

# From OSCAR3 to OSCAR4

## *Fundamental Changes*

Jeremy Enos, Jason Brechin, Terry Fleury, and Neil Gorsuch

## Abstract

We discuss the origins of OSCAR (Open Source Cluster Applications Resources) and the evolution to the current state of the program, version 3.0. We then propose several major architecture changes to the program in order to address many of the limitations of the current version. The new OSCAR 4 will include modifications such as: a new GUI installer based on Perl-Qt to provide a wizard-driven installation and an overall consistent look-and-feel; universal use of ODA (the OSCAR DATABASE) for storage and retrieval of all OSCAR-related data; extensions to SIS to allow automatic detection of disk types; automatic package/file dependency calculations; arbitrary grouping of nodes to allow heterogeneity of software installation/configuration; enhancements in node discovery and assignment; and the introduction of NEST (Node Event and Synchronization Tool) which enables nodes to update themselves based on differences between their current configuration and a target configuration. We also mention other development efforts parallel to the current redesign which may be integrated at a future date.

## 1. Introduction

OSCAR (Open Source Cluster Application Resources) is a tool designed to lower entry barriers into Linux cluster computing. It provides a wizard-driven GUI interface for cluster installation and software component configuration. A single supported Linux host is required to install and run the OSCAR wizard. After that the rest of the cluster is built over the network from the “OSCAR Server” host. The end result is a standard Beowulf cluster installed with software and configurations which match *best known practices* agreed upon by diverse and experienced contributors.

The framework of OSCAR essentially serves as a software installation API for Linux clusters. Linux clusters vary in size, proportions, configurations, etc. However, there are some common characteristics and requirements of Linux clusters upon which OSCAR can build. OSCAR also offers flexibility not typically seen in Linux clusters. For example, most Linux clusters require compilation, computation, administration, and access point functionality. The software that OSCAR installs should not need to know the number of compute nodes or the list of hostnames. Rather, it can just know that cluster compute nodes get *configuration type A*, and cluster access points get *configuration type B*. OSCAR, using information entered into the wizard, fills the information gap (i.e. the user’s custom cluster layout) and takes action to install and configure the cluster. The result is a custom built cluster with a tested and recommended configuration, without requiring the cluster expertise usually required when performing the task manually.

### 1.1. Past

In September 2000, the Open Cluster Group was formed. It consisted of the OSCAR group and other groups parallel to it. An OSCAR Steering Committee was also formed, along with a set of bylaws which mandated a fair balance in the steering committee between industry and academia. Various contributing organizations were operating in a range everywhere from a background skunkworks project to an officially funded task with full time employees. The original goal of OSCAR was to provide an open source clustering solution which worked on standard commodity components via a wizard-driven graphical interface. The program of what is now known as OSCAR was written in September 2000. Each member organization assembled their volunteered components and version 1.0 was released April 2001. Over the next three years, 12 more official releases were cut (not counting third party re-rolls) on the way to version 3.0. These releases carried new distribution support, updated software components, bug fixes, and new features that began to expand the realm of OSCAR past a cluster installation utility

into a maintenance tool. This was accomplished in large part by the capability to add/remove nodes and software packages to/from a running cluster.

## 1.2. Present

OSCAR has made great strides towards its original goal of simplified cluster creation. However, OSCAR is a work in progress. Linux clusters have become much larger and they demand sophisticated methods for scaling. Typical large scale machines call for a more heterogeneous software distribution than smaller machines, and thus require an installation tool to reach further into the maintenance realm than OSCAR currently does. Plans for OSCAR call for enhancements to address such issues.

## 1.3. Future

In the long term, OSCAR will maintain the perpetual goals of expanding support to new distributions and architectures, *oscarizing* more software packages, and adding new features. There is demand for Itanium, x86-64, and even PowerPC architecture support, which correlates somewhat to the demand for Debian, SUSE, and OS X support. At the time of this writing, a utility to abstract package types (rpm, deb, etc.) has been created and integration is underway. However, expanding support to include new distributions involves many more challenges than simply abstract package handling. There are several other changes to OSCAR that must come first.

Short term goals, i.e. the planned 2004 accomplishments, are the primary focus of this document. There are some major additions and rewrites to the core infrastructure design of OSCAR 4. The phrase “taking one step backwards in order to take three steps forward” certainly applies to some of the changes. In order not to impose a fixed limit on OSCAR’s future scalability, some foundation level changes are required. These changes will affect the package API somewhat, so packages will need to be upgraded at that time. This is a cost better incurred sooner, rather than later, when more third party packages are available. Hence there is a high priority on this change relative to extending support to other distributions. Because these changes add more interface options, the GUI requires modification as well. For more than a year now, the Perl-Tk GUI panels have been replaced by Perl-Qt panels any time panels were modified. Following that pattern, the scope of the changes occurring in OSCAR 4 will introduce a GUI fully converted to Perl-Qt. Because the low level design changes and GUI modifications are intertwined and codependent, they can’t be spread across releases very easily and thus will show up mostly in version 3.1. On the positive side, as new features appear some current problems will disappear as an inherent side effect of such heavy changes. The negative side effect is the longer release interval than previously required (around six months). However, given the nature of the type of product OSCAR is, a user installs it as infrequently as they install an OS (since OSCAR is a superset of an OS installation). So by that comparison six months isn’t long at all. Expect it to be worth the wait.

