Configuration Data Deserves a Database

Adwait Tumbde, Matthew J. Renzelmann and Michael M. Swift
University of Wisconsin-Madison
{adwait,mjr,swift}@cs.wisc.edu

Abstract

Configuration management is one of the largest causes of system and application failure. In one study, twenty four percent of Windows NT downtime was attributed to system configuration and maintenance [13]. Furthermore, system configuration is a large expense: 60-80% of the total cost of computer ownership is system management [3]. The problem is increasing as systems and applications get larger.

We seek to address a key aspect of this problem, *configuration storage*: how configuration data is stored and managed by the OS. Existing mechanisms, such as files in Linux, property lists in MacOS X, and the registry in Windows do not adequately support application and administrator needs. Instead, we propose to store configuration data in a relational database. A database back-end simplifies many common application and management tasks, as well as providing key services needed for dependability, such as logging and transactions.

1 Introduction

Modern operating systems are incredibly flexible, but this flexibility comes at a large cost: configuration management. Configuration management adversely impacts both system availability and the cost of ownership. The Computing Research Association reports that 60-80% of the cost of ownership is due to system management [3]. A study of Windows NT systems indicated that configuration management was responsible for 24% of downtime [13]. Common configuration problems arise when an install or uninstall process fails, when an application or administrator corrupts data, or when an upgrade overwrites common configuration settings incorrectly [6].

A major contributor to the difficulty of system management is the *organization* of configuration data. For example, Windows XP stores settings in a configuration registry, a hierarchical database of key-value pairs. A typical machine has approximately 200,000 settings [7]. However, additional information of use to administrators, such as default values, schema, and comments are not available in this model. Furthermore, the rigid hierarchical structure prevents applications that interact from associating their joint configuration with both applications. Conversely, Unix systems traditionally store data in application-defined text file formats. This provides flexibility and speed to applications but complicates management, as each application requires a separate parser.

Furthermore, application developers and administrators construct higher-level services on top of configuration storage, but must do so in an ad-hoc, application-specific fashion. For example, applications commonly implement an inheritance model, where users may have per-user settings that override system-wide defaults. As another example, many applications implement *profiles*, a grouping of settings that may be selected en masse. Finally, applications interested in robustness must implement transactions and rollback when modifying configuration settings, to ensure that they can recover after a failure. Unfortunately, such improvised services raise the complexity of configuration management. These services remove the semantics of configuration from the data, because applications now use their private logic to construct internal settings from the stored data.

We seek to simplify the job of programmers and administrators by offering a better configuration storage mechanism. This mechanism simplifies many of the services currently provided by applications, supports common administration tasks, and ensures that the configuration state of an application is visible to administrators and not hidden behind a layer of application semantics.

Our Approach. In this paper we present the design of the Configuration Data Management System, CDMS, built on top of a relational database. CDMS is capable of storing all configuration data, including OS settings, application settings, and user preferences. It not only presents a standard hierarchical data model to applications, but also groups related settings into *configuration objects* that may be assembled into a collection of settings visible to an application instance. This service supports operations that applications must currently implement, such as transactions, inheritance, profiles. Additionally, CDMS provides a rich query interface and a log of all persistent changes.

We believe databases are acceptable to use within operating systems for two primary reasons. First, the quantity of data is now so large that simple storage techniques do not scale. A desktop or server system may have hundreds of thousands of settings exceeding twenty megabytes. Second, embedded databases are now common in many systems and are usable early in the boot process. For example, Microsoft sought to place database functionality into the file system with WinFS [8].

The next section describes the shortcomings of existing configuration storage systems. Section 3 details the architecture of CDMS, and Section 4 compares it against storing configuration data in the file system. We present related work in Section 5 and conclude in Section 6.

2 The Problem

We find ample evidence in modern operating systems that existing methods of storing configuration data are inadequate. While we primarily discuss Windows and Linux, we find problems in MacOS X as well.

