UNIVERSITY OF OSLO
Department of Informatics

A Potpourri of System Configuration Concepts

Master thesis

Tam Weng Seng

23rd May 2005

Contents

Abstract

For many reasons, large and small installations of computers can benefit from Automated Configuration Management tools. All the processes from installation, configuration, to maintenance and updating the computers can benefit from automation for the following reasons.

- Consistency across all the machines.
- Timeliness in maintenance and updates
- Simplify the process through the use declarative instructions.

Meanwhile in software configuration management, they are examining the problems of identifying, controlling, monitoring and verifying changes in software development projects. To complicate matters, some of the reasons for software configuration management

- Consistency in the source code.
- Timeliness in updates to the project members so that they have what is needed.
- A need to simplify documentation and development of complex projects.

Subsequently, the purpose of this thesis is to understand how concepts from Software Configuration Management can aid the development of the field of System Configuration. To achieve this purpose, this thesis will start with an examination of the similiarities between SCM and System Configuration. This will be followed by an examination of different key concepts in System Configuration and the following three different tools that have taken different approaches to the problem.

- ✓ Cfengine
- ✓ ISconf
- ✓ LSconf

With an understanding of how System Configuration and SCM are similar and an understanding of many of the major concepts in System Configuration, the next step is to examine some of the difference between the two fields. From there, it should be possible to see how some concepts from SCM could be applied to System Configuration. It should also be possible to examine concepts from System Configuration that could be applied to SCM.

Introduction

"Configuration Management (CM) is a management discipline traditionally applied to hardware development whose application in conjunction with other disciplines leads to orderly and structured system development. CM is generally concerned with the consistent labeling, tracking, and change control of the hardware elements of a system."
-Edward H. Bersoff [01]

In working with computers, both programmers and system administrators have realized the importance of configuration management. To differentiate the study of configuration management from the different perspectives, I will refer to Software Configuration Management (SCM) as the study of configuration management from the software engineering perspective and System Configuration as the system administration perspective. In the field of system administration, the growing number of desktop computers, and the increasing size of server farms, and clusters create a need to configure and maintain a large number of machines. The need for this can be seen in freely available programs such as BCFG, Cfengine, ISconf, and LCFG. It can also be seen in commercial programs such as BigFix Enterprise Suite from BigFix inc., HP Utility Data Center from Hewlett Packard (HP) and Tivoli from International Business Machines (IBM).

From a technical standpoint, the problem of configuring these computers increases with the number of computers to configure. Some of the difficulties come in the form maintaining a consistent and correct configuration on each computer and timeliness in changes and updates. Another problem is that as the number of computers grows, so does the complexity of the managing relationship between the computers. For example, a change in IP addresses of all the Domain Name Servers (DNS) may cause a cascade of changes that must be made to all the clients. In the event that a DNS has been delegated the responsibility for several domains, the changes may involve several organizations. Adding software or enabling a user to access restricted resources on a computer may require updating configuration information in a server. While adding more people to the task of configuring the computers may be a quick solution to the problem, this does not eliminate the problem and it creates new problem.

Meanwhile, in the field of software engineering, large teams of programmers have been assembled to tackle larger problems. Unfortunately, as programming teams grow in size, so do the problems associated with programming in teams. One of the problems is that multiple programmers may need to work on the same file. They may even require making changes to the same part of a file. Another problem in having a large programming team is the distribution of information, source code or binary components to team members in a timely fashion. Subsequently, as a project grows in size, it also grows in complexity. This complexity is reflected in the number of files, functions and data structures that make up the source code.

Another motivation of increasing relevance to the system administration is ethical and/or legal obligations. As more and more sensitive or confidential information is collected at one point, it could become a target for abuse. For example, financial information, such as credit card number, bank account numbers; medical information, such as treatment plans, medical results from tests, medicines; and proprietary information, such as source code, armored car routes, proposals. With financial information, cloning credit cards for fraudulent use could be a problem. Identity theft by using the personal information collected from the organization with poorly configured systems could be another problem. Consequently, countries may begin to hold system administrators responsible for maintaining due diligence in the administration of the computers. For example, in the US certain types of information are covered by the Sarbanes-Oxley Act of 2002. Under this act of law, IT systems used for tracking financial information in these companies must be audited and found compliant with the law. One implication for a system administrator could be that they must be able to show an auditor that the systems involved in gathering financial information are reasonable secure from undetectable tempering. Regardless of the legal obligations, the ethical obligation of protecting sensitive information is unavoidable and must be taken seriously if system administration is to be considered a profession.

To address this problem, process management and automation are a part of many proposed solutions. At the heart of process management is documentation. Unfortunately, as experience as has shown documentation tends to be neglected. The reason that documentation tends to be neglected is because it tends to be tedious and highly personalized. In addition, the more complex is the system to be documented, the more complex is the task of documentation. Subsequently, to address those issues standards established in the process management to make documentation more accessible to readers who did not author the documents. Automation should reduce the tediousness and increase the likelihood that the documentation is both accurate and updated in a timely manner. It should be noted that documentation that is out of date is arguably more harmful than if there were no documentation.

To conclude the introduction, this thesis will attempt to show how Software Configuration Management (SCM) and System Configuration are closely intertwined. It will attempt to use concepts from SCM to develop an Iterative Model similar to the concept in Software Engineering.