## 2. OSCAR 4 Design-Specific Changes

### 2.1. Exclusive Use of ODA

#### 2.1.1. What is ODA?

Previous versions of OSCAR relied on “flat files” for storage/retrieval of information. In the next version of OSCAR, data used by the system will be stored/retrieved using ODA. ODA stands for “OSCAR DAtabase”. However, there is much more to ODA than just a database. ODA contains all data collected by the user interface, as well as automatically detected data. Information is stored per node regarding hardware and software configuration. For example, ODA stores per-node information pertaining to configuration of the node, software compiled on the node, and software installed to the node after the initial setup phase.

MySQL is the backend software for ODA, but MySQL commands aren’t required to access ODA. Most of the data can be read/written from/to ODA by entering *shortcut commands* on the command line. See Figure 1 for some example *shortcuts*.

```
[root@posic /]# oda list_shortcuts
add_dns_server_to_node
add_node_to_cluster_partition
add_node_to_node_group
...

[root@posic /]# oda create_node_group pvfs_iods
[root@posic /]# oda add_node_to_node_group posic001.ncsa.uiuc.edu pvfs_iods
[root@posic /]# oda add_node_to_node_group posic002.ncsa.uiuc.edu pvfs_iods
[root@posic /]# oda add_node_to_node_group posic003.ncsa.uiuc.edu pvfs_iods
[root@posic /]# oda add_node_to_node_group posic004.ncsa.uiuc.edu pvfs_iods
[root@posic /]# oda nodes_in_node_group pvfs_iods
posic001.ncsa.uiuc.edu
posic002.ncsa.uiuc.edu
posic003.ncsa.uiuc.edu
posic004.ncsa.uiuc.edu
[root@posic /]#
```

Figure 1.

#### 2.1.2. Centralized Data Storage

Previous versions of OSCAR used multiple locations for storing configuration and hardware data. Hardware data was stored within the SIS database, and software configuration data was stored in cache files at each OSCAR package location. As the SIS API was subject to changes between versions, and there was no clear model for interaction between OSCAR packages, that model was clearly insufficient for a dynamic system. The next version of OSCAR will be the first in which *all* cluster configuration data is centrally stored and universally accessible. (See Figure 2.) The user interface will use ODA as an abstract layer for accessing the tools receiving data input.

#### 2.1.3. Network Accessibility

ODA is accessible over the network. In this manner, nodes individually identify how they should be configured, removing the burden on a centralized server to push configuration to the cluster in broad sweeps. This is discussed later in the second on “NEST”.

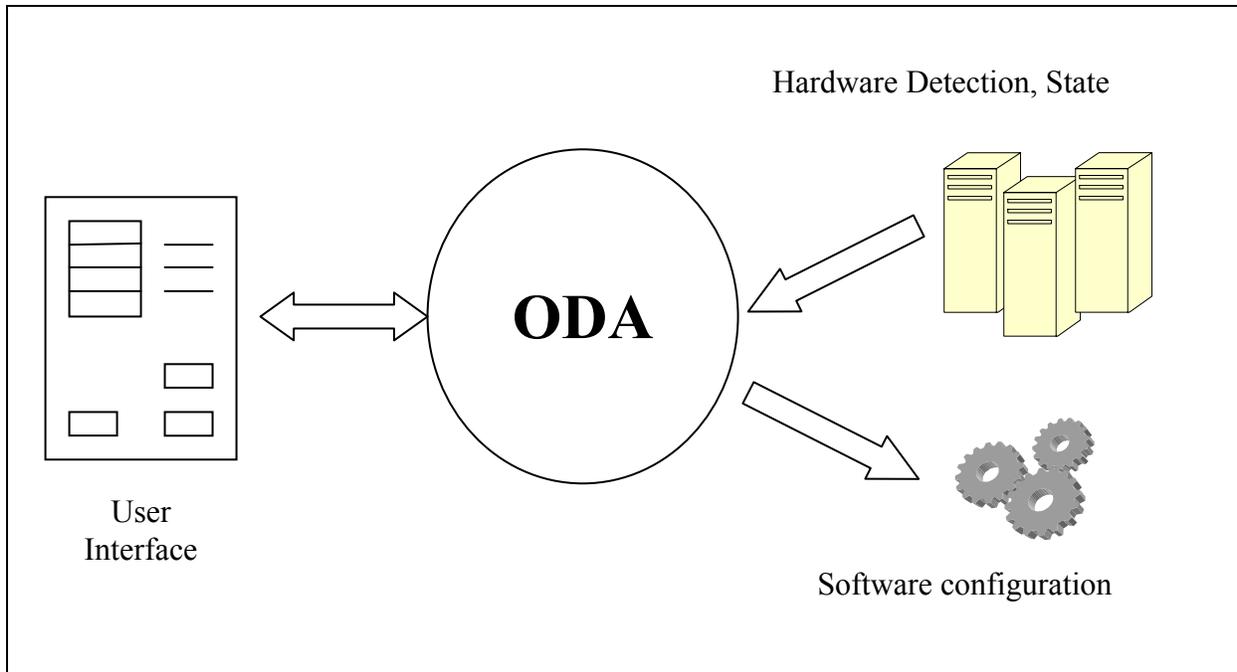


Figure 2.