Ad-hoc representation. In Linux, there is no single configuration service, leading to hundreds of application-specific data formats. This variety leads to formatting errors when changing settings. While Windows supports a common storage system, the registry, it does not provide high level features such as naming. applications that link settings to other applications use their own naming convention, such as globally unique identifiers (GUIDs) to indirectly reference other applications. However, the context for looking up these strings is known only to the application, which conceals the semantics of these links from administrators. This usage creates implicit dependencies from one part of the registry to another, thereby increasing the possibility of a management error.

Encapsulation. Hierarchical configuration storage, either in the file system or in the registry, paradoxically prevents applications from encapsulating their data in a single location. Settings that relate one application to another must belong to either application, or to a third party. For example, Figure 1 shows the distribution of Microsoft Office 2003 configuration data compared to the overall structure of the registry. As the figure shows, Office stores settings throughout the entire registry. As a result, administrators have difficulty in identifying the settings related to a single application.

Reliability. Configuration data corruption due to aborted or misguided management operations often impacts application and system reliability. For this reason, Windows XP includes a rollback mechanism in its install facility, Windows Vista includes transactional registry access, and the System Restore feature allows the whole registry to be checkpointed. However, these features do not go far enough. Rollback during installation is of little use when failures occur during other management tasks, and external configuration changes may prevent rollback from succeeding. For example, we manually injected errors into the configuration data for Adobe Reader 7.0.8 by deleting settings and found that neither the uninstaller nor the installer would function. In addition, system-wide rollback undoes *all* changes and not just those at fault. Rather, the ability to undo any change at any time is needed to fully support reliable management [2].

Scoping. Configuration data commonly applies to a fixed scope, such as a single user, an application, or the entire machine. MacOS X [1] and GConf [10], for example, provide a fixed set of scopes with inheritance for propagating settings from a global scope to a per-user scope. However, a fixed set of scopes cannot support common usage scenarios, such as sharing printer or network settings between selected applications. Furthermore, these systems store only a single copy of each scope, so that the users cannot choose at runtime which settings to use. Applications that support switching between groups of settings, such as Mozilla with user profiles, must implement the feature themselves. As a result, the semantics of which settings are in effect are hidden inside application logic and are not visible to generic management tools.

In summary, we have identified several features where existing configuration storage systems fall short resulting in increased management cost and decreased reliability.

3 Configuration Data Management System

We propose to store configuration data in a database. To this end, we designed the Configuration Data Management System (CDMS) that addresses the problems described in the previous section. We discuss the organization of CDMS, its data model, and the services it offers.

3.1 CDMS Overview

CDMS is a system-wide service for storing all configuration data, including system settings, applications settings, and user preferences. This service provides a global view of the configuration data in a system, which facilitates development of shared management tools. The centralized system also provides services such as transactions and querying to all applications, avoiding the need for reimplementing these on a per-application basis. Finally, a centralized system provides better control over settings to administrators, as all settings are visible through a common interface.

The architecture of CDMS is illustrated in Figure 2. CDMS implements our novel data model on top of a relational database and augments the database with an interface for application programmers and a data model specific to configuration data.

The database storage engine provides several instant benefits compared to files or special-purpose stores. First, a database provides transactional updates, which improves reliability. Second, databases log updates to provide atomic commit, and CDMS retains the log to create a persistent record of all changes to configuration settings. This log is exposed to administrators, enabling rollback of faulty changes as well as time-travel debugging [12].

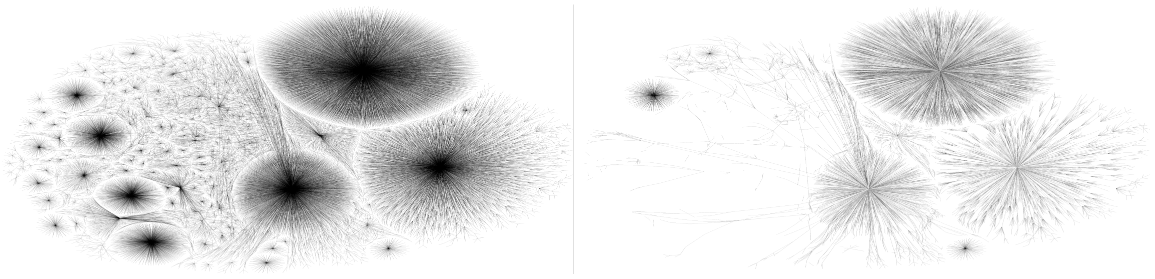

Figure 1: **The graph on the left represents the Windows registry as a whole. Nodes represent registry keys and edges denote a parent-child relationship. The graph on the right includes only registry keys added during Office 2003 installation.**

