.org/os/community/install/files/anaconda.pdf>

<http://www.opensolaris.org/os/community/install/files/sinstall.pdf>

The Sun Purple Haze technology is still Sun confidential so the data from that investigation cannot be published at this time.

Typically architectures are expressed in terms of functional blocks with a defined hierarchy. The hierarchical model works well in a top down processing environment. The install architecture does not lend itself to such a model since the flow of information is non-linear, dependent on user input and environment. Thus, the approach taken for the Caiman architecture was to group common sets of functionality in to logical groupings called “services”. These services in concert provide the install experience. The service model allows each grouping of functionality to be started, stopped and resumed on their own.

One important note: The services described in this document are not intended to be the SMF type services that are available on Solaris. Service is used as a generic term to indicate a grouping of functionality and independence of operation.

This document describes the architecture of each of the Caiman installation services. These descriptions include:

- Component requirements
- Component interfaces
- Services that component is dependent on
- Dependent services

Service dependencies may be hard or soft, but have not been noted as such specifically. A hard dependency is one in which the service dependent on another service will not operate without the noted services. A soft dependency is one in which the service dependent on another service will operate but in perhaps a degraded mode if the other service is not available.

Design specific details have been deliberately left out of this document to avoid driving to a particular implementation with one exception:

- "in place" upgrades are not supported.

The primary purpose of "in place" upgrades was to save space. This was important when Solaris install was developed in the early 1990s, when disks were small (around 120MB) and expensive. Today, it's hard to find disks smaller than 30GB. While Solaris distribution has grown over the years, relative to today's disk sizes it occupies a small fraction of the total disk space. Thus multiple Solaris distributions can be stored on a single drive at very little cost. The explosive growth in drive capacity therefore enables one to provide non-destructive type of upgrades, such as Live Upgrade. The code required to support "in-place" upgrade is complex and error-prone due to a large number of edge conditions. Hence the cost benefit analysis does not support providing this capability.

While the document addresses most of the requirements listed in the strategy paper ([http://www.opensolaris.org/os/community/install/files/install\\_strategy.pdf](http://www.opensolaris.org/os/community/install/files/install_strategy.pdf)), the following issues are not addressed:

- *Open sourcing of install and packaging sources.*

There is a separate ongoing effort to open source this code and therefore it is not addressed here.

- *Solaris packaging and patching*

The focus of Caiman is install and upgrade. While packaging and patching are important in delivering Solaris it is considered a separable problem area.

Definitions that will be helpful prior to reading the rest of this document:

Service - A caiman service provides certain useful actions. Every service must support the following functionality

- Start – This enables, invokes the service
- Stop – This disables the service
- Capabilities – Describes the capabilities and provides additional information to avail those capabilities

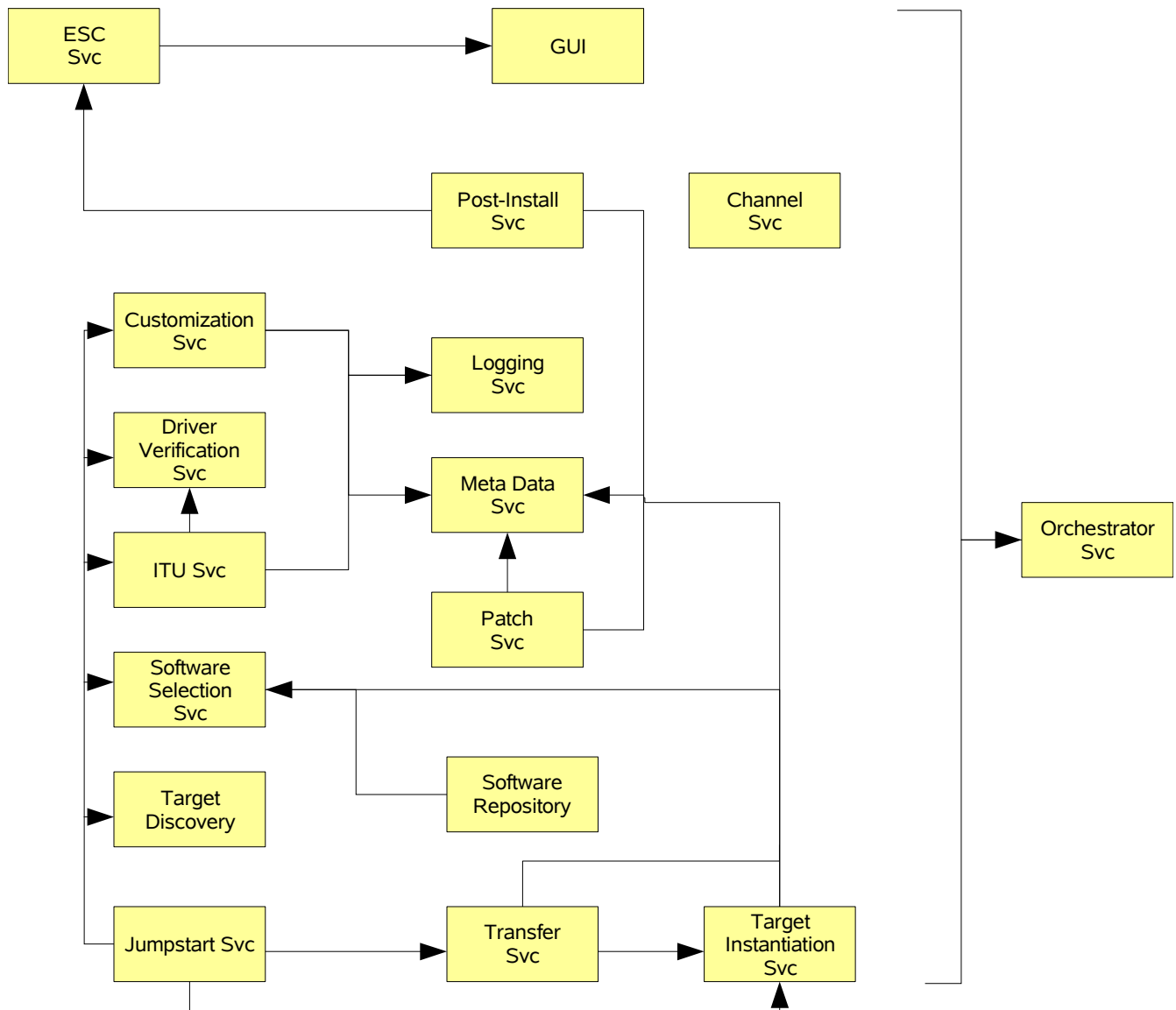
Optionally a service could also support:

- Suspend - Invoking this action suspends the action being performed by a service
- Resume - Resumes the action of a service.
- Abort - This entry point forcefully stops the service

This optional functionality of a service will be listed in the service description when needed.