Software Configuration Management (SCM)

"Configuration Management is the process of identifying and defining the items in the system, controlling the change of these items throughout their lifecycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items."
-IEEE Standard 729-1983 [07]

As mentioned in the introduction, both programmers and system administrators have realized the need for configuration management. Programmers working in large teams across many different sites with the requirement of supporting many different operating systems and hardware combinations also face a similar problem. Although software configuration management has different objectives, there are concepts that could be used in SC.

In this chapter, the four main SCM activities will be discussed. In doing so, an argument will be presented showing how SCM and system configuration are related. Having established some ground of similarity between the two fields, it is hope that this will be sufficient to convenience people that the two fields can benefit from each other.

With this in mind, IEEE standard 729-1983 as well as IEEE standard 1042-1987 and IEEE standard 828-1998, mentions out four areas of configuration management and is summarized by Susan Dart [02]:

- ✓ Identification: A system that breaks down a product into the different components, while showing their relationships to each other, so that each component can be uniquely identified.
- ✓ Control: This aspect covers the release of a product and changes throughout the lifecycle of the product by creation of snapshots so that all releases are consistent.
- ✓ Status Accounting: This aspect records and reports on the status of different components and requested changes. In executing this aspect of SCM, the information gathered should enable people to generate the necessary reports and should aid in the next aspect of SCM, Audit and Review.
- ✓ Audit and Review: This aspect is to track the completeness of a product and to help ensure that the components are well-defined and consistent with each other.

There are 3 additional aspects that may be found in some SCM tools are proposed by Susan Dart [02] :

- ✓ Manufacture: The process of building the final product in an optimal manner.
- ✓ Process Management: This aspect is an attempt to enforce organizational procedures, policies and the lifecycle model.
- ✓ Team Work: This aspect is to allow multiple users to work together.

From this overview of some of the key areas of SCM, this chapter will turn to examine each of these areas to see how they could apply to System Configuration. This is needed to establish some common ground between the two fields and to form the basis of the argument that there are applicable to each other.

Configuration Identification

The aspect of Configuration Identification is to be able to uniquely name and describe each configuration item that needs to be managed. In addition, this aspect covers documenting the relationship between configuration items. In SCM a configuration item includes source code, user documentation, test cases. It also includes parts of the development environment such as compilers, programming tools, and the operating system [06].

In System Configuration, at a very basic level, configuration files and source code are comparable. While source code is made up of explicit instructions to a compiler, configuration files are explicit instructions to a program on how it should behave. For example, the configuration files for web server will include instructions on where to locate the files making up the website. It can also include instruction restricting access to certain areas depending on the source IP address of the requester, and whether or not users on the system are permitted to have home directories.

On a more general level, configuration items and components that have to be managed by a system administrator are may be the same objects that a programmer might track as a configuration item. For example, the program binaries and dynamic libraries used by software could be the same files that a system administrator has to install to ensure the proper function of software on a computer. The C compiler used to compile a new program may be the same compiler used to generate a new kernel or driver for a new piece of hardware.

Another general similarity between the two comes in the form of relationships between the different components or configuration items in a program or computer system. With System Configuration, the impact of software dependencies affecting a computer is discussed in paper such as [16], [17] and [18]. It interesting to note that the solution discussed in [17] might have just as easily been implemented by a frustrated system administrator trying to install an open source program on several flavors of UNIX as it was most likely the developers of this software by the vendors.

Meanwhile in [16] and [18], these papers show how the dependencies between different software packages, especially the dynamically linked libraries, are crucial to the proper functioning of a computer. This need is even more evident as it has been observed that dynamic libraries maybe updated over time and that different software packages on the same computer may require different versions of the same library to function. To address this problem, all package management software such as Debian Package (dpkg) or RPM record dependency information and attempt to resolve the dependencies of the packages at installation or update. Unfortunately, these systems do not guarantee that the configuration items in each of the software packages are unique and a set of dynamic libraries might overwrite another version of the same set libraries or the symbolic link to those libraries. This can cause some program to fail as different program may require different versions of a library [16][17][18]. Another potential problem is that these systems often do not take note of configuration specific dependencies. For example, a package might require that a web server daemon to be listening on a non-standard port.

To conclude this point, a thorough Configuration Identification scheme must be able to uniquely identify the Configuration Items along with their relationships to each other. This is aspect of SCM seems to be identical to the concept that a proscriptive approach should be able to "specify everything about the configuration of a host or network" [12]. In Principles of Network and System Administration, principle 11 is summarized as "One Name for One Object" [10], and it could be argued as further justification for a thorough configuration identification scheme in any System Configuration plan.

Configuration Control

Configuration control is the aspect of SCM that creates the procedures to make changes to a project baseline. More specifically, these procedures cover who can make alterations, how and when alterations can be made to a configuration item. For example, alterations to a configuration item may require permission from a change control board at a later stage of development. At an earlier stage of development a developer might have more freedom in making changes. [08][06]

Meanwhile, in the field of System Configuration, the concept of configuration control could be directly applicable to some configuration files and most, if not all, libraries and binary executables. The most direct application of configuration control would be to use an SCM tool to control most the aspects of an operating system. Certain directories, such as a temporary working directory and users' home directories are probably poor candidate for Configuration Control. In addition some files, such as log files which messages from programs, and process identification numbers should also be excluded from change management. This could be compared to the user generated output from using an application that is unlikely to be placed under change management in a software development project.