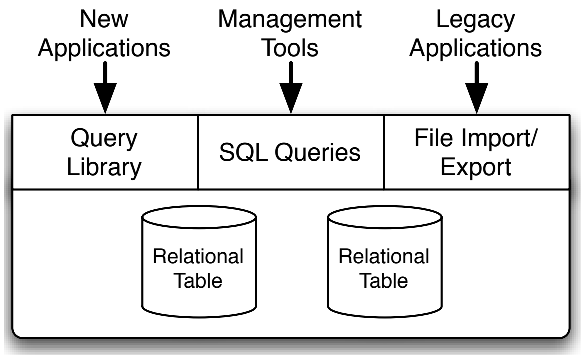

Figure 2: **CDMS Architecture**

A unique element of CDMS is its data model, which provides a two dimensional system for organizing data. We now describe how settings are named in CDMS and how they are organized into *configuration objects*: groups of related settings.

3.2 Data Model

Existing configuration management services such as the Windows Registry, GConf, and MacOS X property lists store data as a hierarchy of key-value pairs. CDMS instead stores flat key-value pairs in relational tables; the hierarchy is imposed by a naming mechanism. Values default to text, but may be optionally interpreted as a number, a file name, a link, or other type, if metadata indicating the type is stored.

Unlike other configuration services, CDMS provides another dimension of organization: settings related to a single user, application, or service are grouped into objects. This allows related settings that have unrelated names to be associated. These objects represent a group of settings in a *scope* and are the building blocks of an *inheritance* mechanism, where an object may inherit settings from many other objects.

Naming. We adopt the naming convention of Mozilla Firefox and give each configuration setting a dotted name, leading with a vendor, application, or service name and ending with the name of the individual setting. For example, an Apache setting might be named `apache.v2.timeout`, indicating that the application name is "apache" version 2, and the setting name is "timeout." While simple, this name format supports data already stored in the Registry or property lists. Common operations such as enumeration can be implemented as queries over names. For example, a list of printers may be specified as `printer.color-floor1`, `printer.bw-floor3`, and may be enumerated by querying for `printer.*`. The database query engine enables this style of access.

Like other configuration systems, this naming mechanism allows applications to group related settings. In addition, it provides a global namespace, which allows a setting to refer to other settings directly. For example, one setting may take on the value of another setting by storing a symbolic link. This enables a new version of an application, with a different configuration schema, to refer to values from previous versions. In contrast, the Windows Registry does not make use of a global name space, and instead uses application-specific identifiers, such as GUIDs, to reference other settings.

Configuration Objects and Inheritance. In addition to name-based grouping described above, CDMS also allows grouping using *configuration objects*. A configuration object is a named group of related configuration settings and is independent of the naming mechanism. For example, objects may exist for application defaults, user preferences and system-wide mandatory settings, as shown in Figure 3. These objects may contain overlapping settings. These overlaps are resolved by *inheritance*, in which a configuration object overrides the settings of its ancestors. Unlike files, where an application must name the file containing a setting, a database can rapidly query a large collection of configuration objects to find the relevant settings.

Each installed application or service has at least one configuration object. Users who wish to customize an application, for example with their own preferences, have an additional configuration object for the application. To execute an application with both per-application and per-user settings, CDMS supports *configuration spaces*, an address space of configuration data that maps names onto values. This mapping is created by assembling a group of configuration objects into an ordered list. Settings at