## 1.1. Caiman Services and their dependencies

As noted above a Caiman service is a logical grouping of functionality and independence of operation. In an effort to clarify how all of this fits together Figure 1.1.1 shows all the Caiman services and their dependent relationships. The dependencies are defined as **A->B** means that service **A is dependent on Service B**. All services are dependent on the Orchestrator service.



*Drawing 1.1.1: Caiman Services Dependencies*

## 2. Orchestrator service

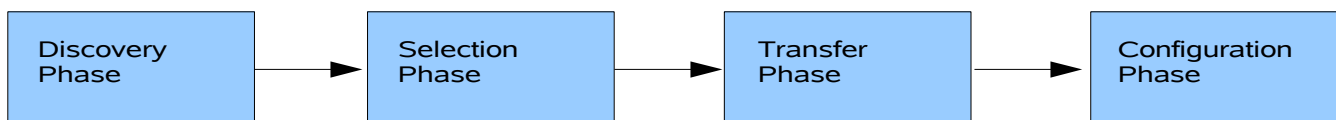
The Orchestrator service is central to the Caiman install operations. It provides central entry points for all installation applications and provides the registration mechanism for all installation services to register their public interfaces. It also provides monitoring services for other installation services to provide status updates and control of service if required. All points lead to and from the Orchestrator service.

Every other service in Caiman is considered subordinate to the Orchestrator service. No install application can directly call, invoke, or in any way affect a subordinate service. The idea is to create an architecture which truly encapsulates functionality and provides the opaqueness desired.

The next sections will outline the basic install process in phases, with an attempt at showing the relevant flows for each phase, and then the full install flow. All of the additional service definitions follow in Section 2. The Orchestrator service requirements, interfaces and dependencies follow these diagrams.

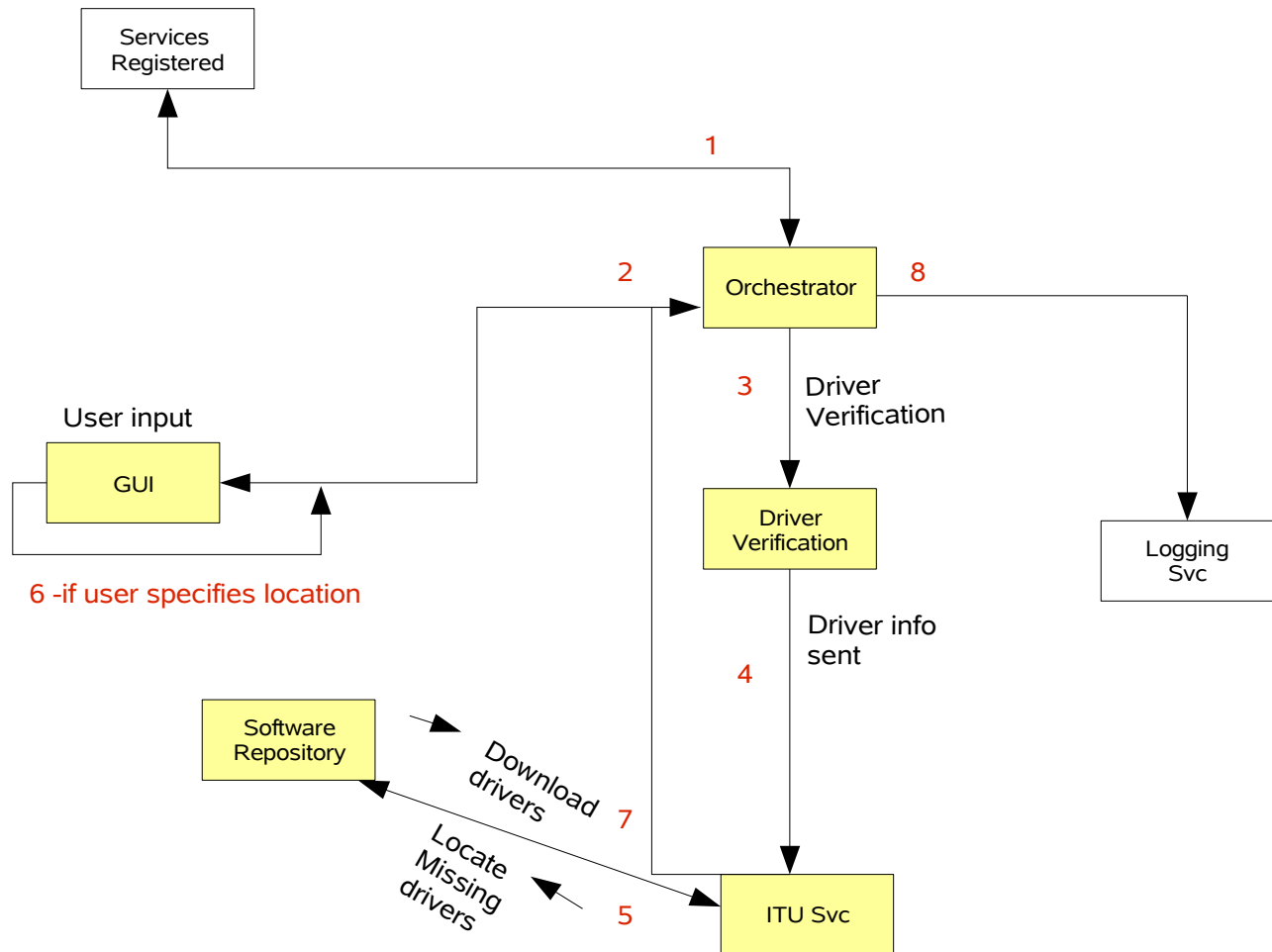
### 2.1. Installation process, Orchestrator Service and flow diagrams

To enable understanding of the Orchestrator service and how all the other subordinate services interact with it during an installation we provide a textual description of a basic installation process. We also provide several Caiman installation flow diagrams to aide in the understanding of how this all works together. There are four phases that are identified as part of a Solaris installation process.



