Reprinted from the

Proceedings of the Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Wild Open Source, Inc.*

GConf: Manageable User Preferences

Havoc Pennington
Red Hat, Inc.
hp@redhat.com, http://pobox.com/~hp

Abstract

GConf is a system for storing user preferences, being deployed as part of the GNOME 2.0 desktop. This paper discusses the benefits of the GConf system, the strengths and weaknesses of the current implementation, and plans for future enhancements.

1 Introduction

GConf started out very simply in late 1999, as my first project at Red Hat. It was a straightforward response to concrete problems encountered while creating the GNOME desktop, and designed to be implementable in a few months by a programmer with limited experience (i.e., by me). Much of the current implementation (let's call it Phase One) was created at that time.

GConf saw some limited use as part of the GNOME 1.4 platform—specifically in Nautilus and Galeon—but it wasn't widely adopted until recently, as part of GNOME 2.0. It has been quite successful as a GNOME 2.0 component, both as a labor-saving device for programmers and as a foundation for some of GNOME's UI enhancements.

This paper first presents an overview of Phase One from a conceptual standpoint; then discusses the Phase One implementation. Then it goes on to briefly describe some other systems for managing user preferences, such as IntelliMirror and ACAP. Finally, it presents some initial ideas for a Phase Two version of GConf. Phase Two is very much a work-in-progress.

2 Motivation

Several years ago it was becoming obvious that GNOME's initial approach to storing preferences (a simple API for writing key-value pairs to files) had a lot of limitations:

- It did not work for preferences that affected or were manipulated by multiple applications, because there was no way for applications to know a setting had changed. GNOME 1.x contains numerous hacks to work around this, and apps have to be restarted before certain preferences take effect.
- Administration features were missing, such as applying different defaults to different groups of users, or storing defaults for many machines on one network server.
- From a programmer standpoint, applications were usually designed with

`load_prefs()` and `save_prefs()` routines, and had a number of bad hacks to update parts of the application when preferences were modified. To avoid this mess, a model-view design would be ideal.

- We wanted features such as "Undo" and "Revert To Defaults" that were complicated to implement manually.

- We wanted a way to document each configuration option, and present the documentation in generic tools for managing preferences.

- To change the default setting for a preference, it was necessary to edit each code path that loaded the setting, and modify its fallback in the case that the setting was not present.

Some of these issues can be avoided by simple elaborations to the trivial file-based API, but others require a more elaborate solution. GConf set out to decide on the right thing to do and then do it.

3 Phase One: Design

The first principle of the GConf design was to keep it simple; it had to be implemented in only a short time, and had to be comprehensible to application developers.

The second principle of the design was to make GConf a system for storing end user preferences. Support for system configuration (e.g. Apache or networking) was explicitly excluded from the list of requirements. Support for storing arbitrary data was also excluded.

3.1 Key-Value Hierarchy

From an application point of view, GConf looks like a hierarchy of key-value pairs. Keys are named as UNIX-file-like paths. Here are the default settings for the GNOME CD player for example, listed using the `gconftool-2` utility:

```
$ gconftool-2 -R /apps/gnome-cd
theme-name = red-lcd
on-stop = 0
on-start = 0
device = /dev/cdrom
close-on-start = false
```

The key `/apps/gnome-cd/theme-name` has a string value `red-lcd`, the key `/apps/gnome-cd/close-on-start` has the boolean value `false`, and so forth.

Each key may be writable or not writable; applications are expected to disable the GUI for modifying a writable key. That is, if a setting is locked down and not available to a particular user, it should be made insensitive ("grayed out").

The API for interacting with the key hierarchy has a model-view design. That is, applications receive a notification message when a key has a new value; code in the application which *sets* a value need not have any relationship to code which *is affected by* the value.

The key hierarchy is process transparent; that is, a change to the hierarchy made by one process will be seen immediately by any other interested processes as well. This avoids ad hoc hacks for propagating settings changes across the desktop, and is an essential feature when writing single applications made up of multiple processes.

A “symlinks” feature for the GConf hierarchy has been suggested several times, but links (hard or symbolic) were deliberately left out of the design because they create implementation complexity. For example, notifying interested applications when a key changes value would require the ability to locate all symlinks pointing to the key that was modified.