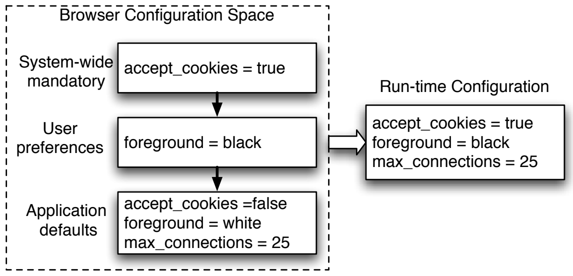

Figure 3: **Configuration objects, Space and Inheritance**

the front of the list override settings further away, allowing users or applications to override system settings on a case-by-case basis. An example configuration space for a browser application is shown in Figure 3. The system-wide mandatory settings, user preferences and application defaults form an ordered list with the system-wide settings object at the head. Windows Vista provides a similar function to allow untrusted applications to modify a private copy of global settings, but the feature is not generally available for use by applications or users.

Configuration spaces enable flexible inheritance scenarios, such as applications sharing a configuration object with common preferences (e.g. default printer). In addition, mandatory system-wide settings may be implemented by forcing a system-wide configuration object to be the head of all configuration spaces. These spaces provide better encapsulation of settings than files because all the settings belonging to an application go into a single configuration space, even those that impact system-wide features. Consequently, administrators can quickly find all the settings of an application.

Inheritance is implemented by the database query processing. When retrieving a setting by name, the database consults a view over the data specific to the space that merges settings from multiple objects. This can be implemented as a query over all configuration objects in the space, choosing the top element from the results sorted according to the list. Thus, CDMS provides inheritance as a system service rather than as an ad-hoc feature implemented by applications.

3.3 Data Storage and Access

CDMS stores key-value pairs and associated metadata, organized into one table per user along with one or more tables for system-wide data. Additional tables are maintained by CDMS to track global information. Organizing data as per-user tables corresponds closely with the operating system's notion of a security domain.

CDMS provides three interfaces to access data: a library for applications, a file import/export mechanism for legacy applications, and a query language interface for administrators. The library provides a simple interface to read and update named configuration settings, optionally within a named configuration object. In addition, it provides wild card searches to enumerating related settings and transactions to atomically group reads or updates.

For legacy applications that have not been updated to use the library interface, CDMS supports file import/export through *wrappers*. These wrappers translate the data from the database format to the format expected by the application [5], and could leverage XSL transformations to convert from XML-formatted data.

The increasing volume and complexity of the configuration data mandates a powerful query interface. CDMS supports direct queries using SQL. This exposes the full relational power of the database, for example allowing searches for related entries. Furthermore, SQL access provides a basis for writing management tools, which can issue SQL queries to read and modify settings.

3.4 CDMS Services

The core features of the CDMS storage and data model can serve as the foundation for additional functionality.

Profiles. Some applications support multiple groups of preferences and allow a user to choose a group to use, for example based on his current project. Profiles are easily provided by CDMS with configuration objects. Each profile stores the settings unique to the profile in a single object, while settings common to all profiles are stored in a separate object. To use a profile, the user constructs a configuration space with the per-profile object as the head followed by the common object. The same feature can be used for system settings, such as adapting network settings to different environments.

Time Travel. CDMS can provide time travel either through its persistent log or by snapshotting configuration objects. With the log, a user or administrator can roll back the changes to all objects, any single object, or just a specific change. Time travel can also be implemented by snapshotting the configuration objects in use and starting an application in a configuration space using the snapshot.

Multiple Versions. Multiple versions of an application or service can coexist because their settings are stored in distinct configuration objects. Installation of a new version will not overwrite the configuration settings of older version. A user or administrator can then select the desired version by constructing a space referring to that version.

CDMS improves reliability and simplifies management by (1) storing data in a database, which supports transactions and logging, and (2) grouping settings into configuration objects that may be organized into a configuration space. CDMS moves the semantics of which settings are in effect out of application logic and into the inheritance mechanism, which exposes the configuration of applications to administrators.

4 What About Files?

A traditional argument against configuration services is that they lack the flexibility and easy access of text files. In this section, we describe how databases can achieve and add on to the benefits of the text files.

Copying. Copying and renaming of files enables alternate configurations, selective backup, and passing configurations to other users or systems. CDMS can provide the same functionality, either by exporting data to a file or by duplicating data into a new configuration object within the database. These new objects can serve as a backup or as configuration for other users and applications.