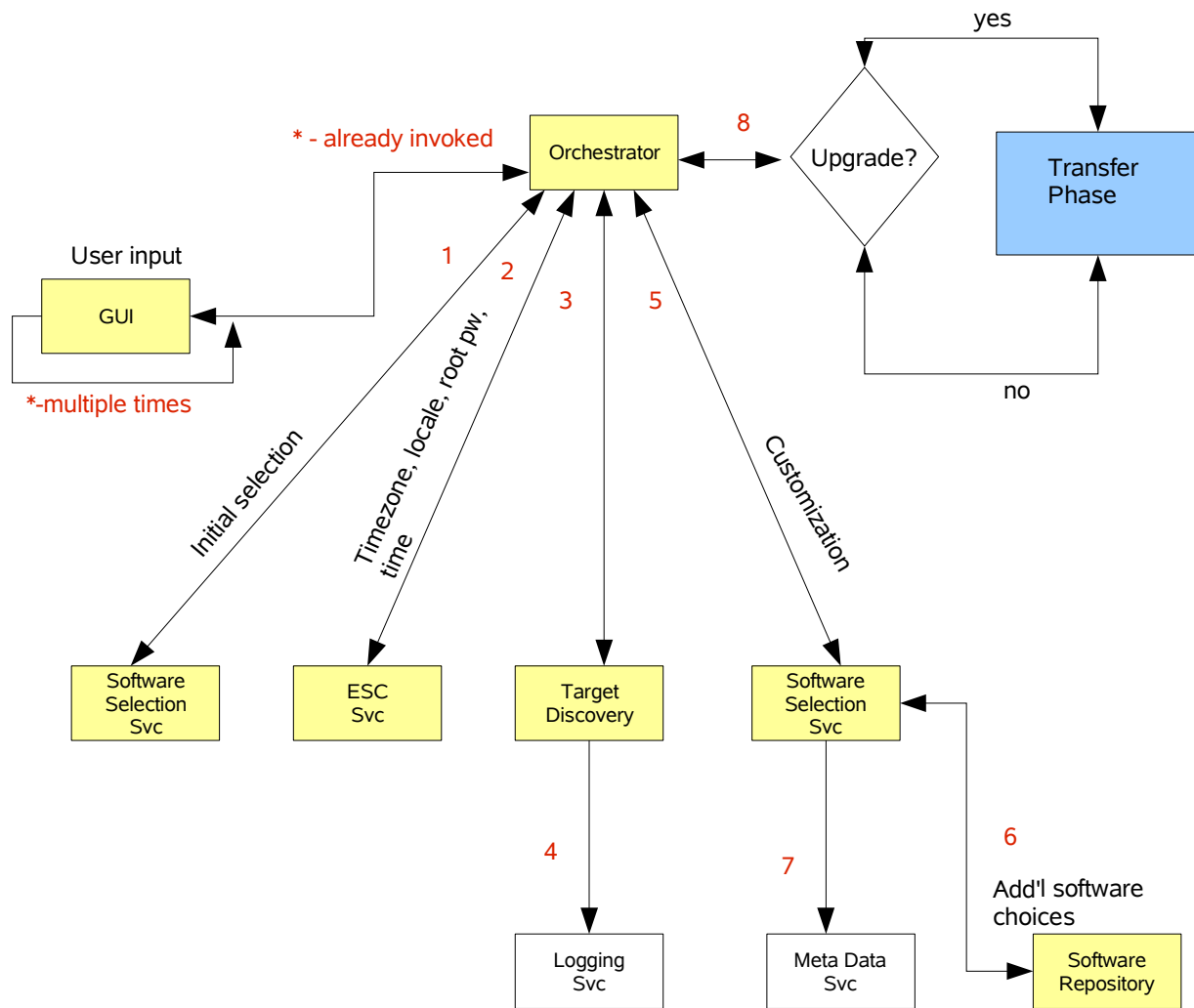
*Drawing 2.1.1: Phases Of Install*

The process of installing a system interactively begins by clicking the *Solaris Installer* icon on the desktop. It invokes the *Orchestrator service(1)* which starts the subordinate service registration process, invokes the *GUI service(2)*, the *Logging service(8)* to capture the output and provide information for debug. The *Orchestrator service* also begins probing the system by starting the *Driver verification service(3)*. The *Driver verification service* matches the devices on the system with the drivers available on the media. It gathers a list of inaccessible devices and calls the *Software repository service* to search for the missing device drivers. The *Software repository service* searches a list of predefined local and if possible, remote repositories. On completion of this service, devices are matched with the appropriate drivers. The location for the drivers are also recorded. The user is informed about supported and unsupported devices via the GUI. If drivers are required, the Orchestrator invokes the *ITU service(7)*. This uses the information from the *Driver Verification service(4)* and gets the drivers(5,6) and loads them on the running system thereby enabling the devices.