## 2.2. GUI (Graphical User Interface)

### 2.2.1. Perl-QT GUI Toolkit

The OSCAR GUI started as a pure Perl-Tk implementation, and has been slowly evolving to use Perl-Qt. Benefits of Perl-Qt include a more polished look, better design tools, and greater functionality. OSCAR software packages had their configuration options written in HTML format for the previous wizard. Because Perl-Qt also has an HTML rendering capability, this transition will be seamless for packages. In fact, more complex options can be offered by packages once the Qt Configurator is available.

### 2.2.2. Installation “Wizard” Look and Feel

All OSCAR versions to date have had a similar look to the main wizard window. This consisted of a set of steps numbered in the order they should occur in any cluster install. While this has worked well for us in the past, many steps are becoming re-runnable as OSCAR expands into the cluster maintenance realm. Having numbered steps in which only certain steps are re-runnable can be confusing to the end user. This is just one of many reasons for a change. The next release of OSCAR will not use any Perl-Tk panels, nor will it have numbered steps. Instead, it will offer a `<Back|Next>` interface for the mandatory installation steps, now called *Tasks*. This is what many users think of when they envision a “wizard”. The optional customization steps, called *Tools*, will be available via a pull-down menu system. Various panels may be extensible to show advanced configuration options.

### 2.2.3. Workspace Model

Another sharp difference from the past is the move to a *workspace* model where a single (parent) window contains all the (child) wizard panels. The main application may also detect whether or not it is run for installation or for maintenance of a cluster, and present the user with an appropriate starting point. An API for the workspace which describes how a developer can integrate a Task or Tool into the new GUI has been established and is constantly expanding to encompass new developer requirements. It is important that each GUI component utilizes the workspace API as this API allows for communication between the various Tasks and Tools, a feature which was sorely missed in previous versions of OSCAR. For example, one component can refresh its display when another

component updates ODA. This type of communication layer, along with the centralized data storage provided by ODA, serves to keep displayed information consistent.

## 2.3. SIS

### 2.3.1. Disk Type Auto-Detect

The System Installation Suite, or SIS, is currently the component in OSCAR which performs the initial node installation over the network. In order to install a node, an *image* must be created first on the server. The image is basically a snapshot of what the node's filesystem will look like after installation, minus any node-specific changes. The image is then 'rsync'ed to the node for installation, and can be used later by some users for maintenance purposes. This process is known as Image Based Management, and is discussed in detail later in the section on NEST. In the past, SIS images have been created to work with a single disk type such as SCSI or IDE. Because of this, a single image cannot be used to install two systems with differing disk types. For users that have nodes with heterogeneous disk types, this results in complication and extra burden, especially for those users who use Image Based Management for maintenance. There are OSCAR-design related reasons why this is a burden as well, so there is currently development underway for a patch which would make SIS capable of auto-detecting the disk type, even in systems which contain multiple disk types. Specification of the disk type is currently the only input required from the user to create an image. Once this feature is complete, the image build will happen automatically without user interaction, further simplifying the installation process.

### 2.3.2. Image Management Front End

Though the image build will occur without user interaction, a user may still have custom needs and wish to add or delete images. SIS provides command line tools such as *mksiimage* to accomplish this, but the next version of OSCAR will provide a GUI front-end to such tools. The GUI will also have extra functionality such as node assignment options. There may also be a front-end for tools such as *getimage*. (The *getimage* tool is a component that synchronizes images to live nodes.)

## 2.4. Software Package Handling Improvements

### 2.4.1. Abstraction

Distribution support has always been limited to those distributions that are RPM based. RPM based distributions were and still are a solid baseline for OSCAR due to the overwhelming popularity with consumers and third party software. However, some unrelated factors have begun to change this bias. For example, when the availability of a free, RPM-based Itanium distribution narrowed, demand increased for some distributions which do not use RPM. A rewrite of OSCAR's OS package handling mechanism is needed for other reasons as well, but abstraction away from the OS package type is the main motivation.

### 2.4.2. Automatic Dependency Calculation

The largest reason for a rewrite of package handling is the automation of software dependency installation. This makes an OSCAR package author's job easier because RPM dependencies per distribution are computed on-the-fly. In OSCAR packages to date, an author needed to specify all dependencies for each supported distribution within their package. Removal of OSCAR packages is also simplified because the package author won't be forced to list all dependent RPMS within the package. The removal process will only attempt to remove the pertinent RPMS, not the dependencies. This is safer because another installed package might share the same dependencies, in which case the dependencies should not be removed.