Configuration Status Accounting

Configuration Status Accounting is the aspect of SCM responsible for the procedures to gather information required for all the aspects of SCM. According to IEEE standard 828-1998, a plan under this aspect of SCM should include information on what should be monitored and documented for baselines and changes. It should also specify what type of reports and the frequency of these reports. The method of collecting, storing, processing, reporting and last but not least access control to the status data should also be a part of the plan. [08]

Meanwhile in the field of System Configuration, it could be argued that efforts such as cfenvd in Cfengine are an attempt at automating the collection of data for a form of Configuration Status Accounting. Programs such as MRTG, Snort, and Tripwire are also programs that gather information about the status of the configuration items that may be of interest to a System Configuration plan.

Subsequently, it interesting to note that changes in a system governed by SCM are almost exclusively made by programmers or people involved with the project. Unfortunately, for System Configuration the users can intentionally or unintentionally alter the state of the components or configuration items that a system administrator would like to manage. It is also possible that external influences such as automated worms or hackers could alter the state of a configuration item. In this way, status accounting may actually be much more vital to System Configuration than it is in SCM as this information will be vital to an ongoing audit and review.

Audit and Review

Audit and Review is the aspect of SCM that covers the examination of configuration items. It could be described as the validation plan to make sure that the release matches the requirements. In this process, deviations from the requirements or defects should be detected and recorded along with the necessary corrective actions. [08]

In System Configuration, the concept of convergence could be argued to be a form of automated Audit and Review, and this is a concept that will be revisit in a later chapter. In addition, certifying machines for different purposes or Year 2000 readiness are examples of procedures in System Configuration that resemble the Audit and Review procedures in SCM.

Conclusion

To conclude this chapter, in [12] and in [21] Alva Couch mentions a form maturity model for System Configuration. In the proposed system for System Configuration, the four levels are:

1	Ad-Hoc	No system configuration.
2	Documented	Some documentation to record system administration activities.
3	Reproducibility	The ability to replace parts of the system as required.
4	Interchangeability	The ability to replace personnel without the loss of vital information.

In many ways, this is not very different from the five levels specified by the Software Engineering Institute in the Capability Maturity Model [22]. Consequently, in observing how much similarity exists behind the Capability Maturity Models proposed for software engineering and System Configuration, it should not be too far fetch an argument to make that the rational for these models are also similar. The advantages of maturing as an organization should also be similar.

In conclusion, SCM is a vital to a maturing software organization. In this way, it is probably not too much to argue that for similar reasons System Configuration is vital to a system administration departments to mature as an organization.

System Configuration

“Many people seem to believe that the choice of tool determines ease of configuration management. In fact, it’s the practice of using the tools that determines how well the tool works. Choice of tool makes little difference; discipline of use is everything.”
-Alva Couch [12]

In this chapter, we begin by examining the scope of System Configuration. This will be followed by an examination of several System Configuration tools. Having examined several tools which have taken different approaches to the System Configuration problem, it creates a context from which it is possible to discuss several key System Configuration concepts.

The System Configuration Problem

The system configuration problem has been described in many different presentations and papers. In a presentation by Paul Anderson in 2002 several key concepts are used to summarize the automated configuration problem. They are “Handling scale, handling diversity, handling change, supporting devolved management, providing explicit representation, providing high-level models, providing consistency and security.[11]”

In RFC 3139, they talk about configuration management and it presents a list of requirements for an IP based configuration management system for network devices. The following is an attempt to modify it to apply to more generally to the problem of System Configuration and some of the features that a System Configuration tool might provide.

- ✓ A declarative language that specifies the role and relationship of machines between each other at a level of abstraction higher than the level specific to different components or applications at a node level.
- ✓ The ability to translate the higher level declarative descriptions into specific configurations for the different components or applications on each node in the entire system.
- ✓ The ability to add, modify, delete, obtain or restore configuration information for an application or component on a node either in full or in part.
- ✓ The ability to interpret the current configuration, status and monitoring information from each component or application on each node.
- ✓ The ability for nodes to report or provide feedback information to a centralized source so the status of the entire system can be examined in a single place.
- ✓ The ability to provide configuration information to a node, to modify its behavior in response to the situation triggered by changes in the environment or the status of equipment in the system.
- ✓ The efficient means of communicating large amounts of configuration information.
- ✓ A secure means of transferring configuration data. Unlike the RFC, most sites may not require support for access control and authentication of users in different roles and access privileges, but a System Configuration tool should probably have some means of access control,

authentication and integrity-checking to make sure the policy information come from an authorized source and is transmitted correctly.

- ✓ The system should have some means for handling the partial transmission of configuration data, and a method of recovery in the event of failure.
- ✓ The system should be extensible to accommodate future needs.

Three Tools, Three approaches

As mentioned in the introduction to the chapter, three tools were selected to be examined in more details because they represent three approaches. Cfengine is often associated with the concept of convergence. It has been proposed that it operates at a level of files and processes which could be considered the fundamental building blocks of a computer system. ISConf also works on the level of files and processes, but is associated with the concept of Congruence. Meanwhile, the main concept in programs like LSConf, and Bcfg2 appear to be mainly target the configuration of applications. This approach could also be classified as a form of configuration "generator" [15]. Although generators that read profiles could be written for Cfengine in the form of scripts, that would not make it a core function of the program.