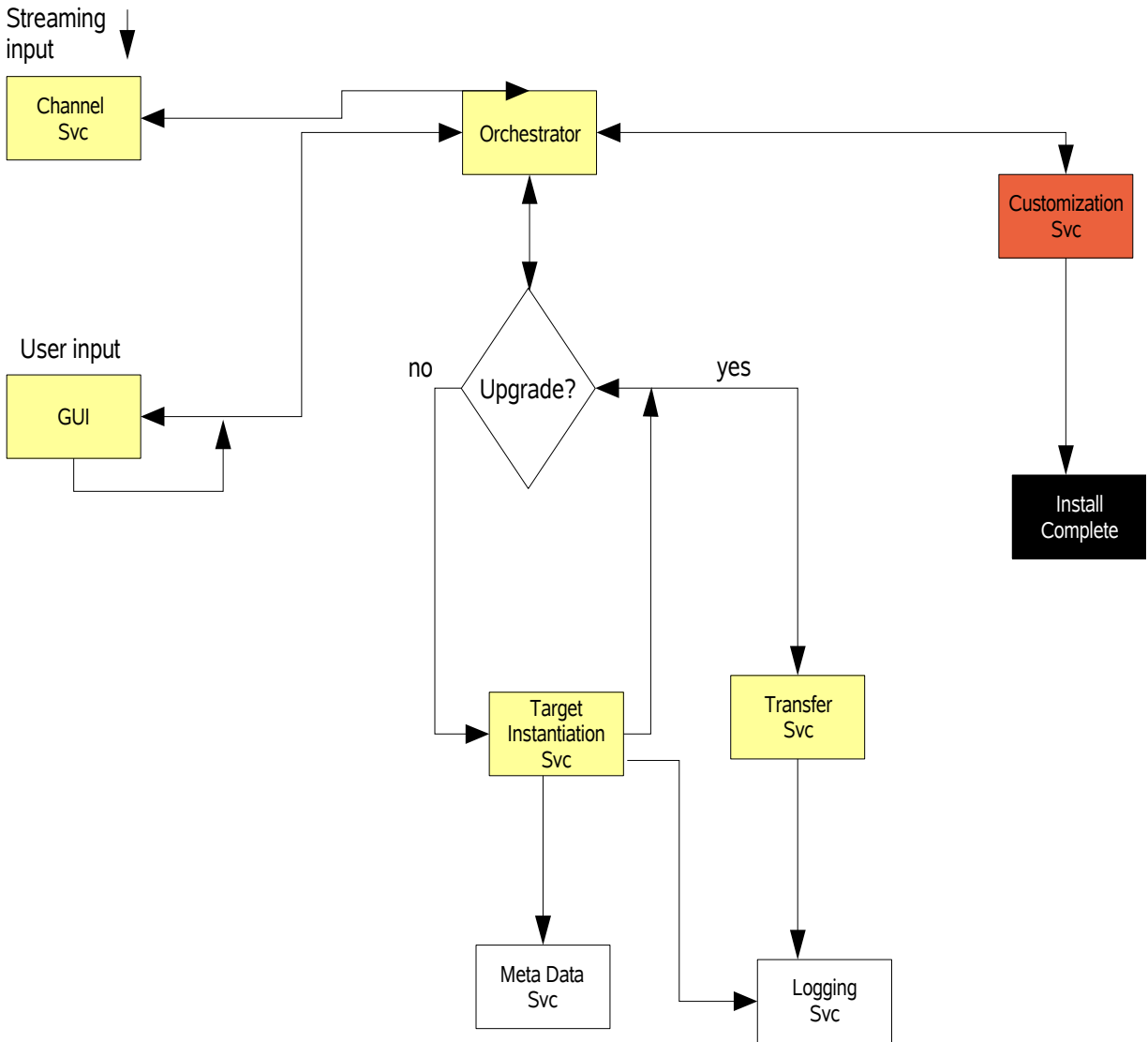
***Drawing 2.1.2: Discovery Phase***

With all the possible devices accessible, the focus of install shifts to capture the intent of the user and information about the software required. The orchestrator invokes the GUI which presents a screen to the user to accomplish both these tasks. The user can choose to either install, upgrade among others. The process of selecting software is driven by the usage of the system rather than software bundles. This approach greatly simplifies the choices presented to the user without assuming a Solaris centric bias. Thus software selection is based on system usage, i.e., desktop, server or server and desktop. The intent of the user and the software selection drives the rest of the install process. The orchestrator starts the **Software selection service(1,2)** to verify the consistency of the software selection. If the intent of the user is to perform a fresh install, **Essential System Configuration (ESC) service(2)** is called to set system settings such as time zone and locale. Both, Install and upgrade cause the **Target discovery service(3,4)** to be started, which gathers information about the possible devices onto which Solaris can be installed or updated. This device information list is presented to the user via the GUI. Once the user decides to proceed, install process moves to the next phase.



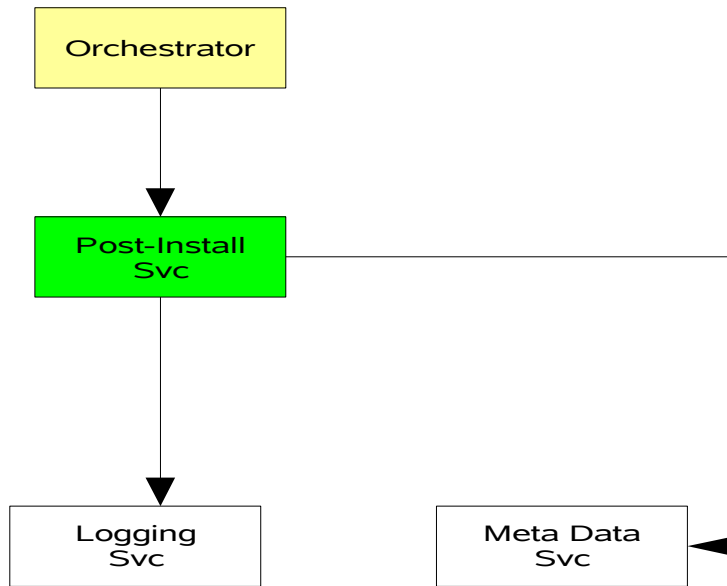
**Drawing 2.1.3: Software Selection Phase**

The orchestrator starts the *Target instantiation service* to configure the chosen device appropriately for a fresh install. The orchestrator then invokes the *Transfer service* with the information gathered by the previous service to populate the target device. Since this process can take sometime, the GUI is provided with a “progress report”. The Orchestrator also starts the *Channel service*, to provide information to the user.



***Drawing 2.1.4: Transfer Phase***

Once this transfer is complete, the user has the option to customize the setup using the ***Customization service***. In most cases, the system would reboot. On boot up, the ***post-install service*** would be invoked by the orchestrator.