## 2.5. Arbitrary Node Grouping

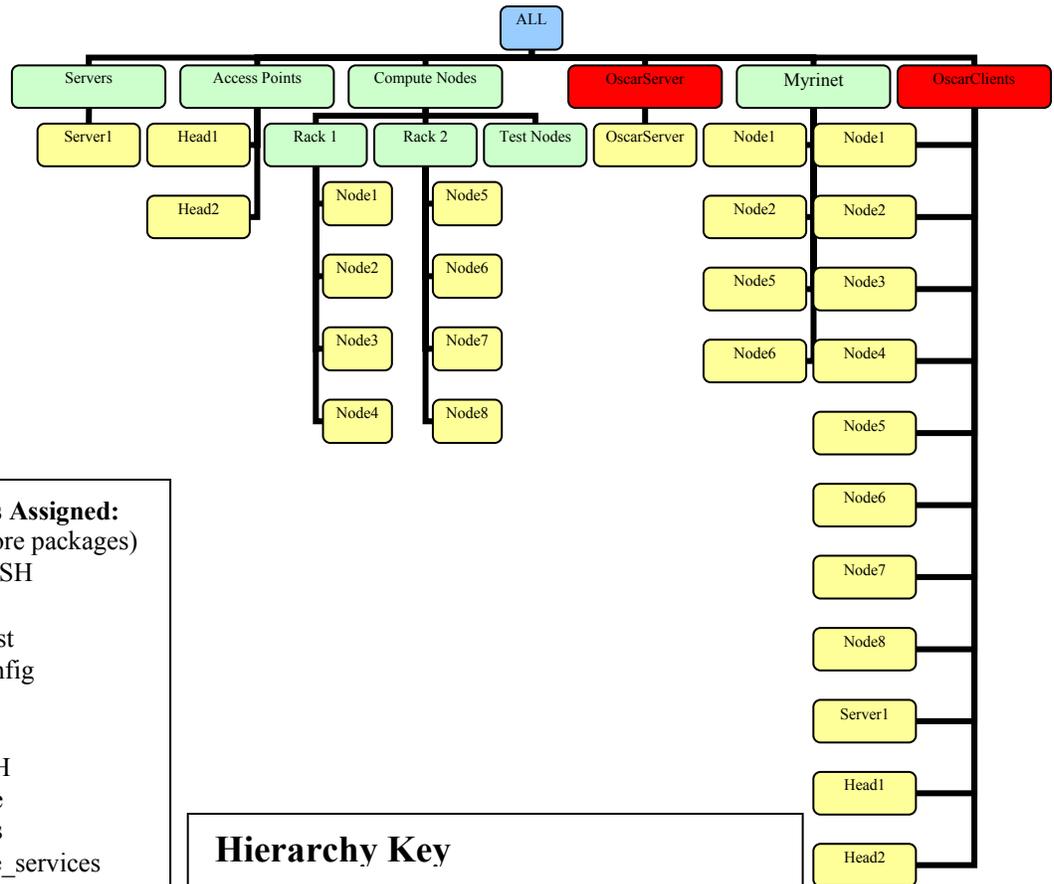
### 2.5.1. Heterogeneity within the Cluster

Support for node grouping, i.e. the assignment of an alias to a set of nodes which can be used in place of those nodes where applicable, is a long standing feature request in OSCAR. The ability to group groups, i.e. the assignment of an alias to a set of node groups, is also an important feature. For the purposes of this discussion, let “node grouping” imply “node and group grouping”. The ability to use group aliases for nodes is useful for several reasons, particularly when there is a need to treat one group of nodes differently from another. For example, system administrators may wish to assign groups based on physical node location, node property, node usage, or differing software configurations. OSCAR packages may also utilize groups for their own purposes, such as keeping track of client-server relationships. The first use of groups will likely be by the wizard itself. The new GUI will provide options to use group aliases to assign and manipulate software configurations.

### 2.5.2. Arbitrary versus Exclusive Grouping

Design discussions regarding groups made it clear that there were definitely needs for arbitrary- AND exclusive-style groups. Arbitrary groups are needed for a variety of uses, such as node properties where groups should allow overlap. Exclusive groups are needed for uses such as configuration assignment, where multiple configurations might conflict with each other. Initial design discussions suggested that we needed to have two separate node grouping concepts: one for exclusive groups, one for arbitrary groups. However, dividing this functionality would have drawbacks. It would be confusing to the system administrator and also difficult to implement. Further analysis of the expected usage of groups yielded an alternative.

Both types of grouping are certainly needed, but the exclusive type of use (for saved configuration) is more predictable, meaning that an arbitrary model would work if the exclusive type of use provides its own safeguards to prevent overlaps and conflicts. The primary use for the exclusive type grouping is in the OSCAR wizard itself, when assigning saved configurations. Because this is something we control, a safeguard is possible. The wizard must check for conflicts when assigning configurations to groups, or when modifying group membership. At that point, we have the best of both worlds.



- Configurations Assigned:**
- ALL (at least core packages)
    - OpenSSH
    - C3
    - Loghost
    - Ntpconfig
    - Pfilter
    - LAM
    - MPICH
    - Torque
  - Compute Nodes
    - disable\_services
  - Access Points
    - Development
    - Pfilter (config override)
  - Servers
    - *not* LAM
    - *not* MPICH
    - Maui
  - Myrinet
    - GM
  - Node6
    - Package\_foo

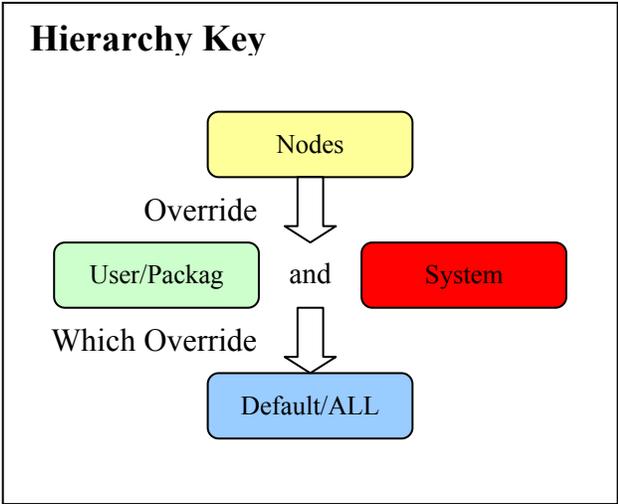


Figure 3.

