

Technologies for Large-Scale Configuration Management

Paul Anderson <dcspaul@inf.ed.ac.uk>
George Beckett <g.beckett@epcc.ed.ac.uk>
Kostas Kavoussanakis <k.kavoussanakis@epcc.ed.ac.uk>
Guillaume Mecheneau <guillaume.mecheneau@hp.com>
Peter Toft <peter.toft@hp.com>

Abstract

This report examines the current state-of-the-art in large-scale system configuration. It introduces the scope and principles of the configuration task, and describes a selected range of technologies that are representative of the approaches in use today. It concludes by summarising the dominant approaches to system configuration offered by these technologies.

Acknowledgements

The report is a deliverable of the *GridWeaver* project, a collaboration between the University of Edinburgh School of Informatics, HP Labs in Bristol, and EPCC (the Edinburgh Parallel Computing Centre).

GridWeaver is part of the UK e-Science Core Programme, and we gratefully acknowledge the programme's assistance with this activity. We also wish to thank the other members of the GridWeaver team – Carwyn Edwards, Patrick Goldsack, John Hawkins – for their contributions to the project as well as Alexander Holt for his valuable input.

For administrative purposes, GridWeaver is also known under the project name *HP-FabMan*.



Revision 1.0 – December 9, 2002

Contents

1	Introduction	7
1.1	Motivation	7
1.2	Building on Existing Technologies	8
1.2.1	LCFG	8
1.2.2	SmartFrog	9
1.2.3	Bringing things together	9
1.3	Structure and Objectives of this Report	10
2	System Configuration	11
2.1	The Configuration Task	11
2.1.1	Pre-install tasks	11
2.1.2	Software installation	11
2.1.3	Initial configuration	12
2.1.4	Configuration change	12
2.1.5	Feedback and fault recovery	12
2.2	Automated Configuration	12
2.2.1	Handling scale	12
2.2.2	Handling diversity	13
2.2.3	Handling change	13
2.2.4	Devolved management	13
2.2.5	Explicit specifications	13
2.2.6	High level representation	13
2.2.7	Consistency	14
2.2.8	Monitoring and fault recovery	14
2.2.9	Security	14
2.2.10	Disconnected operation	14
2.2.11	Modularity	15

2.3	Design Approaches	15
2.3.1	Static vs. dynamic	15
2.3.2	Cloning vs. scripting	15
2.3.3	Declarative vs. procedural	15
2.3.4	Abstract vs. concrete	15
2.3.5	Central vs. distributed control	16
2.3.6	Synchronous vs. asynchronous	16
2.3.7	Feedback	16
3	Configuration Technologies	17
3.1	Cfengine	18
3.1.1	Configuration tasks	18
3.1.2	Automated configuration	18
3.1.3	Design	19
3.1.4	Summary	19
3.2	EDG Fabric Management	20
3.2.1	Configuration tasks	20
3.2.2	Automated configuration	20
3.2.3	Design	20
3.2.4	Summary	21
3.3	HP Utility Data Center	22
3.3.1	Configuration tasks	22
3.3.2	Automated configuration	22
3.3.3	Design	22
3.3.4	Summary	23
3.4	LCFG	24
3.4.1	Configuration tasks	24
3.4.2	Automated configuration	25
3.4.3	Design	25
3.4.4	Summary	25
3.5	Microsoft System Management Server	26
3.5.1	Configuration tasks	26
3.5.2	Automated configuration	26
3.5.3	Design	27

3.5.4	Summary	27
3.6	NPACI Rocks	28
3.6.1	Configuration tasks	28
3.6.2	Automated configuration	28
3.6.3	Design	29
3.6.4	Summary	29
3.7	NetBoot and Network Install	30
3.8	Red Hat Kickstart	31
3.9	Rembo	32
3.10	Solaris JumpStart	33
3.11	SmartFrog	34
3.11.1	Configuration tasks	34
3.11.2	Automated configuration	35
3.11.3	Design	35
3.11.4	Summary	36
3.12	System Imager	37
3.13	Systracker	38
3.14	Tivoli Configuration and Operations Manager	39
3.14.1	Configuration tasks	39
3.14.2	Automated configuration	40
3.14.3	Design	40
3.14.4	Summary	41
3.15	ZENworks for Desktops	42
3.15.1	Configuration tasks	42
3.15.2	Automated configuration	42
3.15.3	Design	43
3.15.4	Summary	43
3.16	Other technologies	45
4	Conclusions	47
4.1	Summary of the Technologies	47
4.1.1	Basic operations: low-level installation and configuration	47
4.1.2	Consolidation: dealing with multiple nodes as one	48
4.1.3	Constant change: managing changing configurations	48