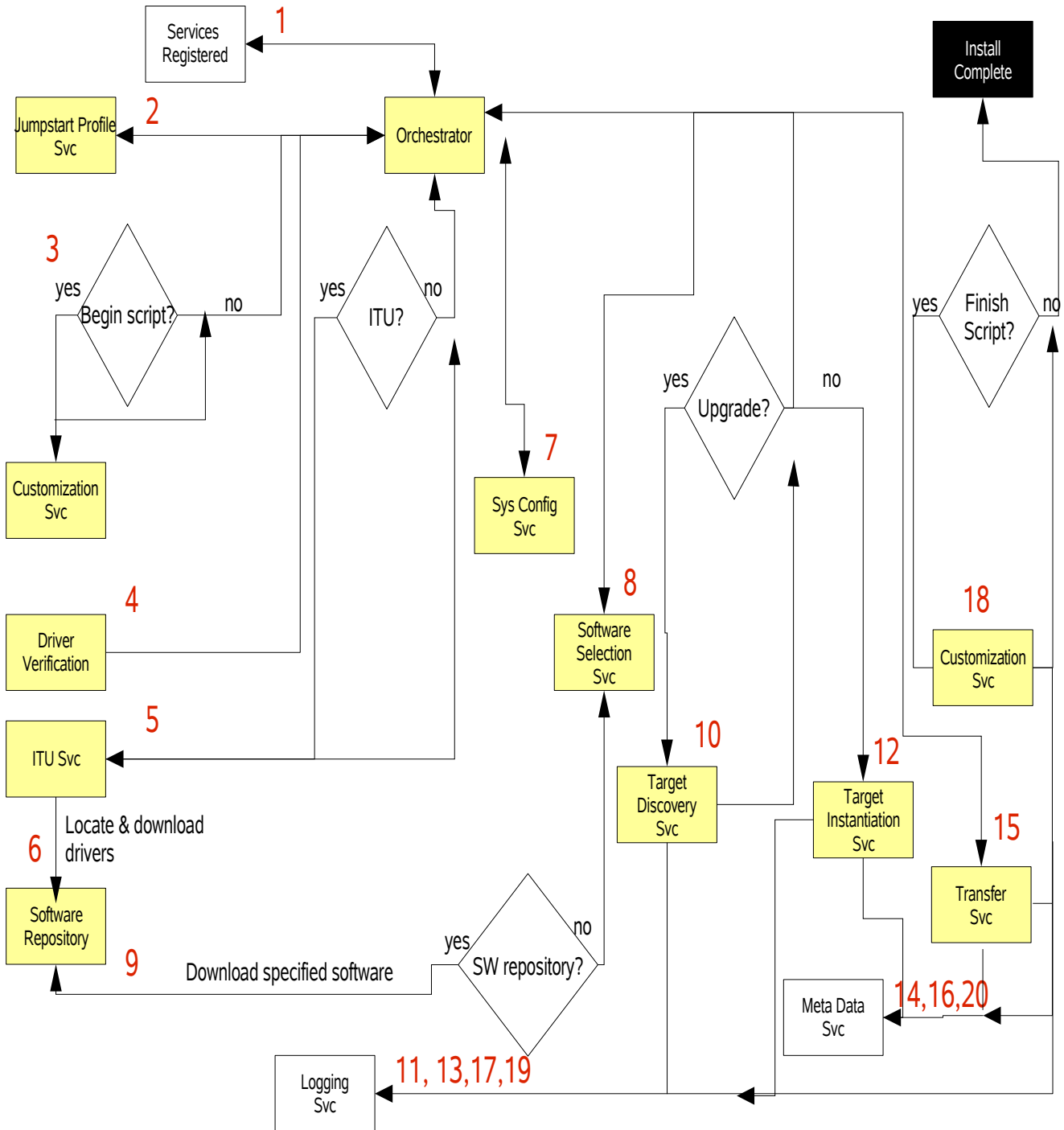


***Drawing 2.1.5: Post Install Phase***

The diagram on the next page shows the full sequencing of the Caiman services during an interactive install. It brings together all of the component diagrams shown above. The diagram is read starting at the top with “Services Registered” and moving through the diagram in numbered order.



For completeness we provide a diagram which shows the flow of control for the Caiman services during a Jumpstart installation. As with the above diagram, the flow is read starting at the top “Services Registered” and moving through the diagram in numbered order.



**Drawing 2.1.7: Jumpstart Installation Flow**

## 2.2. Purpose:

To provide central entry points for all installation applications. To provide the registration mechanism for all installation services to register their public interfaces. Provide monitoring services for other installation services to provide status updates and control of service if required.

## 2.3. Requirements:

1. Must run in interactive installation environment.
2. Must run in hands off installation environment.
3. Must be self re-startable in the event of a failure.
4. Must not allow failures in subordinate services to result in failure in this service.
5. Must provide API to support all installation applications.
6. Will provide registration mechanism for subordinate installation services to register their public interfaces.
7. Will provide subordinate service monitoring.
8. Will provide subordinate service scheduling.
9. Must be able to start and stop services.
10. Will be able to suspend, resume and abort services that are identified with such capabilities.
11. Must be extensible to provide the ability to add new installation services entry points.
12. Will provide UI with all status reporting as gathered from subordinate services.

## 2.4. Interfaces:

- Interface for subordinate services to register their public interfaces that can be used by other services
- Public API for installation applications to use for initiating an installation procedure.
- Interface for services to register with the monitoring feature of the orchestrator service
  - Will provide interface to enable registering service to submit data regarding which process/thread should be monitored.
  - Will provide interface to enable registering service to submit actions to take on monitored process/thread
- List of supported actions.
- Interface to provide list of public interfaces available from the subordinate services.
- Interface for subordinate services to register callback actions with Orchestrator.

## **2.5. Services Dependent On:**

- None

## **2.6. Dependent Services:**

- All others

## **3. Channel Service**

### **3.1. Purpose:**

During an interactive installation, provide the ability to provide of streaming data. These could be canned messages on the media or remote repositories.

### **3.2. Requirements:**

1. Channel service will be invoked only during an interactive install
2. Invocation of this service must not impact the ongoing activity.
3. In case of low memory, the Orchestrator must be able to disable this service.
4. The user must be able to disable this service.

### **3.3. Interfaces:**

- An interface to provide streaming data
- Interface to suspend and resume this service.

### **3.4. Services Dependent On:**

- Orchestrator service

### **3.5. Dependent Services:**

- None

## **4. Customization Service**

### **4.1. Purpose:**

This service allows the user to customize his/her environment after the install/upgrade is complete. This service will be called prior to the reboot after installation task has been completed.

### **4.2. Requirements:**

1. Must run in interactive installation environment.
2. Must run in hands off installation environment.
3. Must log the location of the script with the metadata service
4. Will provide the ability to invoke a script/program that lives in a well defined user specified location.
5. The service will provide no error detection or handling of the user script/program. It will however accept a return value at the end of the execution of the customization script.
6. Will provide logging and debugging facility.

### **4.3. Interfaces:**

- Interface to abort service.
- Interface to read and process script.

### **4.4. Services Dependent On:**

- Metadata service
- Logging service
- Orchestrator service

### **4.5. Dependent Services:**

- Jumpstart

## 5. Driver Verification Service

### 5.1. Purpose:

Provide a high level of confidence that the key devices (keyboard,mouse,display,network, disk and USB) are usable when Solaris is installed on the system.

### 5.2. Requirements:

1. Must be able to discover devices on the system.
2. Must be able to match devices to drivers.
3. Verification service must be reasonably fast.
4. Options must exist to scope device discovery for large systems.  
E.g. - devices required for install
  - scan storage devices
  - scan all devices
5. Must be able to query repositories to check for latest versions of the drivers. Options to query the repositories must be at the behest of the user.
6. Must be able to provide information on missing device drivers.
7. The service must provide enough useful information about the device to the user.  
This might entail doing more analysis than just providing the vendor id. For PCI devices using bus bridge and motherboard information might be useful. For USB devices usb product name should be added.
8. Provide data in a defined format for Install Time Update to download and install the appropriate drivers.
9. Interactive and non-interactive options/interfaces must exist.
10. Must run in LiveDVD, jumpstart as well as on an installed system.
11. Options must exist to redirect the output from this service.

### 5.3. Interfaces:

- Interface for consumers to query for device data that will include:
  - vendor id - usb devices
  - device id - pci devices
  - Hardware information
- Interface to provide Abort functionality required

#### **5.4. Services Dependent On:**

- Orchestrator service

#### **5.5. Dependent Services:**

- ITU service

#### **5.6. Additional Notes:**

Though not directly related to Driver Verification Service, an infrastructure to is required to make this service successful. As a start,

- Driver/software repositories need to be defined.
- A mechanism to register drivers with OpenSolaris project is required.
- Driver validation service would be useful.
- For security purposes, the ITUs and the drivers need to be signed.
- Information about the quality and background of the driver would be useful. For example: is the driver provided by a vendor or a member of the community. What are the known issues, etc.

## **6. Essential Systems Configuration Service**

### **6.1. Purpose:**

Essential Systems Configuration(ESC) service provides a formal interface to the basic system settings required during install and upgrade. Some of these settings need to be set as part of the boot process, such as keyboard,time and time zone. Other parameters, such as root password are set during install.