3.2 Values

Values are limited to a fixed set of simple primitive types: integer, double, UTF-8 string, and boolean. Lists of each type are also allowed (list of integer, etc.). Recursive lists (lists of lists) are not allowed. Lists must be homogeneous in type. Phase One also supports pairs of primitive types (as in Lisp pairs, with `car` and `cdr`), but applications have universally ignored the pair type and in retrospect it was not a useful feature.

The limitations on value types caused a lot of controversy in the GNOME community. Some developers wanted the ability to push structs or other complex data structures into the configuration database, as a programming convenience. There are several reasons why GConf does not support this feature.

First, serialized structs are essentially binary blobs, a frequent complaint about the Windows Registry. All GConf data is human-readable. While a complex type system such as CORBA could be used to create a generic serializer, and a generic un-serializer, the process of manipulating an arbitrary, possibly-recursive serialized CORBA data type is much more complex than the process of manipulating strings and numbers. With the current GConf limitations, it’s very easy to write scripts and tools that can handle any possible GConf value.

Second, many of the use-cases presented for storing arbitrary structs imagined using the GConf API for storing application data, rather than user preferences. For example, the GNOME 1.x configuration file API was used for “.desktop” files and the session manager save file in addition to user preferences. But in the requirements phase, GConf was limited to preferences only, so this was not a concern. Uses of the API that were storing preferences, rather than data, rarely benefited from anything beyond the primitive types.

Third, there are at least two simple workarounds if you need a “struct” type, which don’t have the disadvantages of actually using a serialized struct: do the serialization yourself and store a string, or use a directory containing one key for each field in the struct.

Finally, the elaborate type system needed to handle serialization of arbitrary structs would mean either binding GConf tightly to a particular object/type system, or making up Yet Another Type System, not something anyone would like to see. This would limit our ability to “drain the swamp” (Jim Gettys’s words); broadening the adoption of GConf Phase Two outside of the GNOME Project will require the client side library to be dead simple. If we had a widely-adopted object/type system, as with COM on Windows, using that object/type system to store complex types might make sense. But instead we have XPCOM, UNO, CORBA/Bonobo, GObject, QObject, ad infinitum, and no one is undertaking a serious effort to fix this problem.

I believe that the lack of a “struct serialization” API was a good decision on the whole. It places slightly more burden on application developers, but a good kind of burden; it forces them to describe their application’s preferences in clear, human-readable terms. And it deters people from storing non-preferences data

in GConf.

3.3 Schemas

Each key in the GConf hierarchy is associated with a *schema*. A schema contains metainformation about the key, including the following:

- The expected type of the key's value.
- A short one-sentence description of the key.
- A longer paragraph documenting the key and its possible values.
- The name of the application that provides the schema and uses the key.
- A default value to be used when the key is not set.

The documentation strings and default values can be localized.

The main schema-related design question was: where are they stored, and how are the default values located by applications at runtime? We went with the simplest solution: store the schemas in the configuration hierarchy itself. In addition to the normal primitive types (integer, string), GConf keys can store a schema value, containing the above metainformation.

Schemas are then associated with another key by name. That is, each key in the hierarchy may have the name of the key containing its schema value associated with it. By convention, schemas are stored under a "`/schemas`" toplevel, using names that parallel the main key hierarchy. So `/schemas/foo/bar` might be the schema name associated with `/foo/bar`.

When an application requests the value of `/foo/bar`, the GConf system first checks for a value at `/foo/bar`; if none is found, it checks whether a schema key is associated with `/foo/bar`; finding the schema key `/schemas/foo/bar`, it then looks up the value of `/schemas/foo/bar`. If `/schemas/foo/bar` stores a schema value, the default value is read from the schema value, and returned as the value of `/foo/bar`.

3.4 System Administrator's View

The system administrator sees a somewhat more complex picture of the GConf key/value hierarchy than the application does. To applications, GConf appears to be a single hierarchy of key-value pairs. Moreover, applications have no idea how the data in the hierarchy is stored.