Contents

4.1.4	Doing it all: integrated packages	49
4.1.5	Virtualisation: flexible infrastructure provisioning	49
4.1.6	Automated, distributed applications: handling complex dependencies across nodes	50

Glossary

Chapter 1

Introduction

This report is the first in a series of deliverables from the *GridWeaver* project [gri02], a collaboration between Edinburgh University School of Informatics, HP Labs in Bristol, and EPCC (Edinburgh Parallel Computing Centre), under the auspices of the UK e-Science Core Programme. GridWeaver is focused on the challenges of managing the configuration of very large-scale computational infrastructures, especially those that form parts of computing Grids. Our goal is to develop approaches and technologies that are capable of representing and realising complex system configurations of hardware, software and services, across very large-scale infrastructures incorporating high degrees of heterogeneity, dynamism and devolved control.

1.1 Motivation

The emphasis in Grid computing research has been predominantly on services. Researchers, hardware manufacturers and end-users alike dream of the new scientific and commercial world where resources will be readily available and service provision will be regulated and guaranteed. It is not surprising that most of the high-profile, Grid-related work focuses on services and middleware.

In the GridWeaver project we question the ability of the current infrastructure to respond to the demands of the Grid. We don't doubt the vision, we don't underestimate the availability of computing power. What interests us is the glue that keeps the resources together and makes them available to the middleware which is currently under development. What we call *the fabric*.

Large installations are not unusual in the modern computing world. The methods of managing them vary widely across organisations, and it's probably fair to say that most depend on a locally generated process, accompanied lots of heroics from staff.

Over the years, tools have been developed to aid with the management task. These tools were traditionally developed by operating system vendors, which inherently contains shortcomings: the tools targeted the vendors' own architectures and dealt mainly with the task of installing only the operating system. While useful, these tools were not ideal for heterogeneous clusters, for clusters with significant application installation needs, or

for clusters with fast-changing requirements.

The reaction to these unmet needs came mostly from site managers, who fostered the development of home-grown solutions. In addition to supporting various operating systems, these tools also reached as far as deploying software and executing configuration scripts remotely. The problem with these tools was – and still is – that they are hard to maintain, as they are developed piecemeal, and based on isolated designs customized for a specific environment. They also fail to benefit from the experiences and successes of other organisations.

All of the issues mentioned above with respect to large installations are present in the Grid world. They can be tackled with the current approaches and knowledge to a some extent. What the Grid adds to the equation is the need for interoperability. The Grid middleware crosses organisational boundaries, and coordination of fabric management approaches will be difficult with the current generation tools. So, in addition to tackling scale, complexity and dynamism, we will have to address federation of management: independent sites will have to provide services (perhaps through groups of resources) reliably, and the fabric will somehow have to be configured correctly and in a distributed and secure fashion.

The latest work to accommodate these needs raises the level of abstraction of resources, allowing one to describe resources based on the roles they perform (e.g., a file-server). This role-based modelling also lends itself to expressing relationships between resources, a combination that enables customisation through the use of high-level primitives. One can specify resources to be subjected to a configuration change *en masse*, or select groups and subgroups based on the roles and relationships of these resources. For example, one should be able to select all the database servers running a specific version of a DBMS and apply a configuration change to them, then possibly refine this to a smaller subgroup including only those running a specific OS. It is this approach that we speculate can be extended to accommodate federated management.

1.2 Building on Existing Technologies

The partners in the GridWeaver project have developed technologies which are highly relevant to the large-scale configuration problem: *LCFG* from the University of Edinburgh, and *SmartFrog* from HP Labs.

1.2.1 LCFG

The School of Informatics at the University of Edinburgh has a very strong track record in the development, real-world deployment, and management of large-scale workstation networks. The results of this research are implemented as a production system on the School's computing network, and described in several publications. The network acts as a test-bed for the research, as well as a production facility.

One major area of research has been system configuration. The practical results of this are implemented in a toolkit known as *LCFG* (Local Configuration System). As well as

forming the configuration infrastructure for the School, LCFG is currently being used on the test-beds for the European DataGRID [dat].