Subsequently, it should be pointed out that certain concepts will be introduced in discussing the tools, but their explanation will be deferred to a later part of the chapter. This decision was made because it was felt that this order would establish a context for the discussion of some of the concepts, which is the focus of this thesis. To reverse the order, would place more focus on the tools and references to information that is specific to an approach might be confusing without an appropriate context for a frame of reference.

Cfegine

Cfengine is a program created by Mark Burgess and is mainly associated with the concept of Convergence. The heart of this tool is a program called cfagent. This program reads policy information from configuration files, where the main configuration file is usually called cfagent.conf. This configuration file has syntax similar to a programming language, and can specify additional files as part of the policy to enable an administrator to divide a site policy across several files, so that it does not have to be specified in a single file. This program works with another program called cfservd. The purpose of cfservd is to distribute files to other computers that can request it as a part of their configuration policy. In this way, Cfengine and

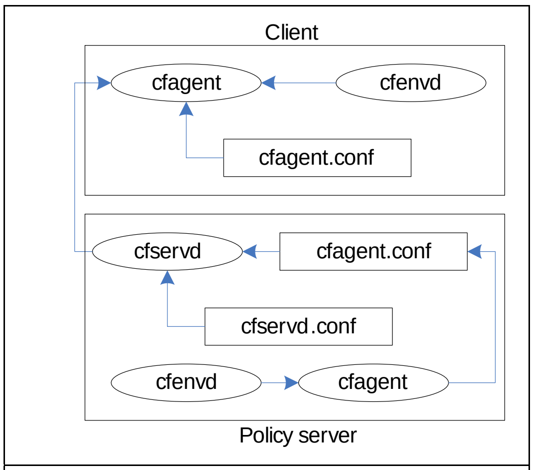

Figure 1. Main elements of Cfengine

cfservd is uses the concept of pulling configuration data from a server by a client. In Figure 1, we can see how some of the main elements might interact with each other.

Having said that, it can be argued that a programming language has more structure and is procedural, therefore the language used by Cfengine is quite different from most modern programming languages. On the other hand, Table 1 shows an example of the language used by Cfengine and in this example is a structure that will be found in Cfengine scripts for cfagent. In the case of this example, it begins with a control section that specifies the order of “control” classes that it should execute. In each “control” class are additional classes that might be defined by cfagent or imported from cfenvd. Under each of these classes are supposed to be declarative statements that describe the intended policy state. In this example, under the “control” class of files, the policy values for the file permission and ownership of two files, on machines that are defined as Linux machines, is specified. A corrective action is also specified.

```
control:
        actionsequence = (files tidy)
files:
        Linux::
        /etc/passwd mode=644 owner=root action=fixall
        /etc/shadow mode=600 owner=root action=fixall
tidy:
        Unix::
        /home pat=core R=0 age=1
```

Table 1. A simple example for cfagent.conf

Subsequently, the core operations in Cfengine are similarly targeted at the level of files and processes, which can also be argued to be the fundamental blocks of a computer system. For this reason, each policy declaration could be interpreted as an operation that take a policy value and attempts to make the system reflects the desired policy state. While these operations might include a great deal of sanity checks and exhibit the property of being convergent, nonetheless they do translate into a series of actions that might be specified from a command line or through a scripting language.

To conclude the discussion of Cfegine, it could be argued that the language used by Cfengine is targeted at the very basic building blocks of a computer system. In this way, knowledge of application and system specific details is most likely required to be able to specify the desired policy state. Furthermore, it might be reasonable to argue that the proximity of the instructions to the basic commands provided by most operating systems makes it more of a procedural language than a declarative language.

ISConf

ISConf version 2 and 4 are written by Steve Traugott, and version 3 is a Perl implementation written by Luke Kanies. As of writing this document, version 4 has not been released to the public and version 2 is the recommended version by the author. To use this set of scripts, a number of assumptions have to met. For example, it appears that a CVS server is required. A file server and a time server is also required to keep all the hosts in synch and to allow files to be mounted on different computers from a common file server. The purpose for having a common time server is to make sure that system time is the same across all the systems. This requirement comes from the use of the make facility to control the sequence of patches or stanzas that will be used to modify a system. The reason for using make stems from the desire that the order of patches should be executed in a deterministic fashion.

LSConf

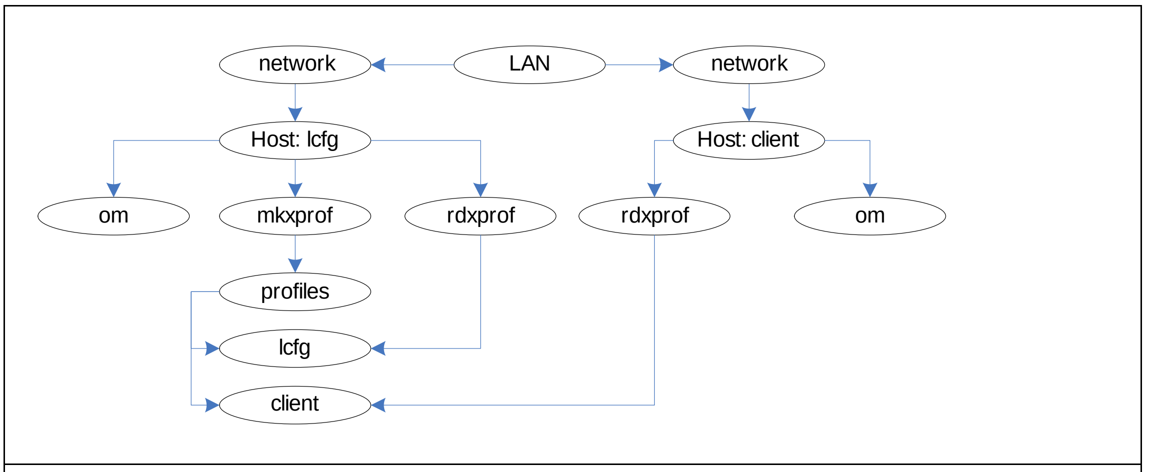

Figure 2. A rough overview of the main components in LCFG.