From an administrator standpoint, GConf generates its hierarchy by merging a list of *configuration sources*. A configuration source is a concrete storage location, with three important attributes:

- a *backend*, i.e. which configuration source implementation should be used. For example, you might have a backend that uses XML files, or one that uses ACAP.
- *flags*, most importantly "read-only" or "read-write."
- the *address details*, for a file-based backend this might be the location to write files, for a network backend it might be the server's hostname.

Configuration sources are listed in a configuration file, `/etc/gconf/2/path`. Each time an application asks for the value of a

key, GConf searches the configuration sources in order until it finds a value, or runs out of sources. When an application sets a new value, that value is stored in the first writable source. However, the set will fail if a non-writable source earlier in the search path sets the value already. This allows administrators to impose mandatorythree settings.

The default GConf search path has three elements: one read-only systemwide configuration source intended to contain mandatory settings, one read-write configuration source in the user's home directory intended to store changes made by the user, and finally one read-only configuration source at the end of the search path intended to contain both systemwide default values and schema values.

To impose a mandatory value, the administrator sets that value in the read-only source at the front of the search path; to provide a system default, the administrator sets the default in the read-only source at the end of the search path. Factory defaults are kept in the schemas, so are separate from site defaults.

The documentation strings found in schemas are intended to assist administration tools in presenting a reasonable interface for performing these kind of tasks.

4 Phase One: Implementation

Even with a fairly simple design, there were some challenging implementation issues that had to be addressed for Phase One. In summary:

- Process transparency; how to make all applications see the same hierarchy and deliver dynamic notification of changes?
- Caching; the GConf hierarchy contains a fairly large amount of data in total, it would be bad if each application had to maintain its own cache of this data.
- Storing the hierarchy; a filesystem-like data structure is fairly complex to store on disk, balancing efficiency, robustness, and so forth.
- Locking; you can't have two processes trying to modify the same file at the same time.

4.1 Per-Home-Directory CORBA Server

The implementation of Phase One is to have a small per-user daemon communicating with applications via CORBA. CORBA was used as a prebuilt IPC mechanism, for speed of development. The definition of "user" in per-user was "home directory"; the GConf daemon holds a lock in the user's home directory, and if the user logs in to two machines at the same time, the same daemon instance will be shared between those machines (one machine connects to the daemon on the other).

The per-user daemon has three functions:

- it serves as a global shared cache
- it serves as a global lock on the configuration data
- since it processes all changes to the data, it can notify applications of said changes

CORBA was not a great match for the networking needs of the GConf daemon. All communication between client and server needs to be nonblocking; this can only be achieved through the use of ORBit-specific nonstandard

CORBA features. CORBA is pretty much overkill for this very simple IPC, and discourages the adoption of GConf outside of the GNOME Project.

Scoping the daemon per-home-directory wasn't the best decision either. One problem is that `fcntl()` file locking doesn't work so well; most Linux distributions are shipping buggy versions of nfs-utils that will leave stuck locks in some situations, and some combinations of NFS client OS and NFS server OS don't work quite correctly. Due to rumors of problems like this, the original GConf implementation tried to use a home-brew locking solution; but while that was portable, it was also non-working, and duplicate copies of the GConf daemon would often be created.

The per-home-directory solution was also unpopular because it involves remote TCP/IP connections (and open ports) between all machines where a user might log in using the same home directory. This turned out to be inappropriate for many sites, especially those using AFS. It also involves enabling TCP/IP for ORBit, meaning that a lot of programs are suddenly listening on open ports.

On a higher level, Owen Taylor pointed out that definining "per-user" as "per-home-directory" isn't correct anyway; a user may have multiple home directories, for example one on their laptop and one at work. You want preferences to be associated with a real world human user, not a specific login on a specific computer.

4.2 XML Backend

The GConf daemon dynamically loads backend modules that know how to read and write configuration data to some persistent storage location. Two backends were implemented for Phase One, an XML backend and a Berkeley DB backend. The DB backend was never enabled by default, and few users have tried it to my knowledge. It stores the GConf hierarchy in a single DB database file.

The XML backend stores a directory hierarchy on the filesystem that corresponds to the GConf hierarchy, and in each filesystem directory stores an XML file containing GConf values. The general idea was to split the hierarchy into multiple files (for robustness, and to avoid having to parse a singe huge file on startup, when relatively little of it might be used).