LCFG started its life as a home-grown package. It does not have a high-level language associated with it, though it has the potential to be driven by one. The toolkit is reviewed in section 3.4 of the report.

1.2.2 SmartFrog

HP Labs' SmartFrog ("Smart Framework for Object Groups") framework arose from our research into *Computing Utilities*—a model for computation based on the notion that computing power can be provided on-demand just like conventional utilities such as electricity, water and gas.

The concepts of the computing utility and the Grid have much in common particularly with respect to fabric management. Both must address questions such as:

- How do users of the computing utility describe the exact configuration of resources and applications that they require the utility to deliver?
- How does the computing utility take the description of a desired system configuration, and automatically create a running system that the user can then make use of?
- How do we secure one client's computation and data from another's inside the utility? How do we secure them from the operators of the utility itself?
- Once a system configuration is deployed, how do we ensure its reliability?

SmartFrog is a framework for the development of configuration-driven systems, targeted at addressing some of the problems above. It was originally designed as a framework for building and managing large, distributed monitoring systems where flexible, but correct, configurations are essential, and indeed the technology in an earlier form is present in a number of complex network management products.

The framework defines systems and sub-systems as collections of software components with certain properties. It provides mechanisms for describing these component collections, deploying and instantiating them and then managing them during their entire lifecycle. The technology, reviewed in section 3.11, is therefore an expression of the resource-relationship model.

1.2.3 Bringing things together

To configure the fabric, first we must be able to describe the precise specification, or configuration, of the system we want to realise. The configuration might include everything from the low-level computation, storage and connectivity requirements of the system, through the supporting applications and services that must be present, up to the actual functionality of the system itself.

Secondly, the fabric must be able to interpret each system configuration, and deploy, instantiate and manage the system through its lifecycle. This must be done securely, and with defined reliability characteristics.

LCFG and SmartFrog have complementary views on the configuration problem. LCFG provides technology for configuration of specific resources, SmartFrog offers a high-level language for expressing complex configurations and relationships, and a deployment infrastructure for setting up applications that run across multiple nodes. GridWeaver aims to bring the technologies together, as a test-bed for exploring large-scale configuration research problems.

1.3 Structure and Objectives of this Report

Our initial report focuses on surveying the current state-of-the-art in system configuration, including examining LCFG and SmartFrog. Our purpose in doing so is twofold: first, to establish a baseline for our research by understanding the approaches, technologies and best-practices that are available today. Secondly, to capture this knowledge in a form that is useful to others wanting a general review paper to introduce them to the area.

The report is structured as follows:

- Chapter 2 – “System Configuration” – defines the scope and meaning of system configuration as addressed by the GridWeaver project. It introduces the motivation for automated system configuration, and identifies some of the key principles and concepts that underlie it.
- Chapter 3 – “Configuration Technologies” – provides an overview of a range of system configuration technologies. While clearly not an exhaustive selection of technologies, we have chosen a range of that is representative of configuration approaches in use today. Each technology is considered individually, and described briefly.
- Chapter 4 – “Conclusions” – summarises and integrates the dominant approaches we encountered in our technology review.

We also include a glossary of common system configuration terms.

Finally, we acknowledge that the work in this report builds on a number of other studies that have reported on technologies and approaches in use for large-scale configuration, e.g., the technology evaluation deliverable from the European Data Grid fabric configuration work-package [Bar01], and Remy Evard’s influential survey of Unix system configuration [Eva97].

Chapter 2

System Configuration

The term “configuration” is widely used with different meanings in different contexts. This section attempts to define the scope of the configuration process as addressed in this project. The motivation for automated configuration tools is then discussed, and some important requirements are identified. Finally, we describe a number of alternative design approaches that have been used in practical configuration tools; these have been distilled from an investigation of existing solutions, and are useful in attempting to classify the various technologies.

2.1 The Configuration Task

We consider the configuration task to encompass everything that is necessary to take a collection of networked computers, with no software, and convert this “bare metal” hardware into a single functioning system, according to a given specification. We also cover the ability to maintain the correspondence between the specification and the actual system, in the event of external changes to the specification, or to the system itself.

This involves the stages listed below. Note, however, that practical tools might not distinguish all these stages; for example, some tools might consider the initial configuration and ongoing configuration maintenance to be slight variations of the same process. Other tools might consider configuration information to be simply the contents of configuration files which are distributed in exactly the same way as the rest of the software.

2.1.1 Pre-install tasks

Pre-install tasks include BIOS configuration, and other low-level operations that are necessary before software can be loaded from the network.