LCfg is a program written in Perl. It requires several Perl libraries that must be installed in the correct directories so that they can be located by the Perl interpreter. It also requires a web server to publish the Extensible Markup Language (XML) profiles. As of this writing this document, two Linux distributions, Red Hat and Fedora, and Solaris are the targeted operating systems. Other Linux distribution should be able to support LCFG unaltered so long as librpm.so is at least version 4.2. If that is not the case, the perl module interface with the RPM libararies can be commented out of a Perl module to enable the program to work.

The three most important programs in LCFG is mkxprof, rdxprof, om. The first program mkxprof is takes a file describing the desired configuration for a node and turns it into an XML profile. The next program rdxprof takes this XML profile and parses it. The program rdxprof then stores the information gathered from the XML profile in a specially formatted file so that the different LCFG components can fetch configuration information from it. A third program called om (object manager), is used to control the components that make up the host. Last but not least, a program called qxprof can be used to inspect information that rdxprof has parsed and stored.

Key Concepts

In this section, several key concepts of System Configuration will be discussed. Although many of these concepts have been discussed in many venues on System Configuration, an overview of many of these concepts can be found in a presentation by Paul Anderson [11].

Policy

Policy is arguably the most important concept in System Configuration. In fact, policy to System Configuration could be compared to the requirements specification for Software Engineering. In Analytical System Administration, policy is defined as “a description of what is intended and desirable about a system. It includes a set of ad hoc choices, goals, compromises, schedules, definitions and limitation about the system. Where humans are involved, compromises often include psychological considerations, and welfare issues” [09].

Declarative and Procedural Languages

In the field of System Configuration, the configuration of the System Configuration tool may require the use of a configuration file that has a language similar to a programming language. Just as there are many types of software programming languages, two types of System Configuration languages have been proposed.

A declarative language appears to be defined as a language that defines a desired state, and not the means to achieve this state. For example, an LCFG source file consists of several declarations specifying to the settings for different components that make up a node. In Cfengine, the syntax of the cfagent file is considered declarative because it describes the desired policy state without specifying the specific steps needed to achieve it.

Meanwhile, a procedural language appears to be defined as a set of instructions that will be used to implement the changes to a system. For example, a shell scripting language would be considered a procedural language.

Centralized or Decentralized

These key words are sometimes associated with how a policy will be enforced. For example, Cfengine is said to be decentralized because the Cfagent program on each machine is responsible for keeping the machine as close as possible to the policy state. Meanwhile, a centralized system would rely on a server to generate configuration data.

The Push or the Pull of Configuration data

In the field of System Configuration another aspect of discussion is whether or not the client system should pull or a server should push configuration data or software packages. Traditionally, the data is said to be pushed if the transfer is started by the server and pulled if the client requests the data. However, it might be interesting to break this topic up into three questions.

- ✓ Does the server or client check for availability of the other?
- ✓ Does the server or client check to see what needs to be transferred?
- ✓ Does the server or client make the request for specific information to be transferred?

With this in mind, is possible to move beyond the dichotomy of push/pull to examine viable alternatives in an attempt to gain the advantages of both systems without many of the disadvantages. First and foremost, not all systems have a means for checking, either the client or the server, whether or not the transfer of the data is required. In this situation, the options are reduced to that of the traditional push (sender-intent-sender push), pull (receiver-intent-receiver-pull), receiver-intent-based-sender push and sender-intent-based-receiver-pull [20] However, if it is possible for the client or the server to check whether or not the data needs to be transferred than it is possible to talk about another set of hybrid solutions to the question of push versus pull, creating a total of 8 variations to the theme.

To conclude this point, a web browser may be a good example of the classic pull. A multicast broadcast could be an example of the traditional push. Meanwhile, in a paper titled “Extensible, Scalable Monitoring for Clusters of Computers”, Cluster Administration using Relational Databases (CARD) seems to use a receiver-intent-based-sender-push model [19]. A mailing list server and the subscribers on the list are another example of receiver-intent-based-sender-push.

Convergence and Congruence

Convergence and congruence are sometimes seen as competing philosophies. Convergence is property of the tool Cfengine and the basic idea is that an operation should alter the state of the object that is not conforming to policy to a known policy state, or it should fail in a harmless manner. If the object is in the required policy state, the operation should not alter the object [13]. Another important aspect of convergence is the idea of homeostasis where equilibrium is the policy state and this is compared to an immune system [14].