The XML backend's approach is a little bit messy, and inode-intensive. It's also surprisingly complicated to implement and has been a noticeable source of GConf bugs.

However, in practice the XML backend has worked reasonably well, now that it's been debugged; it is fairly scalable, assuming that a single directory never contains a huge amount of data. The problems of the XML backend seems difficult or impossible to avoid while using human-readable text files—the alternative design is to have one big file, which would require oceans of RAM to parse, and would put all the user's eggs in one basket.

5 Prior Art

GConf is hardly the first system for storing preferences ever invented. It's worth surveying some of the other notable systems alongside the Phase One design, to aid in planning Phase Two.

5.1 Windows 95 Registry

The much-maligned registry really isn't anything complicated; it's pretty much just a big hash table. Though at least one entire book has been written about it ([Petrusha]).

People like to compare GConf to the registry. While GConf also stores key-value pairs, it has little else in common with the registry:

- The registry stores systemwide configuration, GConf contains only user preferences.
- The registry typically contains binary data blobs, GConf goes out of its way to avoid those.
- GConf keys are documented and clearly named.
- The GConf design and application-visible semantics do not expose a specific format or location for the persistent data store.
- GConf provides mechanism for system/workgroup defaults and mandatory settings; the registry does not (but see the next section on IntelliMirror).
- The registry lacks change notification; if one application changes the registry, ad hoc hacks must be used to notify other applications. Or the user has to reboot.

(Some of the above information may no longer be accurate for newer versions of Windows; the above is based on Ron Petrusha's book.)

5.2 Windows 2000 IntelliMirror and Group Policy

While the registry is boring, IntelliMirror at least has good hype. Microsoft's white paper ([Microsoft]) on IntelliMirror cites three problems addressed by the feature:

- "User Data Management"
- "Software Installation and Maintenance"
- "User Settings Management"

"User Data Management" means that IntelliMirror keeps a copy of a user's documents in a central network location. When the user connects a machine to the network, their documents are copied to the local system. If the user edits a document while disconnected, the new document becomes the master copy and is synced to the network later. This means that documents are automatically backed up, and available on any machine the user logs on to.

"Software Installation and Maintenance" means that when a user logs on to a machine, the software appropriate for that user gets installed automatically. So users always have the same applications available in their menus.

"User Settings Management" means that user preferences (registry settings presumably) are automatically synced to/from network storage in the same way that documents and other data are synced. Also, certain settings may be locked down, and defaults may be established for particular groups of user.

These three features are most useful in combination, because providing all three allows users to easily move between machines, or switch to a new machine when their old machine breaks. Support for disconnected operation is a truly

useful feature of IntelliMirror that can't be achieved using NFS on UNIX. If you take your laptop home, edit your preferences or a document, then reconnect it to the network at work, your changes will be automatically synced to network storage.

The IntelliMirror process is driven by directory services; Active Directory stores the information about a user, including what software they are supposed to have, where their data lives, and so forth. Policies can be established for particular workgroups and applied to all users in those groups.

GConf needs to be part of an IntelliMirror-style solution for Linux and UNIX operating systems, rather than an impediment to such a solution. Other existing components of the solution might include Red Hat's Kickstart, and filesystems such as InterMezzo. For situations where disconnected machines aren't important, NFS and AFS might also be useful solutions.

5.3 ACAP

RFC 2244 defines ACAP, or Application Configuration Access Protocol. It seems that ACAP never caught on, and is more or less dead; the only server implementation is written in ML, few if any applications support it, and the web page doesn't seem to have been updated in years. ACAP was an attempt to do almost exactly what GConf is supposed to do, however, and is worth looking at. The RFC describes ACAP's design goal thusly: "ACAP's primary purpose is to allow users access to their configuration data from multiple network-connected computers." ACAP is a text-based protocol similar to IMAP.

The ACAP RFC is somewhat vague, and there's not much in the way of implementation code to look at, so some of my descriptions of the system may be inaccurate.