2.1.2 Software installation

Loading of all necessary software, including the operating system.

2.1.3 Initial configuration

Initial configuration involves setting those configuration parameters which differentiate between individual machines with the same software. For example, the network address, or the services that the particular machine is intended to provide.

2.1.4 Configuration change

If the specification is changed, it is necessary to change the installed software, or the configuration, to reflect the changes in the specification.

2.1.5 Feedback and fault recovery

To maintain the correspondence between the system specification and the actual state, it is necessary to monitor the system for changes due to external events (for example, hardware failure), and to modify the system to correct any discrepancy.

2.2 Automated Configuration

The need for automated configuration tools has been obvious ever since the appearance of networked clusters of workstations. This was initially motivated by the inefficiency of manually configuring and maintaining large numbers of machines; in a typical cluster intended as a modern Grid resource, the sheer scale makes such manual configuration impossible. However, manual configuration also has a number of other serious drawbacks; in particular, it is impossible to guarantee the consistency of individual node configurations, or their conformance to any designated specification. This has wide-ranging implications from incorrect computation results, to security vulnerabilities.

The following requirements are important considerations for any automatic configuration tool:

2.2.1 Handling scale

As noted above, the ability to efficiently manage a large number of machines is the primary motivation for automated configuration tools. As the quantity of machines increases, it becomes very important to avoid all manual intervention except in the case of physical hardware failure. Although large scale does present some difficulties, managing large numbers of identical machines, which do not change their configuration, is comparatively straightforward; it is normally the complexity and the other issues listed below which cause the major difficulties.

2.2.2 Handling diversity

If there is a lot of variation between different machines, then dealing with the complexity can be very difficult; this is much more problematic than simple scale. The problem has been acknowledged and addressed in many workstation clusters for some time, although typical Grid clusters often still attempt to impose a degree of uniformity to avoid these difficulties. Avoiding diversity is unlikely to be sustainable as the need to automatically manage more servers, and the complexity of the clusters increases.

2.2.3 Handling change

Likewise, handling rapid configuration change is also very difficult. This too is something which is avoided in many cases by not changing cluster configurations except by relatively infrequent rebuilding. The flexibility to perform dynamic reconfiguration without a complete rebuild is certain to become increasingly important.

A fully automatic tool needs to be able to handle the sequencing of related configuration changes without manual intervention. Changes to the configuration on one machine need to be carefully synchronised with other changes on the same machine, changes on related (server) machines, and with user jobs which might be affected by the change.

2.2.4 Devolved management

In any large installation, many different people will be involved in managing a particular cluster. The management is unlikely to be partitioned on the basis of machines; it is more likely to involve different “aspects” of the configuration. For example, different people will be responsible for the network, the security, the applications, and the web service, across all machines. The configuration of any particular machines will be composed of many different parameters determined by different people. Delegating authority for access to these parameters, and ensuring that they do not conflict is a problem for the configuration tool.

2.2.5 Explicit specifications

Traditionally, the configuration of a machine has been represented by the contents of various configuration files and databases stored on the machine itself. This is a very implicit definition of the configuration specification, and is inaccessible when the machine is down. There is a need for a more explicit representation of the desired configuration, stored externally from the the machine being configured.

2.2.6 High level representation

The level of abstraction represented by the native machine configuration files is usually very low. This prevents reasoning about the system at a level which is meaningful for large and complex clusters. Some representation of the configuration information in a

more abstract form is very useful; for example, to be able to specify the "mail relay" for a cluster of machines, without considering the details of the configuration files used by all the different mail agents.

As clusters become more complex, it is important to be able to represent and manipulate configurations at an even higher level; for example to represent relationships between clients and servers. There has been considerable interest recently in the design of languages to facilitate this approach (see [And01, Gol03, CP02, LFBE02, HP01]). User interface design is also important, but this area has currently received less attention.

2.2.7 Consistency

In many applications it is important that the individual nodes of a cluster are absolutely consistent. In theory, nodes conforming to a given specification will be consistent at the level of the specification, and variations not covered by the specification will not be relevant. In practice, some aspects not covered by the specification may be significant, and their inconsistency across the cluster can introduce unexpected errors (see [TB02]).

2.2.8 Monitoring and fault recovery

As noted in [subsection 2.1.5](#), some degree of monitoring and fault recovery is essential in order to maintain correct configurations in the event of hardware failures and other unexpected events. A minimal requirement is simply to be able to detect errors and deviations from the designated configuration, and to perform a reboot, reinstall, or request a hardware replacement as necessary.