Convergence can also be compared to a person exploring a maze. The final policy state could be the exit to the maze or some specific destination. Should the system or computer deviate from the desired state, convergence is the process by which the system is brought back to the policy state through a series of operations or moves. With this analogy in mind, figure 2 of [13] is a depiction of this analogy. From this figure, Mark Burgess show that the number of possible paths back to policy state grows rapidly the further the system gets from policy state. Therefore, he concludes “that it is in the system’s best interests to remain close to the ideal state at all times. If the remedy to a particular large deviation were unknown, the search for a remedy, in state space, would become extremely time-consuming as the magnitude of the problem increased” [13].

Subsequently, to gain a better understanding of Convergence, it might be useful to examine this concept with a Mathematical model. This can be found in Definition 58 of Analytical Network and System Administration [09]. Another important concept related with Convergence is the idea of Commuting Operations as found in Definition 56. When these two concepts are combined, it can be shown mathematically that a series of commuting and convergent operations can be occur in any order and they will always give the same results. For this reason, a great deal of effort is put into making sure that operations in Cfengine are both commuting and convergent. Unfortunately, some operations cannot be made to commute for very simple reasons. Firstly, it should be fairly obvious that a file system has to be mounted before operations on the file system can be performed. It should also be readily apparent that a file must exist before it can be edited.

With this understanding of the nature of Convergence, people who advocate Congruence believe that order is vital. Therefore, the simplest way to keep a system at policy state is for it to mirror a machine that is defined carefully maintained in policy state. By following executing every operation or change in precisely the same order, it is easier to guarantee that the end result is exactly the same. An advantage of this approach is that the operations do not have to be commuting or convergent. A major disadvantage to this approach is that machines have to start in a known state. This may require rebuilding a system from scratch. Furthermore, should a host deviate from the policy state due to an updates that failed in an unexpected manner, unauthorized changes or unintended delayed side effect from a change, the host might once again have to be reinstall

SCM and System Configuration

When a problem or failure occurs within a computing environment, the first question an experienced system administrator will ask is: *What changed?* Was something added? Was something modified? Is someone trying to use the system in a way different from the defined use? Has something in the computing system's environment changed? Was there a recent power outage? Is there a problem with the environmental controls? This course of investigation – figuring out what *change* has occurred – will often lead the system administrator to the cause of the failure.
- Sally J. Howden and Frank B. Northrup [27]

In the chapter on SCM, the similarity between System Configuration and SCM was established. In the previous chapter, important System Configuration concepts were covered. In this chapter, the advantages and disadvantages that come with the differences between SCM and System Configuration will be explored. With some understanding of the similarities and differences, this chapter will turn its attention to examine some of the areas of SCM that could be applied to System Configuration.

Differences between SCM and System Configuration

The major difference between SCM and System Configuration can be found in the nature and quantity of the components or devices that need to be managed. Another area of difference can be found in the nature of the configuration files. Another difference can be found in the duration of the maintenance activity and the fact that a system administrator may be relied upon to ensure the proper functioning of software long after development has ceased on a project.

Firstly, a system administrator at a large installation not only has to deal with a large number of components, but may also have to deal with a large number of diverse computers with different types of operating systems and flavors of the same operating system. A development team may not have to deal with this problem as this could be a design decision as to whether they will support various platforms. Even if they do have to deal with various platforms, they are not usually responsible for ensuring that the program is installed and configured correctly on all the different computers for their users. On the other hand, a system administrator might be responsible for a large number of computers, where there are a large set of computers that are identical, but where a few to all of the computers may have small to large variations in various combinations.

Another difference between SCM and System Configuration is that most components, applications or devices are purchased, licensed or downloaded from somewhere or another organization. System administrators might write scripts or even programs to automate the job, but a majority of all the components and devices that they manage are not under their control at the source code level. Even open source programs are rarely truly accessible to a majority of system administrator for making serious modifications. The reason could very from the lack of time to the lack of knowledge. On the other hand, programmers are often called upon to create new components or modify existing components.

Moving on, some areas of System Administration are easier to define than a software project. The argument for this position is largely based on the fact that some aspects of system administration can be reduced to the question of whether or not it works. When a user visits a web site or tries to access a database through a search engine or query tool, the system administrator is responsible to make sure that the user gets a response. The response may not necessarily be correct, but more often than not, the problem is with the data or the software, and not the more easily defined system. The reason a system is more easily defined is because there is less human fuzziness in installing binaries, data files or libraries in the correct places. There is less fuzziness in the restricted world of configuration files where the syntax is often predetermined by the application reading that file and the task can be reduced to a yes or no question. For example, does the web server respond to requests on port 666. SCM on the other hand, often has to deal with the requirement form users that may have a great deal of qualitative or arbitrary elements.

Application of SCM in System Configuration

The first application that might be proposed is the concept of a System Configuration Plan. In this plan, procedures and documentation used in SCM could be translated into procedures for System Configuration. In fact, a paper presented at the LISA conference by Sally J. Howden and Frank B. Northrup [27] follows many of the concepts in that would be documented or created in an SCM plan.

System Configuration Plans

By using the article by Sally J. Howden and Farank B. Northup as a start, the first step would be to establish and document the system's baseline. In an ideal situation, all the files associated with the operating system and applications on a computer would be uniquely identified and documented. The relationships between all the documented objects would also be established to eliminate any hidden dependencies. With a proper identification scheme, all the objects should be uniquely identifiable and there should not be ambiguity caused by two different objects having the same name. Since a large site may have a large number of computers, automation will be necessary to gather all this information and a database is probably the most efficient manner to store this information as Jon Finke had learned from experience [28]. Applications like Sowhat [16] can also help to make automate the gathering of this information.