### 2.5.3. Hierarchical Grouping

After running through some example usage scenarios, it was determined that there may be another improvement we could make. Take Figure 3, for example. Here, a user has several package configurations created and bound to the “ALL” group, but would like to test a new firewall configuration on just a few nodes. If the wizard is protecting against configuration conflicts between groups, then the following steps can be taken:

1. The configurations for the ALL group are copied (moved) to a newly created subgroup which wouldn't conflict.
2. Another new group is created for the test nodes at some level which wouldn't conflict with the first group to take on all the “normal” nodes.
3. Other tedious group manipulation may need to be done in order to maintain the original state of all the nodes which were affected by the changes in configuration assignment which they were previously getting from the ALL group.

A very simple need could indeed turn into a very complicated manipulation of group membership and configuration assignments, which also increases the chance of an error resulting in an unintended cluster reconfiguration.

The answer to this mess is a hierarchical relationship between groups. This way, the ALL group can still retain all the configurations originally bound to it without conflicts, because any configuration binding which is closer to the node itself is dominant. This allows much more flexibility because a group can be created at any point, as a member of any other group, with any nodes or groups as members. Programmatically, all entities in Figure 3 are groups. However, there are three special kinds of groups for the system which cannot be modified by the user:

1. ALL
  - serves as a default
  - has no parent
  - can be overridden by any child
  - contains all managed entities
2. Individual nodes
  - appear to the user to be nodes, but are actually groups (for implementation simplicity)
  - are the only groups to actually contain nodes (where all other groups contain only groups)
  - can have only a single node, with the group name the same as the node name (e.g. node06 is the sole member of group “node06”)
  - can accept configuration assignments individually, just like any other group
3. System Created
  - oscar\_server group - contains server
  - oscar\_clients group - contains all nodes managed by server

Figure 3 illustrates the hierarchical relationship. The flexibility advantages are outstanding. Checks for conflicts must still be made because conflicts could occur between two groups at the same level. However, with the hierarchical system, conflicts won't occur nearly as often, and are easy to avoid. With this model, homogeneous configurations are as easy to manage as they once were, but without sacrificing the ability to have a complex heterogeneous cluster. Such complex requirements occur more frequently as cluster sizes increase.

## 2.6. Node Management

### 2.6.1. Consolidation of Functions

In versions of OSCAR up to and including version 3.0, the system administrator has been required to follow different paths through the wizard for creation of nodes during an installation, for creation of nodes after an installation, and for deletion of nodes. The new Node Management GUI (Figure 4) will encompass all of those paths and provide a single place where nodes can be discovered (via MAC collection), created, or removed. This consolidation will replace four of the 12 buttons on the main GUI window from version 3.0. This is quite an improvement. The new Node Management GUI also heralds the introduction of a new automation feature.