### **6.2. Requirements:**

1. Must run in a hands-off environment
2. Must run in a graphical environment
3. Must run in a text-only environment
4. Must be able to provide a list and value of settings
5. Must provide a mechanism to add new settings
6. Must be able to set some or all of the settings
7. Must provide information on errors logged.

### **6.3. Interfaces:**

- Interface to provide a list of managed settings
- Interface to set and update the managed settings
- Interface to add new settings

### **6.4. Services Depended on:**

- Partially on the Orchestrator service
- system boot up

### **6.5. Dependent Services:**

- GUI Service

## **7. GUI Service**

### **7.1. Purpose:**

Provide the interactive user installation front. Will take input from user and from Orchestrator service to formulate the installation request. Will determine system configuration data automatically if possible.

### **7.2. Requirements:**

1. Must be easy to use.
2. Must be intuitive.
3. Will provide user input selections for all installation capabilities.
4. Will utilize orchestrator service API for installation tasks

### **7.3. Interfaces:**

- User interface as specified by Xdesign organization.

### **7.4. Services Dependent on:**

- Essential System Configuration Service
- Target Discovery Service
- Software Selection Service

## 8. Install Time Update Service (ITU)

### 8.1. Purpose:

Provide an out of band mechanism for delivering critical drivers required to boot and install a system.

An ITU consists of a binary driver, configuration file for that driver and the SVR4 pkg for that driver. Thus the driver could be pkgadd'ed on to the system as part of regular install.

### 8.2. Requirements:

1. Tools to create ITU must be easy to use.
2. Behavior must be identical on sparc and x86/x64.
3. Must be able to get ITUs from a wide range of media including repositories on the internet.
4. Must be able to process the the information generated by the driver verification service to determine the ITUs to download
5. If the ITU is being pulled from an external repository, user confirmation is required during interactive installation mode.
6. In interactive mode, the ITU service must provide information about the ITUs available on a given media.
7. Jumpstart must be extended to enable ITU usage. In this mode no user confirmation will be enabled, even if ITU is pulled from external repository.

### 8.3. Interfaces:

- Interface to specify location of ITU repository
- Interface to list well known ITU repositories
- Interface to provide information about ITU's available
- Interface to abort this service must be available

### 8.4. Service Dependent On:

- Orchestrator service
- Driver Verification service
- Software repository service
- Logging service

### 8.5. Dependent Services:

- None

## 9. Jumpstart Profile Service

### 9.1. Purpose:

To deploy 'hands-off' (currently known as jumpstart profile) installations.

### 9.2. Requirements:

1. Will provide automated detection of jumpstart installation request.
2. Will provide profile testing capability.
3. Will provide completely hands off installation, upgrade and updates to systems.
4. Will support full image installations, advanced deployment installations, and live upgrades.
5. Will support creation of non-global zones and creation of Xen DomU's.
6. Will support upgrade of Solaris zones.
7. Must support remote software updates.
8. Must allow for flexibility of software groupings defined for installation/upgrade.
9. Will provide support to upgrade from a partial repository.
10. Will provide support for software add/update only(without an installation type of keyword).
11. Will provide support for creation of ZFS root pools.
12. Will provide support for ITU.
13. Will provide partial installation functionality support.\*

\*This requirement is in reference to providing the ability to do software updates/refreshes without invoking a full install or upgrade. Jumpstart will be modified to do only a software update if the user requests.

### 9.3. Assumptions:

- Caiman customization service will handle the begin and finish scripts defined by the user.
- Assumes that the necessary locating and setup to enable the jumpstart install has been determined and completed.

### 9.4. Interfaces:

- Interface that allows for testing and validation of profile.
- Interface that reads and processes 'profile' request.

### 9.5. Services Dependent On:

- Orchestrator service
- Customization service

- Driver Verification service
- ITU service
- Software Selection service
- Target discovery service
- Target instantiation service
- Patching service
- Metadata service
- Software repository service
- Logging service
- Transfer service

## **9.6. Dependent Services:**

- None

## 10. Logging Service

### 10.1. Purpose:

Provide a logging mechanism to log data pertaining to success or failure of installations.

### 10.2. Requirements:

1. Must run in interactive installation environments.
2. Must run in hands off installation environments.
3. Must provide multiple 'levels' of logging capability that can be set by the user.
4. Will provide debugging options for use by developers and testers.
5. Will allow descriptive logging messages.
6. Must provide logs in well known location by default, and will be public for use by all installation applications.
7. Will provide ability to override logfile location.
8. Will provide ability to locate current log file.
9. Will provide an extensible error code format to enable rich messaging capabilities for installation applications.
10. Will provide an extensible success code format to enable rich messaging capabilities for installation applications.
11. Will provide logs in multiple formats, text, html.
12. Will provide mechanism for sending logs to interested parties.
13. Will Provide the capability for forwarding messages to another host on the network.
14. Should be able to proceed in parallel while other processes are still pending.
15. Will provide a user specified configuration file mechanism that can be used to override system settings.

### 10.3. Non-Requirements:

1. Localization/internationalization capabilities for messages  
All localization is handled by the calling program

### 10.4. Interfaces:

- Interface for specifying log file(s) location
  - Will keep track of current location specified, default or user specified.
- Interface for writing log message
  - Will check for configuration file data, if present

- Will check for appropriate permissions
  - Will not overwrite existing log file
  - Will preserve most recent log file if it exists
  - Will check log 'level' to do appropriate logging
  - Will send log messages via email to specified recipients.
  - Will send log messages to syslogd if requested.
  - Will send log messages to remote host if requested.
- Interface for specifying format of message strings
  - Standard Message type definitions must be defined
  - Logging service configuration file syntax

### **10.5. Services Dependent On:**

- none

### **10.6. Dependent Services:**

- Orchestrator service
- Customization service
- Transfer service
- Jumpstart profile service
- Postinstall service
- ITU service
- Target instantiation service

## 11. Metadata Service

### 11.1. Purpose:

This service provides the mechanism for storing system metadata in a repository and generating jumpstart profiles, and appropriate CSN format for use in Software Update manager.

### 11.2. Requirements:

1. Must run in interactive installation environment.
2. Must run in hands off installation environment.
3. Will be performant so as not to add an unnecessary burden to installation time.
4. Will provide the required metadata so that a profile representing the current installed system can be generated.
5. Will provide the required metadata for the CSN Software Update service.
6. Will be extensible enough to provide for extensions in the metadata format.

For example, if jumpstart is modified to add support for zone creation/update, then this service should be extensible to support addition of generation of appropriate metadata to provide the profile support.

### 11.3. Interfaces:

- An interface to 'generate' the metadata requested depending on type.
- An interface to write the metadata to the repository.
- An interface to retrieve data from the repository and return it in jumpstart profile format.
- An interface to retrieve the data from the repository and return it in CSN required format.
- Published format of metadata in repository.
- An interface to repopulate the repository.