Subsequently, while a SCM project might require a scheme for promoting a configuration item from version to another, the version numbers for System Configuration should come from the configuration item itself. For example, dynamically linked libraries and most binary executables come with version numbers, and so those numbers should be used.

After having defined the procedures need to collect the information, the next aspect to document and test intentional changes could be called Change Management. The first part of change management is a procedure to verify a request for change and to a way determine whether or not the change will be approved. To expedite changes that occur often, some changes could be granted on condition that the changes are documented and carried out according to a tested procedure that had been previously established. The next part of change management is the procedures needed to test and verify that the changes did not violate policy and that they met the requirements without negative side-effects. If the changes should fail, a procedure for governing fault recovery is also needed.

With the two set of procedures above, another set of procedures are needed to monitor the system to ensure that system does not stray too far from the policy state. In a large installation of computers, these procedures will probably have to be supplemented with automation to make it easier for a system administrator to find problems that the automation could not fix on its own. To determine these procedures concepts from the configuration status accounting and Audit and Review from SCM could be applicable.

In conclusion, just as the formation of a SCM plan is good practice in Software Engineering, the formation of a System Configuration Plan could also be argued to be a good practice in System Administration.

Conclusions

"I have mixed emotions on the topic of configuration management. On the one hand I am amazed at the amount of published material on the topic (it's more than I expected); on the other hand, I am disappointed that a topic which is considered essential in hardware engineering and has been well defined in the software engineering for 25 years, is largely ignored in practice. As the practice of software engineering becomes more mature (scientific?), this is bound to change."
-Rudy Bezelman [04]

In writing this document, several conclusions might be drawn. First of all, these procedures may come across as overkill for a small site with a limited number of computers. Nonetheless, all the procedures created in a System Configuration plan or a SCM plan have a purpose, and while they might be less formal in a small organization, nonetheless contribute to the smooth operation and expansion of a site or a software project.

To assist in reducing the tedium in maintain these documents, automation in documentation is useful. Meanwhile, it should be noted that a System Configuration tool can not only automate changes made on a system, but it can also be used to automatically register the changes made according to site policy.

Another observation is that programmers write software that system administrators must manage. Therefore, it would be in the best interest of the field of system administration to show programmers why they should consider the problem of System Configuration as it affects the usability of the software that they produce. For example, when releasing software packages such as RPMs or Debian packages, it might be useful if all the developers could refer to libraries or other software packages in an uniform method. This form of standardization would reduce some of the problems observed in the paper on validation drift [18].

Last but not least, it could be argued that many of the problems in System Administration and Software development are not technical in nature. There are disciplined procedures that and tools to aid in the practice of these processes that successful site follow and improve upon. Failure to take a disciplined approach to solving to managing the system, be it a software development project or a network of computers, could be argued to the main source of trouble.

References

[01] Bersoff, Edward H., Henderson, Vilas D., Siegel, Stan G. (1978). Sotfware Configuration Management. Proceedings of the software quality assurance workshop on Functional and performance issues. pg. 9-17.

[02] Dart, Susan. (1991). Concepts in configuration management systems. Proceedings of the 3rd international workshop on Software configuration management. pg. 1-18.

[03] Conradi, Reidar and Westfechtel, Bernhard. (1998). Version models for software configuration management. ACM Computer Surveys. 30(2), pg. 232-282.

[04] Bazelmans, Rudy. (1985). Evolution of configuration management. SIGSOFT Software Engineering Notes. 10(5), pg. 37-46.

[05] Sanchez, L, McCloghrie, K. and Saperia, J. (2001) Requirements for Configuration Management of IP-based Networks. RFC 3139. Retrieved from http://rfc.net/rfc3139.html on May 20, 2005.

[06] IEEE Guide to Software Configuration Management. (1987) IEEE/ANSI Standard 1042-1987.

[07] IEEE Glossary of Software Engineering Terminology (1983) IEEE/ANSI Standard 729-1983.

[08] IEEE Guide to Software Configuration Management Plans (1998) IEEE/ANSI Standard 828-1988.

[09] Burgess, Mark. (2005). Analytical System Administration. Weinheim, Germany: John Wiley & Son, Ltd.

[10] Burgess, Mark. (2004). Principles of Network and System Administration. Weinheim, Germany: John Wiley & Son, Ltd.

[11] Anderson, Paul. (2002). Automatic System Configuration: Tasks, Principles and Techniques. Retrieved from http://homepages.informatics.ed.ac.uk/group/lssconf/config2002/slides/anderson.pdf.

[12] Anderson, Paul and Couch, Alva. (2004). What is This Thing Called "System Configuration". Retrieved from http://www.sagecertification.org/events/lisa04/tech/talks/couch.pdf on 20th May, 2005.

[13] Burgess, M. (2000). Theoretical System Administration. Proceedings of the Fourteenth Systems Administration Conference (LISA XIV). Usenix, Berkeley, CA. pg. 1-13.

[14] Burgess, M. (1998). Computer Immunology. Proceedings of the Seventh Systems Administration Conference (LISA VII), Usenix, Boston, MA, pg. 283-298.