Metadata. Text files, by their flexibility, simplify the addition of comments and other metadata, including default values, to configuration files. For example, 683 of the 940 lines in the distributed `httpd.conf` file for the Apache web server are comments. Many of these comments are optional values, sample settings, and descriptions. CDMS can support these same services equally well by attaching additional text columns to the table that contain comments or other data for human consumption. Optional settings can be supported with an "enabled" bit on all settings.

Tool support. Configuration management tools can benefit from the rich body of functionality provided by SQL. For example, the database query engine can provide search functionality which exploits data semantics like naming in contrast to a simple *grep* call. Common scripting languages, such as Perl, Python, WSH and Visual Basic already support APIs for querying databases, directly supporting management tools in those languages.

5 Related Work

Several prior projects have sought to change how configuration data is stored. GConf [10] and Nix [4] both provide new services, although in restricted domains: GConf only applies to user preferences and Nix to package management. GConf may optionally store preferences in a database, but does not expose database features like queries to administrators.

The notion of separating configuration into objects that can be optionally applied is a core feature of Windows group policy objects [9], but these are only used for system settings and not application or user settings. In addition, there is no hierarchy, so only a single object covers a particular setting.

Several aspects of CDMS have been proposed, but not as a single package. Logging configuration changes is one aspect of Flight Data Recorder (FDR) [11]. However, FDR is a full-system tracer, whose overhead may not be appropriate for many cases. Databases have been used for storing configuration data [5], however this approach exported text files and hence could not support applications that modify their own configuration, such as common desktop applications.

6 Conclusion

We identify several shortcomings of existing methods of storing configuration data. Based on these problems, we propose to store data in a relational database. This provides several key benefits: transactions and logging, a rich management interface via a query language, and inheritance and profiles via configuration objects. Furthermore, we counter the standard arguments for configuration files by demonstrating that similar or greater functionality can be provided by a database.

References

[1] Apple Inc. Runtime configuration guidelines. `http://developer.apple.com/documentation/MacOSX/Conceptual/BPRuntimeConfig/BPRuntimeConfig.pdf`, 2006.

[2] A. Brown. Toward system-wide undo for distributed services. Technical Report UCB/CSD-03-1298, EECS Department, University of California, Berkeley, 2003.

[3] Computing Research Association. Final report of the cra conference on grand research challenges in information systems. `http://www.cra.org/reports/gc.systems.pdf`, 2003.

[4] E. Dolstra, M. de Jonge, and E. Visser. Nix: A safe and policy-free system for software deployment. In *18th USENIX LISA*, 2004.

[5] J. Finke. An improved approach for generating configuration files from a database. In *14th USENIX LISA*, 2000.

[6] A. Ganapathi, Y.-M. Wang, N. Lao, and J.-R. Wen. Why pcs are fragile and what we can do about it: A study of windows registry problems. In *2004 IEEE DSN*, 2004.

[7] E. Kiciman and Y.-M. Wang. Discovering correctness constraints for self-management of system configuration. In *1st Intl. Conf. on Autonomic Computing (ICAC)*, 2004.

[8] D. Malkhi and D. Terry. Concise version vectors in WinFS. In *19th. Intl. Symp. on Distributed Computing*, Sept. 2005.

[9] Microsoft Corp. Windows server 2003 group policy. `http://technet2.microsoft.com/windowsserver/en/technologies/featured/gp/default.mspx`.

[10] H. Pennington. Gconf: Manageable user preferences. In *2002 Ottawa Linux Symp.*, June 2002.

[11] C. Verbowski, E. Kiciman, A. Kumar, B. Daniels, S. Lu, J. Lee, Y.-M. Wang, and R. Roussev. Flight data recorder: Monitoring persistent-state interactions to improve systems management. In *7th USENIX OSDI*, Nov. 2006.

[12] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration debugging as search: Finding the needle in the haystack. In *6th USENIX OSDI*, Dec. 2004.

[13] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked windows nt system field failure data analysis. In *1999 Pacific Rim Intl. Symp. on Dependable Computing*, Dec. 1999.