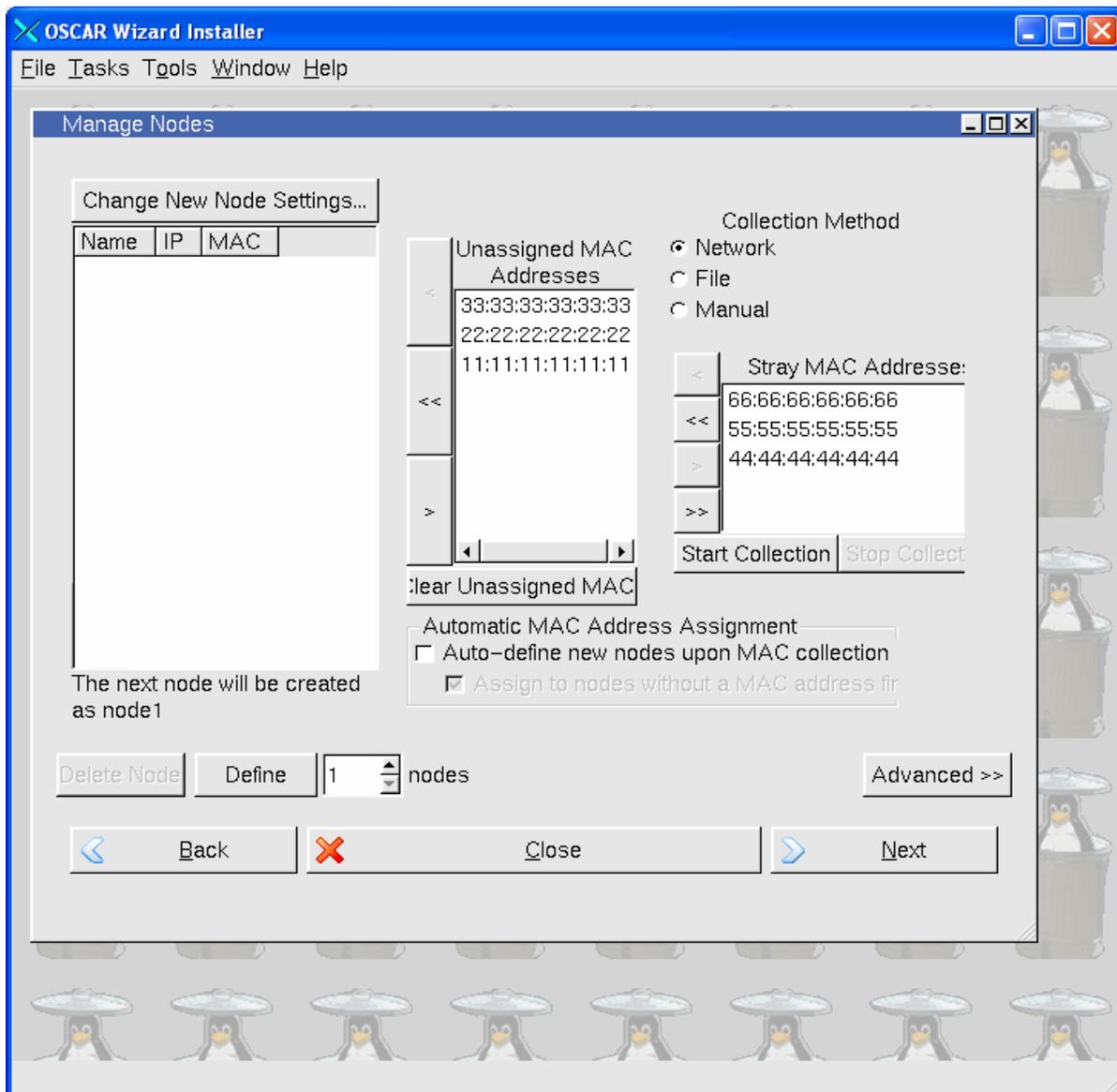


Figure 4.

## 2.6.2. Auto Detect-Define-Build Node Feature

OSCAR users have clamored for a one-step node discovery and build process. All versions to date have essentially required the user to first define a node allocation on the server, and then reboot the node twice, once for the MAC address discovery for identification, and again for the actual build process. Clearly, it would be more convenient to simply power on a node and have it automatically become the next member of the cluster. The ability to automatically detect, define, and then build a node requires fundamental changes to the way in which these tasks have previously been done, but it is certainly possible. This automatic process will be a supported build method in the next release. For users more comfortable with the current process, the original manual mode will still be available.

## 2.6.3. Streamlined MAC Collection

Another change not overt to users deals with the MAC collection process. It will be possible to enter MAC addresses manually (new), import from a file, or discover them from the network. Contrary to the past, any collection method will send the MAC addresses along the same code path. This will ease future feature additions such as MAC filtering.

# 2.7. *Configurator*

## 2.7.1. Master Installation Control

In previous OSCAR versions which had a “Configurator” (i.e. a widget allowing for the setting of configuration options specific to a package), users were shown a list of packages which could be configured as specified by the package author. Basically, that hasn’t changed, but the new Configurator will be a much more powerful tool than the previous one. Upon entering the Configurator stage, a user will see a list of all packages and associated information. (See Figure 5.) In this window, a checked checkbox indicates that the package will be installed on the cluster based on its configuration options. If the checkbox is not checked, then the package is not installed on the cluster at all. This allows for an easy way for a package to be removed from the cluster without having to configure the package’s node assignments. Each package’s configuration button will bring up a Configurator for that package, which is where most of the new features reside.