[15] Couch, Alva. (2005, March 04). [lssconf-discuss] What language choices are there? Message posted to Large Scale System Configuration (lssconf), archived at http://lists.inf.ed.ac.uk/pipermail/lssconf-discuss/2005-March/000531.html.

[16] Sun, Yizhan and Couch, Alva. (2001). Global Impact Analysis of Dynamic Library Dependencies. Proceedings of the Fifteenth Systems Administration Conference (LISA XV). Usenix. San Diego, CA. pg. 145-149

[17] Bottomley, James E.J. and Clements, Paul. (2001). Managing Distributions from the Software Vendor's Perspective. Proceedings of the Fifth Annual Linux Showcase & Conference. Oakland, CA. pg. 119-126.

[18] Hart, John and D'Amelia, Jeffrey (2002). An Analysis of RPM Validation Drift. Proceedings of the Sixteenth Systems Administration Conference (LISA XVI) Usenix. Berkeley, CA. pg. 155-166.

[19] Anderson, Eric and Patterson, Dave. (1997). Extensible, Scalable Monitoring for Clusters of Computers. Proceedings of the Eleventh Systems Administration Conference (LISA XI). Usenix,

San Diego, CA. pg. 9-16.

[20] Duan, Zhenhai, Dong, Yingfei, and Gopalan, Kartik. (2005). Push vs. Pull: Implications of Protocol Design on Controlling Unwanted Traffic. Steps to Reducing Unwanted Traffic on the Internet Workshop. Cambride, MA. http://www.cs.fsu.edu/~duan/publications/sruti05_duan.pdf.

[21] Couch Alva. (2004). Why people don't adopt configuration management tools. Retrieved from http://homepages.informatics.ed.ac.uk/group/lssconf/config2004/slides/alva/workshop.pdf on May 20, 2005

[22] Paulk, Mark C, Curtis, Bill, Chrissis, Mary Beth, and Weber, Charles V. (1993). Capability Maturity Model for Software, Version 1.1. Retrieved from http://www.sei.cmu.edu/pub/documents/93.reports/pdf/tr24.93.pdf on May 20, 2005.

[23] Banga, Gaurav. Auto-diagnosis of field problems in an appliance operating system..

[24] Anderson, Paul. (2005). The Complete Guide to LCFG. Retrieved from http://www.lcfg.org/doc/guide.pdf on 20th May 2005.

[25] Traugott, Steve (1998). Bootstrapping an Infrastructure. Proceedings of the Tenth Systems Administration Conference (LISA X). Usenix, Boston, MA. pg. 181-196.

[26] Traugott, Steve (2002). Why Order Matters: Turing Equivalence in Automated System Administration. Proceedings of the Sixteenth Systems Administration Conference (LISA XVI) Usenix. Berkeley, CA. pg. 100-120.

[27] Howden, Sally J. and Northup, Frank B. How to Control and Manage Change in a Commercial Data Center Without Losing Your Mind. Proceedings of the Eleventh Systems Administration Conference (LISA XI). Usenix, San Diego, CA. pg. 46-52.

[28] Finke, John. (1997). Automation of Site Configuration Management. Proceedings of the Eleventh Systems Administration Conference (LISA XI). Usenix, San Diego, CA. pg. 155- 168.

Additional Material

[29] Jenkins, Steven. (2004). Case Study Questionnaire Results. Retrieved from http://homepages.informatics.ed.ac.uk/group/lssconf/config2004/studies/results.html.

[30] Burgess, Mark. Distributed Resources Administration Using Cfengine. Retrieved from http://www.iu.hio.no/~mark/papers/paper2.pdf

[31] Burgess, Mark. .A Site Configuration Engine. http://www.iu.hio.no/~mark/papers/paper1.pdf

[32] Wingerd , Laura and Seiwald, Christopher. High-level Best Practices in Software Configuration Management. Retrieved from http://www.perforce.com/perforce/bestpractices.html on 20th May 2005.

Key Terms

Term	Meaning
Application	An application is a software package that might be made up of a collection of files.
Baseline System	A computer that only has the basic software from a install. On top of the operating system, this could include applications that the will be required for the system.
Bcfg	A configuration management tool
Cfengine	A configuration management tool
Cfengine Classes	Attributes defined in configuration data for Cfengine or from environmental data that can be obtained from the operating system or by a program such as cfenvd.
Cloning	A process of duplicating data in a storage device to another storage device. This process can only be used to turn a return a computer to a baseline system.
Configuration Language	A method by which a user expresses setting (prescriptive) or the desired operations (imperative) to a configuration management tool through the text of a configuration file.
Configuration Management	In this context of system administration, configuration management might refer to System Configuration.
Congruent	This is the process by which a computer is made to mirror the policy state maintained on a different machine.
Convergent	This is the process by which a computer is maintained near or at policy state by the means of operations that will attempt to fix any deviation from policy state. If the policy state has already been achieved, these operations are not supposed to take any action.
Implementation Language	The programming language that is used to create the configuration management tool.
Initial Installation	The process of transferring the operating system to some form of storage device accessible by the computer. Additional software may be transferred to a storage device accessible by the computer.
ISConf	A configuration management tool.
LCFG	A configuration management tool.
Operating System	A program fundamental to a computer that enables a user or an application to control or configure the hardware that makes up the computer.
Policy	A document to describe the desired state of a system. This is comparable to the requirements specification in Software Engineering.