Note that the term "monitoring" often has a wider interpretation covering such areas as application performance, which is outside the scope of this project.

2.2.9 Security

Since an automated configuration system can control and completely reconfigure an entire installation, the effects of security breaches can be dramatic. The number of people likely to be involved in the management, and the need for many tools to run with system privileges leads to some difficult security problems.

2.2.10 Disconnected operation

Devices with intermittent network connections such as laptops and PDAs are likely to form part of any modern installation, both as devices to be managed, and as sources of management control. Increased remote management also increases the possibility of temporary disconnections due to network failure. Support for intermittent connection is difficult to add as an after-thought, and requires consideration at a fundamental design level.

2.2.11 Modularity

As noted above, it is essential for large installations that an automated configuration system can manage machines without any manual configuration effort. This means that every configurable subsystem of the machine must come under the control of the configuration system. It is therefore important that new subsystems can be interfaced quickly and easily to the configuration system, without requiring either complex programming, or disruptive procedures. This requires a modular system with well-defined interfaces, and a simple development framework.

2.3 Design Approaches

Existing configuration tools differ in a number of dimensions, representing different trade-offs among the above requirements. Note that practical tools may use a mixture of approaches; for example, a "static cloning" process for rapid initial installation might be followed by a more flexible approach to maintaining the ongoing configuration.

2.3.1 Static vs. dynamic

Some tools are designed largely to handle the initial installation process (both software and configuration). Such tools cannot deal easily with dynamically changing configurations, nor with fault recovery.

2.3.2 Cloning vs. scripting

Many tools are based on the concept of creating a "golden server" which is then "cloned" to create other identical machines. In practice, some customisation of the clones nearly always occurs, but this process is not capable of handling very diverse configurations.

2.3.3 Declarative vs. procedural

Some tools provide a very explicit, declarative description of the desired configuration which the tool attempts to instantiate on the real hardware. Other tools do not have a clear declarative description, and a procedural specification is used to define the steps needed to configure a machine. Without a declarative specification, it is difficult to reason about the desired configuration, or to create higher-level models.

2.3.4 Abstract vs. concrete

Some tools attempt to abstract configuration parameters to a meaningful level. Other tools deal with low-level platform-specific information; for example entire configuration files.

2.3.5 Central vs. distributed control

Some tools attempt to specify the complete configuration of the installation in a central specification, and to implement this under central control. This has the advantage of supporting a high degree of static validation and control, but it is less scalable than systems involving more peer-to-peer communication (such as service location protocols).

2.3.6 Synchronous vs. asynchronous

A synchronous system will attempt to "push" configuration changes to clients, and wait for confirmation of the changes. This clearly gives tight control, and good error feedback, but is usually impractical in large clusters because some proportion of the clients are likely to be uncontactable at any one time.

An asynchronous system may notify clients of changes, but clients will be expected to "pull" the changes when they are ready. Additional feedback is required to monitor the state of all the clients. In a rapidly evolving environment, it is likely that there will always be some systems in transition between one configuration state and the next. The term "asymptotic configuration" has been coined for this process.

2.3.7 Feedback

Some tools initiate configuration changes in response to a procedural user request, but do not have the ability to monitor the target configuration and initiate self-healing changes.

Chapter 3

Configuration Technologies

In this section we discuss our findings from our investigation of tools and technologies related to the configuration of current fabrics. The selection is not exhaustive, as there are too many relevant packages available to examine them all. We include LCFG and SmartFrog so as to be able to compare them with the other technologies and identify strengths and weaknesses.

The discussions are not exhaustive, but rather focus on the features outlined in [Chapter 2](#). It was also impossible to review the whole of the literature related to each of the technologies discussed, and it was outside the scope of the review to validate the literature claims against running versions of the tools or different pieces of literature from the vendors. Similarly, it was not practical to confirm with any vendors the accuracy of the content prior to issuing the report.

What we can guarantee is that our intention when reviewing the related literature was to single out the pros and cons of each technology and reflect them in the report as accurately as possible. This is the only way for us to satisfy the aim of this report, which is to evaluate the strengths and weaknesses of LCFG and SmartFrog. To this end, we commit to correcting any inadvertent errors appearing in this report and encourage our readers to contact us with any such remarks.