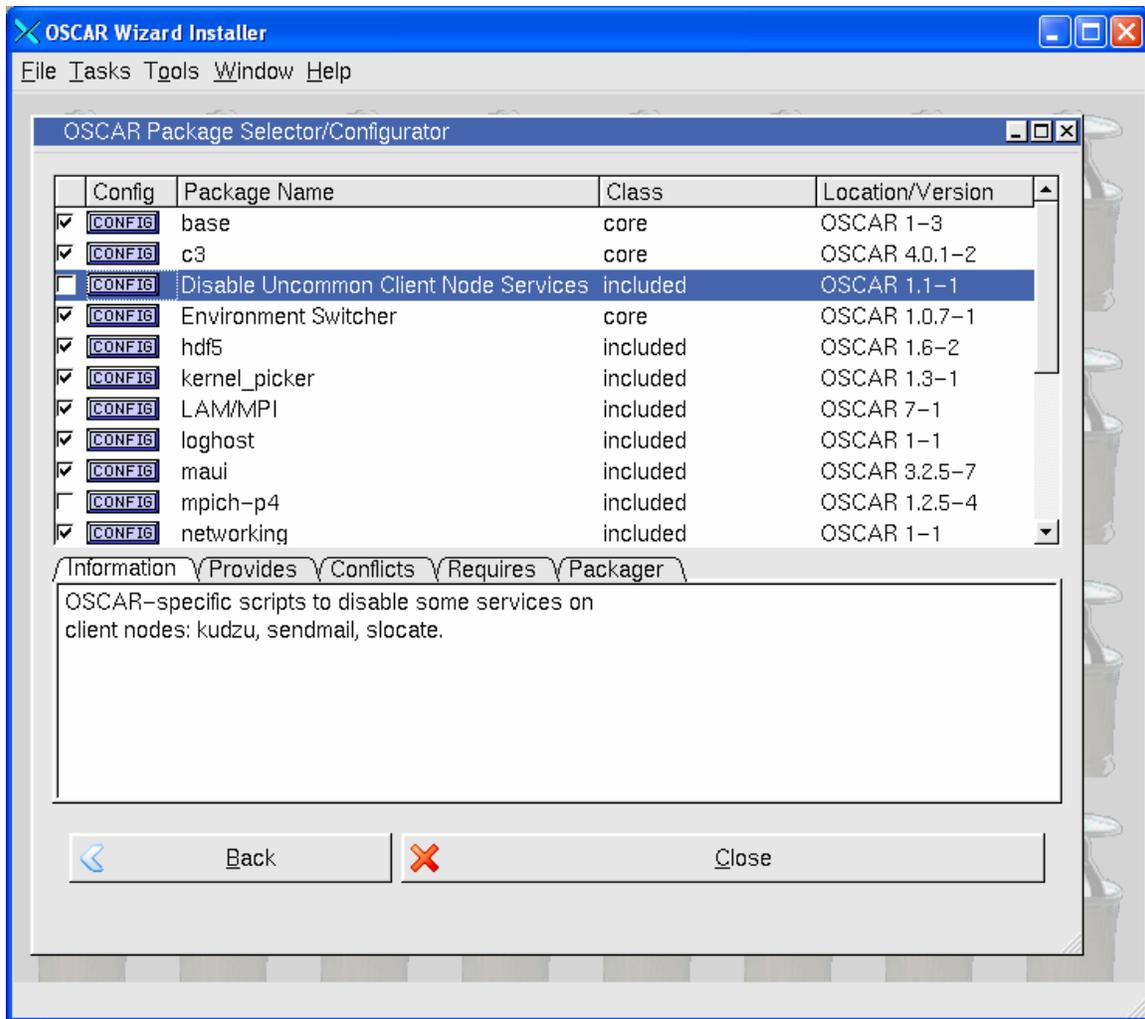


Figure 5.

## 2.7.2. New Features

Whether the master installation checkbox is checked or not for a given package, the CONFIG button will be active for that package and its Configurator window can still be opened and used. There are four major differences from the past Configurator:

1. Part of the window will show a full list of groups and nodes. Per software package, a user may select which nodes and/or groups they wish the configuration on screen to be applied/saved to. Similar to group membership modifications, conflict detection is performed at this step as well. In previous Configurator versions, a single configuration was applied to the entire cluster. This new Configurator feature, combined with node grouping support, allows a cluster to be at either end of the spectrum with respect to simple homogeneous or complex heterogeneous cluster configurations.
2. In addition to the “master” control checkbox for *installed/not installed*, another *installed/not installed* checkbox will appear per package configuration. This is to allow negative overrides via hierarchical group relationships.

3. All packages will appear in the list. Even if the package doesn't have any configuration options, it needs to appear so that it can be assigned to nodes and/or groups. This is because without a node group assignment, the package will not get installed.
4. OSCAR software package authors will have the option of displaying single- or multi-select node/group lists within their individual Configurator panels. This will be accomplished through a pre-defined *widget macro* in the package's HTML file that the Configurator will know how to interpret. These node/group list widgets may even be capable of limiting themselves to the nodes/groups which are selected for the configuration. This is possible because of the workspace API's implementation of communication between separate GUI components.

### 2.7.3. Backwards Compatibility

Package authors will not have to invest much effort, if any, into updating their "configurator.html" files to work with the new Configurator. Even though the rendering engine will be ported from Perl-Tk to Perl-Qt, the HTML code will work in either context. However, package authors will be encouraged to take advantage of the new widgets available if they can make their configuration options more robust.

## 2.8. *NEST (Node Event and Synchronization Tool)*

### 2.8.1. Image-Based vs. Component-Based Management

Original OSCAR versions used a product call LUI (Linux Utility for cluster Install) to network boot the nodes, create a local filesystem, and begin RPM installations from the server which resulted in an installed node. Let's call the LUI approach *component-based* for now. While this got the job done, OSCAR developers saw clear advantages to an *image-based* approach to node builds. This prompted the modification of OSCAR to use SIS for the task of building nodes. With SIS, an image of the node's filesystem was assembled on the server, and then 'rsync'ed out to network booted clients. While this proved to be a better option than LUI, it did not come without a sacrifice. Each method has its advantages.

#### Advantages of Image-Based

- Relative node build speed. RPM (or whichever package model used) must go through the RPM installation process for each node. Images only need to do this once. Also, multicast installs are a realistic option for images.
- Specific customizations are simple. If a user wishes to have a "/etc/foo.conf" file present on an installed node, all that is involved is adding it to the image. In the component-based model, this simple change likely requires a scripted post-install step.
- Preview capability. Before a node is built, it is trivial to get an idea of what its filesystem contains. In the component-based model, no such preview is available prior to a node build.

#### Advantages of Component-Based

- No server assembly time required. Image-based models must spend considerable time preparing an image before a node build may begin.
- Feasible heterogeneous node support. Images are very good at building the exact same node. However, if differentiation is required, the image-based model becomes infeasible very quickly. This is due to the multiplied resources (time, disk space, and image maintenance) that would be required to have a different image for every different type of node. Component-based builds can be extremely granular, only needing the space to store the differences in node description, such as an rpm list.