### 11.4. Services Dependent On:

- Orchestrator service

### 11.5. Dependent Services:

- Post install service
- Customization service
- Patch service
- ITU service
- Transfer service

## 12. Patching Service

### 12.1. Purpose:

To provide installation of patches for Solaris during the installation/upgrade process.

### 12.2. Requirements:

Must run in jumpstart installation environment.

1. May be supported in the interactive installation environment.
2. Will provide patch installation capabilities
3. Will install requested patches correctly.
4. Will provide installations specified via NFS, http, ftp, or locally.
5. Will update metadata repositories as part of patch installation process.
6. Will provide patch dependency checking.
7. Will support rollback to pre-patch system.

### 12.3. Interfaces:

- Interface to install patches listed with specified patch add options.
- Interface must provide access methods to different media from which patches can be loaded.
- Interface to simulate patch installations for testing
  - Validate dependencies
  - Validate space requirements
- Interface to provide list of patches currently installed on system.

### 12.4. Services Dependent On:

- Metadata service
- Orchestrator service

### 12.5. Dependent Services:

- Jumpstart service

### 12.6. Additional Notes:

- A goal of this service is to be software patch agnostic

## 13. Post-Install Service

### 13.1. Purpose:

Post-install service is run on the first reboot after an install. It allows for further system configuration and customization. Most of system configuration tasks are deferred during install and the post-install service is the first opportunity to perform these tasks using the standard system utilities along with user customized tools.

### 13.2. Requirements:

1. Will provide default post-install actions.
2. Must provide a user extensible mechanism to add post-install tasks.
3. Must provide concise and clear information about the tasks to the user.
4. Must log the output of the tasks.
5. Ability to provide a debug the sequence must exist.
6. Must provide a method to bypass post-install.

### 13.3. Interfaces:

- Interface to provide information about tasks to the system.
- Interface to extend the list of tasks.

### 13.4. Services Dependent on:

- Orchestrator
- Logging service
- metadata service

### 13.5. Dependent Services:

- None

## **14. Software Repository Service**

### **14.1. Purpose:**

To provide the ability to locate and download software from software repositories that are not part of the current installable distribution.

### **14.2. Requirements:**

1. Will run in interactive installation environment.
2. Will run in hands off installation environment.
3. Will know about well known Solaris software repository locations.
4. Will validate repositories are online prior to displaying lists of repositories.
5. Will allow for user addition of software repository locations.
6. Must understand format of repository data to effectively provide software listings.
7. In hands-off installation , specification of software to download from remote repository must require no user interaction.
8. Will provide download mechanism for software.

### **14.3. Interfaces:**

- Interface to provide a list of well known software repositories that can be used for installing Solaris software.
- Interface that sets additional software repositories specified by the user.
- Interface that lists software available for download from specific repository site.
- Interface that lists the software selected by the user from the remote repository.
- Interface to validate repository specified is 'online'.
- Interface that downloads selected software.
- Interface to suspend, resume and abort the service.

### **14.4. Services Dependent On:**

- Orchestrator service
- Software selection service – for dependency checking

### **14.5. Dependent Services:**

- ITU service

## 15. Software Selection Service

### 15.1. Purpose:

To provide the ability to select, customize and check dependency of software during install and upgrade.

### 15.2. Requirements:

1. Will allow for "software clusters" to be selected or deselected only, not package level components for interactive installer.
2. Will allow for package level component selection with hands off install.
3. Must run in interactive environment.
4. Must run in hands off environment.
5. Will provide final selected list of software in the appropriate formats based on consumer needs.
6. Will perform appropriate dependency checking.
7. Will mark software with appropriate action to be taken. i.e. remove from cluster, remove old pkgs to upgrade to new, remove all instances of package, etc...
8. Will enforce the requirements to install core packages and clusters.
9. Will provide appropriate architecture compatibility checking.
10. In the event of a failed dependency check the software selection service will output failure data but allow installation to continue.

### 15.3. Interfaces:

- Interface to load software from installation media.
- Interface to load software currently installed on system.
- Interface to provide the list of software that will be installed on system, after all customization has been completed. (that has been selected by user for installation)
- Interface to provide lists of software clusters available from installation media
- Interface to provide lists of selectable software
- Interface to provide lists of required software
- Interface to report dependency status of software selections
- Interface to provide selection/de-selection mechanism for software clusters
- Interface to set the transfer service action for each software package.

### 15.4. Services Dependent On:

- Orchestrator service

## 15.5. Dependent Services:

- Software Selection service
- Transfer service

## 16. Target Detection Service

### 16.1. Purpose:

Target detection service entails discovering and gathering information about targets onto which Solaris can be either installed or upgraded.

### 16.2. Requirements:

1. Must discover physical and virtual targets that can be installed and upgraded. Examples of physical targets are disks, USB media. Virtual targets are virtual machines like DOMs (Xen), Zones.
2. Must be able to understand FDISK partitions, including the Extended Partitions on x86/x64 systems.
3. Must be able to discover if file systems exist on the targets and provide information on those file systems. The file systems include UFS, ZFS, FAT16, FAT32, NTFS as well as a few popular Linux file systems.
4. If a target discovered is a Solaris root device, Solaris version and metacluster installed is required.
5. Must run in hands off installation environment.
6. Must run in interactive installation environment.

### 16.3. Interfaces:

- Interfaces must exist to provide the discovered target information. The information at minimum must include target description, size, file system information, Solaris version (if applicable).
- Interface to abort the service must exist.

### 16.4. Services Dependent On:

- Orchestrator service

### 16.5. Dependent Services:

- None

## 17. Target Instantiation Service

### 17.1. Purpose:

This service primarily gets the target ready for installation.

Targets for install could vary greatly. In case of disks, they may need to be formatted or resized. In case of x86 they may need to have the correct fdisk partition. For VMs, such as zones or Xen DOMU, mappings to define disk space, memory etc. may be required.

### 17.2. Requirements:

1. Needs user provided size and layout information information
2. For ISAx86/x64, interface to a re-partitioning library/utility is required.
3. Will not support non-destructive re-sizing of target
4. Will set up user described disk or file system mapping for Xen and Zones
5. Must work in a DOMU environment (?)
6. Must run in a hands off environment.
7. Must run in an interactive environment.

### Interfaces:

- Provide an interface to abort the service.
- Interface to provide target description .
- Interface to instantiate requested targets.
  - Will do space checking

### 17.3. Services Dependent On:

- Metadata service
- Orchestrator service

### 17.4. Dependent Services:

- Transfer Service

## 18. Transfer Service

### 18.1. Purpose:

To take the user's software selections and get them installed on the system to the specified target devices.