Like GConf, ACAP defines a filesystem-like hierarchical namespace. However, the contents of a "directory" are (a lot) more complicated than they are in GConf. A GConf directory contains entries, each of which is a name and a primitive value. An ACAP "directory" contains a difficult-to-explain object called a "dataset." [Troll] tries to explain datasets, as the ACAP RFC itself isn't very clear. They are roughly speaking Yet Another Hash Table, but with some twists.

A dataset can inherit from another dataset. This feature allows system or workgroup defaults to be set up, much like GConf's configuration source search path. However, it's apparently necessary to configure inheritance on a per-dataset basis, which seems cumbersome.

ACAP datasets can have a "subdataset" associated with them; as far as I can tell, the RFC never explains the purpose or semantics of subdatasets.

ACAP has ACLs for settings, and quotas for individual users. ACLs allow settings to be made read-only, in order to impose mandatory settings. Quotas are useful for obvious reasons.

The concept of multiple users is visible to the ACAP client application. ACAP also exposes the inheritance chain (search path) for looking up values. GConf keeps this information purely in the configuration of the server; the client sees only a single hierarchy.

ACAP provides server-side sorting and searching, allowing clients to search through data stored in ACAP without downloading all the data.

Unlike GConf, the ACAP design goals include

storage for “lightweight data,” such as an address book, in addition to preferences. GConf is not intended to be used in this way; the assumption is that users will have somewhere to put documents and such anyway. By limiting GConf to preferences only, a GConf configuration source can realistically be locked down entirely (no writable locations), and the GConf design can assume that sending the value of a key over the wire is a fairly fast operation. The GConf implementation can be simple (does not need to scale to large data values), and the design need not support a rich type system.

ACAP does not have a standard concept like GConf’s schemas, though arbitrary attributes can be added to a dataset. Keys in ACAP don’t come with documentation as GConf keys do.

ACAP supports IMAP-like authentication (SASL), so works nicely with Kerberos and such.

The article *ACAP vs. Other Protocols* ([Wall2]) summarizes ACAP’s design as follows:

> The key characteristics of ACAP are:
>
> - ACAP is designed to accommodate disconnected use
> - ACAP is designed to allows server data (and data structures) to be writable by user/clients
> - ACAP is designed to handle potentially (though not necessarily) large sets of data
> - ACAP is designed to allow granularity in access to data through an Access Control List mechanism
> - ACAP is designed to allow per-user storage of information (accommodating problems of mobile, disconnected, and "kiosk"-model users)
> - ACAP is designed to allow client definition of data fields, allowing user-side flexibility
> - ACAP is designed with per-user security and authenticated operation states
> - ACAP is structured to enable server-side searching.

GConf needs to be modified to support many of these design features. However, I would not advocate using ACAP as-is; ACAP doesn’t seem quite right, it is not compatible with the current GConf client API, and because no one else uses ACAP there’s little or no value to following an existing RFC. If a reasonable free ACAP server implementation existed (vs. an old one written in ML), it might be interesting to use as the backend for GConf Phase Two.

6 Phase Two

6.1 Design Changes

The application-visible GConf model has worked reasonably well so far. The model/view architecture, process transparency, filesystem-like hierarchical namespace, and so forth are straightforward yet powerful. The current architecture seems roughly correct in terms of balancing simplicity and feature set; much more complex, and people would not be able to use it correctly, much simpler, and it would not address all the problems that need addressing.

However, the current GConf *implementation*

needs a lot of work to scale beyond single-user systems. Like ACAP, it should be client-server oriented. Like IntelliMirror, it should provide for disconnected operation. In general, the Phase One implementation's equation of "user" with "home directory" was wrong; users may have multiple machines, including a laptop.

6.2 Implementation Changes

Phase Two should be mostly invisible to applications, in fact it could conceivably be done without modifying the GConf ABI as shipped with GNOME 2.0. The changes will all be in the implementation.