OSCAR has been employing an image-based model since the switch to SIS as a build utility, but in the move to support heterogeneous node installations, something definitely needs to change. However, we do not want to give up the advantages offered by an image-based model. The answer is to use a *Hybrid* model.

## 2.8.2. Hybrid Management Model

The hybrid model simply acknowledges that even heterogeneous nodes have much in common. The image-based model is used for the beginning portion of the node build, where it is most effective, but no more. Then, differences between the nodes are installed (or uninstalled, as the case may be) by the component-based model. The advantages of both the image-based and the component-based models are found in the hybrid model, though some to a lesser degree. If the difference between the image and the final node installation becomes so great that the component-based portion becomes inefficient, then the image could be synchronized with an installed node as an optimization. The component-based portion will be capable of removing OSCAR packages as well as adding them.

The utility which performs the component-based management in OSCAR does not currently exist, but it is being developed specifically for OSCAR. It will have more capabilities than simply installing and removing software. Because it will also be reading and writing node state and hardware information using ODA, it has been called *NEST* (Node Event and Synchronization Tool).

## 2.8.3. Current Limitations: Scale and Heterogeneity.

The NEST system will replace OSCAR's post installation procedure. A post-install step is an unavoidable requirement of any node installation, because it takes care of specific node configuration differences, detected configurations which aren't present until all nodes are up, and any loose end configuration tasks that an RPM might leave behind. In OSCAR releases to date, this step has been performed by a script running on the head node which iterates through all packages' "post-install" procedures, and delivers configurations to each node by brute force. While this method has worked thus far, it has clear scalability limitations, and does not work in the heterogeneous realm at all. NEST will solve both of these problems.

## 2.8.4. Scalability Help from NEST

NEST involves running a local operation on each individual node instead the server running a global operation. Configuration will be *pulled from* the server rather than *pushed to* the nodes. Because all information describing a given node can be flushed out of ODA, NEST can accomplish its task via network reads from ODA. From this information, NEST can identify packages that need to be removed or added, and it can take action if ODA says it is okay to proceed. (Certain wizard states might wish to suspend any node synchronizations, such as when the Configurator is running, for example. ODA can be used as a medium for this type of stoplight communication.) NEST will also run package-supplied API scripts on the node for post installation, pre/post uninstallation, etc. Package authors will need to convert their post-install steps to run from the perspective of a local client rather than a global server.

Making the clients do most of the post install work individually goes far in alleviating issues involving scalability, but there's another key advantage of the NEST system. NEST will not run post-install scripts in a brute force manner. The Configurator writes a timestamp to ODA for each saved configuration. When ODA applies a configuration, it copies that timestamp to a local cache for future comparison. This way, before even executing a post installation for a package, ODA can compare the timestamps on the given configuration to see if NEST needs to do anything. This feature improves NEST's efficiency considerably.

## 2.8.5. Heterogeneity Help from NEST

Because NEST runs on individual nodes, heterogeneous node configuration support is inherent in its design. By "shaking the group hierarchy tree" via network reads from ODA, all installed packages and their associated configurations will simply fall out, *per node*. NEST never needs to sort out information that belongs to other nodes, and its resulting job is simple.

### 3. Potential Change areas, not OSCAR4 Design-Specific

Changes discussed thus far include the interdependent items comprising the proposed OSCAR 4. Version 3.1 will likely have all of these characteristics. However, many other feature requests and parallel development efforts exist. These may well be a part of OSCAR 4 if they are completed. Below is a list of some of these items.

- Package OSCAR as one or more RPMs
  - Move away from the ‘tarball’ concept
  - May open the door to updatability
- Implement server-side boot control
  - Leave the nodes in network boot mode at all times and control their boot behavior from the `oscar_server`
  - Makes cluster management much easier for those who are forced to change a BIOS setting in order to modify boot order for building
- Enhanced package API
  - Add “`tested_with_oscar_versions`” `config.xml` parameter
  - Add “`minimum_oscar_version`” `config.xml` parameter
  - Update OPD/Selector to utilize these parameters
- SIS Enhancement
  - An intelligent, environment sensitive, customizable network boot package, called *mkbootpackage*. This feature would solve most problems experienced when using hardware which post-dates the SIS boot kernel’s support.
- New Packages
  - Auto-cluster-registration
  - Infiniband
  - SGE
  - OpenSSH (extracted to package level from current hard coded state)
  - Hardware control
    - Hardware control features are in demand, but hardware control will likely come in package form to begin with. A hardware control package would be a great proof-of-concept. After that point, hooks could be inserted into the core OSCAR infrastructure to take advantage of such control.

### 4. Conclusion

At the time of writing, many of the OSCAR 4 design-specific components exist in at least a rough form. Ahead of us, we have the Node Grouping GUI front-end, the Configurator, and NEST to complete. Eliminating rough edges and integrating the completed tools into the current OSCAR engine will take some time as well. After that, there will be a package conversion stage followed by testing. Version 3.1 should not be expected before July 2004 at the earliest. If you are interested in joining the OSCAR effort, please sign up for the “oscar-devel” mailing list at <http://sourceforge.net/projects/oscar>. New contributors are always welcomed.

### 5. Acknowledgements

The OSCAR project accomplishments and future plans are representative of the contributions and goals made by all members of the OSCAR core team. The core team is composed of several academic and commercial organizations which can be found on the project homepage. Congratulations to OSCAR-core for the continued success of the project.