### 18.2. Requirements:

1. Must install the requested bits correctly.
2. Must run in hands off installation environment.
3. Must run in an interactive installation environment.
4. Must handle virtual as well as physical target environments.
5. Must have all install targets setup and ready.
6. Will handle management of modified files that must be placed on disk or in the repository.
7. Will be extensible enough to handle advanced deployment formats as they developed.
8. Will do validation checking of installation environment.
9. Will provide media check.

### 18.3. Non Requirements:

1. Does not have to be re-startable in the event of a failure.\*  
\* Why? Since we support only initial install or live upgrade we can just start over in the event that the installation or upgrade of software fails. Failure here means system shuts down unexpectedly, or something like that.
2. Does not have to provide in place upgrade functionality.

### 18.4. Interfaces:

- An interface to install/upgrade software selected by the user to the appropriate defined targets.
  - Will support flash archives and other advanced deployment formats.
- Provide an interface to abort service.

### 18.5. Services Dependent On:

- Orchestrator service
- Software Selection service
- Target instantiation service
- Metadata service

## **18.6. Dependent Services:**

- Post install services – this is due to the fact that post install won't happen if the transfer service doesn't complete successfully.

## **18.7. Additional Notes:**

- A goal of this service is to be software packaging/format agnostic.

## Appendix A: Caiman requirements mappings

The table below shows the original requirements specified in the install strategy document noted in the Introduction. We have mapped each of these requirements to the architectural elements that will provide for the requirement.

<i>Caiman Requirements</i>	<i>Proposed Approach</i>
1 The user interface must present a modern graphical look and feel, consistent with Sun's branding and consistent with other Sun software products.	Proposed GUI* *Will be pointer to current GUI mockup when available
2 The user interface must support both graphical and text environments.	All services except GUI support this.
3 The user interface must be simplified to present only those questions absolutely necessary to correctly install the software. Non-essential questions should be deferred until the system is installed, or, preferably, eliminated entirely.	GUI
4 The user interface must not require additional user input once the user has selected the software to be installed and the location to which it will be installed and initiated the actual installation process.	GUI
5 The user interface must use clear, non-technical terminology whenever possible.	GUI
6 The user interaction must be similar across both graphical and text environments.	
7 The user interface must support both initial installation and upgrades from prior releases of Solaris	
8 Initial installation and upgrade must be supported in both off-line and live forms	Jumpstart service
9 Initial installation and upgrade must be supported from manufactured media such as DVD's, or images of same.	
10 Initial installation and upgrade must be supported over both local-area and wide-area networks, including through firewalls.	WAN Boot technology
11 Initial installation and upgrade must support authentication of both clients and servers in network transactions, and encrypted communications between clients and servers.	WAN Boot technology
12 Initial installation and upgrade must support integrity checking of the software to be installed and authentication of package creators.	

## ***Caiman Requirements***

## ***Proposed Approach***

- |    |   |   |
|----|---|---|
| 13 | Initial installation should offer to verify support of the system's hardware before attempting to install the software  | Driver Verification Service                                       |
| 14 | Installation must offer the option to integrate non-bundled drivers into the system for use during the installation process, and install those same drivers into the finished system as part of the installation process. | Install Time Update Service                                       |
| 15 | Installation must support coexistence between Solaris and other operating systems on the same system.   | Target Discovery Service  |
| 16 | Installation must not require the use of non-Solaris tools to create space for Solaris on the system's disks when other popular operating systems are also installed.   | Target instantiation service                                      |
| 17 | Configuration interfaces used during the installation process should be similar to the interfaces used after installation for similar tasks..   | Visual Panels   |
| 18 | Initial installation and upgrade must offer the ability to execute without the requirement of any interactive input during the process.   | Support for hands off installation in all services                |
| 19 | Installation tools must offer the customer the ability to create customized installation media.   | Tools based on install metadata                                   |
| 20 | At the completion of installation, the user should have an initial, non-root account available for login.   | GUI   |
| 21 | Equivalent installation capabilities must be offered across all supported hardware platforms.   | No architecture dependencies                                      |
| 22 | Configuring services for network installation on a single server must be one integrated, self-contained user task.  | Not addressed -future install server setup tool                   |
| 23 | Initial installation and upgrade of virtual systems such as Zones and Xen virtual machines must be supported, and must provide similar automatic installation capabilities as for physical systems.                       | Jumpstart, Target Discovery Service, Target Instantiation Service |
| 24 | Initial installation must allocate system resources such as disk space in such a way that customers will be able to later upgrade with either the off-line or live method.  | No "in-place" upgrades, Target Instantiation Service              |
| 25 | Upgrade should provide the ability to revert the system to its pre-upgrade state. A failed upgrade must do so automatically unless otherwise instructed by the user.  | Live Upgrade, rollback capabilities                               |
| 26 | Users must be able to easily install additional operating system software after initial operating system installation, with any software dependencies automatically resolved and installed.                               | GUI and Software selection and Transfer service                   |
| 27 | Initial installation should offer to verify support of the system's hardware before attempting to install the software  | Driver verification, ITU service                                  |
| 28 | Users must be able to create and manage repositories of installable software packages, with those repositories discoverable and usable by the tools used to install software packages.                                    | Software repository service                                       |

### ***Caiman Requirements***

### ***Proposed Approach***

- |    |   |   |
|----|---|---|
| 29 | Users must be able to easily upgrade defined subsets of the system software (for example, the GNOME release), without requiring a complete system upgrade   | Software refresh via interactive and jumpstart installs |
| 30 | Upgrades must be sufficiently fast that the time required when using modern LAN technology will allow daily upgrades by developers without significant impact on their productivity.                                      | Addressed as performance requirements                   |
| 31 | Configuration parameters required to perform an installation should be collected automatically, using the network if possible. The user must be prompted only when necessary, or when specifically requested by the user. | Metadata service  |
| 33 | The installation and packaging technologies used by Solaris must be released into and developed under the OpenSolaris program.  | In progress, not part of Caiman specifically            |
| 34 | Preinstalled systems must not require a reboot between initial power-on and user login.   | Not part of Caiman specifically                         |

## **Appendix B: Glossary**

### **Flash archive:**

A flash archive is an easily transportable version of a reference configuration of the Solaris operating environment plus optional other software. These archives are used for rapid installation of Solaris on a large number of machines.

### **Jumpstart:**

The jumpstart installation method is a command-line interface that enables you to automatically install or upgrade systems based on profiles that are created.

## Appendix C: Revision History

<i>Revision</i>	<i>Date</i>	<i>Changes</i>
1.0	11/1/2006	Initial Draft
1.1	11/7/2006	<ul style="list-style-type: none"><li>● Section numbers added.</li><li>● Added flow diagrams and descriptive text to Section 1.1, and Sections 2.1.</li><li>● Incorporated initial review comments</li></ul>