6.2.1 Client-Server Architecture

In short summary, here is what I envision:

- The GConf daemon will become per-user-login rather than per-home-directory. (Side bonus: it can use a guaranteed-not-to-be-on-NFS directory for locking.) It will listen on a UNIX domain socket and communicate with a simple, fast custom protocol designed for "oneway" mode (avoiding round trips).
- The client side of GConf will become as trivial as possible; it will not use GLib or CORBA, just a tiny custom protocol. The current GConf client library will exist as a GNOME-friendly API but will wrap the more general API.
- More of the daemon's functionality will be moved to the loadable backends; in particular, change notification will come from the backend, instead of from the daemon.
- The default backend will know how to store data in two places; a local filesystem cache, and a remote server. The local cache will be used when disconnected. On connection to the network, the local cache is fully synced with the remote server. While connected, it will sync on a regular basis. For standalone systems, the local cache is simply never synced. (The name "cache" is somewhat misleading, since the cache never expires.)
- The remote server used by the default backend has to be implemented. It will be a system daemon, and will serve requests from multiple users. It should support authentication via SASL (and thus Kerberos, etc.). The remote server is heavily inspired by ACAP.

The general goals are to move to a more secure client-server architecture, support disconnected operation, encourage adoption outside of the GNOME Project, increase robustness, and improve performance.

One outstanding question is how to improve on the current tree of XML files for storing the GConf database. The tree of XML files is robust, but complex to implement and fairly inefficient in terms of both space and speed.

6.2.2 Small Tweaks

In addition to the big-picture rearrangement, there are a number of smaller features to be creeped.

- Hints for interpreting values. This feature adds a simple descriptive string such as "keybinding" to a key, indicating that the

key's value is to be interpreted in a particular standardized way. Generic configuration tools can then provide a nicer UI for editing the key.

- Atomic change sets. This feature allows a block of changes to be made as a unit, without exposing the intermediate state to applications.

- Prompting for authentication. This feature means that backends have a way to ask the user for passwords and other authentication information.

- Clean up the C API a bit. The API has various historical artifacts and just plain mistakes, and better ideas have appeared as it has become clearer how real applications use it. Most likely a new API will be introduced alongside the old, and both will be supported for a while.

- Convenience widgets for preferences dialogs. It's nice to have "data-aware widgets" for GConf, such as a text entry box associated with a string value in the GConf hierarchy.

- The server will probably need to enforce a space quota on individual users to avoid denial-of-service attacks.

- It might be nice to allow admins to define a separate configuration source search path for a particular subtree of the GConf hierarchy, instead of defining a single search path for the whole thing.

- Searching through data needs to be implemented on the server side, so that admin tools can efficiently provide a search function.

7 More Information

For more information about GConf, including an online copy of this paper (perhaps an updated version), see `http://www.gnome.org/projects/gconf/`. To report bugs, see `http://bugzilla.gnome.org`. To discuss GConf, join `gconf-list@gnome.org`.

8 Acknowledgments

Many people have contributed to GConf. All the hackers at Red Hat, especially Owen Taylor and Jonathan Blandford, have given valuable feedback and design suggestions. Red Hat deserves credit for supporting the initial development of GConf when I first arrived at "RHAD Labs" several years ago; Labs director Michael Fulbright gave me the freedom to work on GConf. Thanks are also due to Colm Smyth at Sun for his contributions of code and ideas, and to Sun Microsystems in general.

Dave Camp (now at Ximian) and Kjartan Maraas tie for the honor of being the first outside contributors to GConf, according to the ChangeLog. Since then many people have helped out, too many to list here. Thanks to everyone.

References

[ACAP] *RFC 2244* `http://asg.web.cmu.edu/rfc/rfc2244.html` (1997)

[Microsoft] Microsoft, Inc. White Paper, *Introduction to IntelliMirror Management Technologies*

[Petrusha] Ron Petrusha, *Inside the Windows 95 Registry*, O'Reilly and Associates. (1996)

[Troll] Ryan Troll, *ACAP Dataset Model* `http://www.ietf.org/proceedings/99nov/I-D/draft-ietf-acap-dataset-model-01.txt` (1999)

[Wall] Matthew Wall, *The Application Configuration Access Protocol and User Mobility on the Internet* `http://asg.web.cmu.edu/acap/white-papers/acap-white-paper.html` (1996)

[Wall2] Matthew Wall, *ACAP vs. Other Protocols* `http://asg.web.cmu.edu/acap/white-papers/acap-vs-others.html` (1996)