3.1 Cfengine

Cfengine [Bur95, Bur01] is a toolset for the configuration and maintenance of Unix-like (and to some extent WindowsNT-like) operating systems and applications. Cfengine provides high-level language directives for common configuration tasks. Directives are grouped together into sets of actions that describe how a node or set of nodes is to be configured.

Cfengine implements a *class* mechanism that allows similar systems to be grouped together so that policies regarding their configuration can be specified in one place only. Configuration files can be stored locally on a node, or in a central database that is accessed securely by configuration agents running on each node. Local agents connect to the database in order to discover their required configuration.

The philosophy of Cfengine is centred around the notion of creating systems with self-healing properties. If the configuration of a node managed by Cfengine becomes damaged, Cfengine will return it to a correct, well-formed state.

3.1.1 Configuration tasks

The Cfengine directives cover a range of common configuration tasks, such as file operations (linking, setting permissions, ...), editing text files, automatically mounting remote file systems, and so on. Each directive is a high-level, declarative statement of how a particular part of the system should be configured; the directive masks details of how the configuration is to be achieved — such as the specific commands required on each operating system variant, and the details of any conditional testing that is required to effect the configuration.

3.1.2 Automated configuration

Each node that is to be configured is distinguished by an identity (hostname, address, network, ...), by its hardware architecture and operating system variant, by any abstract, user-defined group to which it belongs, and so on. These distinguishing attributes, along with other determinants such as time, are used to map nodes onto (overlapping) configuration classes, each of which has specified configuration policies.

The policy files are held in a central server repository. Agents running locally on each node contact the central repository to construct their specific configuration directives, and then execute these directives. Agents can be configured to poll the repository for updates at specified periods, or can be triggered externally to do so.

In addition to these components, each node can run a passive cfengine agent which gathers information about the node, which can be used to determine the actual configuration state for each node over time.

3.1.3 Design

Cfengine is composed of a central repository server which contains the node policy files and is accessible to every node in the system. Determination of which specific configuration policies to implement is delegated to each node, and mediated by the agent running on each node.

3.1.4 Summary

Cfengine is a mature and widely-deployed technology that provides a pragmatic abstraction layer above the details of low-level system configuration. The class mechanism allows nodes to be managed as groups, alleviating the need to think at the level of individual nodes. The central repository keeps configuration information in one place where it can be managed and kept safe and consistent.

3.2 EDG Fabric Management

The European DataGRID (“EDG”) project (see [dat]) includes a work-package (WP4) to develop fabric management tools for the DataGRID test-beds and production sites. The complete configuration problem, as defined in this project is covered by two of the WP4 tasks: “configuration” and “installation”. Separate “monitoring” and “fault tolerance” tasks also provide some of the required monitoring and feedback.

At the time of writing, EDG test-beds are using a modified version of LCFG [AS02, And94]. The plan is to replace LCFG completely with a new system over the next year. The architecture of the new system is based on the current LCFG architecture, and hence the fundamental capabilities and weaknesses are similar. Some of this system is still at an early design stage, while other aspects, such as the language compiler have prototype implementations.

3.2.1 Configuration tasks

The WP4 technology is intended to implement the full range of configuration tasks, although there is an emphasis on using existing technologies where possible; for example, it is planned to use vendor installation technologies such as Kickstart [Ham] instead of providing a custom installation tool. This has advantages in terms of portability and effort, at the expense of functionality and close integration.

The monitoring and fault-tolerance should have the capability to provide very good feedback and fault-recovery, but it is not yet clear to what extent this will be realised.

3.2.2 Automated configuration

The WP4 technology is clearly intended to address the problems of current large-scale Grid fabrics. It provides a reasonably high-level declarative description of the configuration which is dynamically implemented on the individual machines. The scope however, is slightly more restricted than LCFG; for example there is much less emphasis on support for disconnected operation, or locally initiated configuration changes.

3.2.3 Design

The entire configuration of the fabric is held in a central “database”, expressed in a language called “PAN” [CP02]. PAN is a prototype-based language which is much larger than the LCFG language, and the configuration data are based on a single “global schema”. As with LCFG, the declarative language is compiled into per-machine XML profiles which are transported to the client nodes over HTTP.

Individual components on the client can register interest in certain parameters and will be notified when these parameters change. As with LCFG, the components implement the configuration changes on the local machine. Unlike LCFG, these components manage only configuration change and do not integrate the control of the subsystem process.

3.2.4 Summary

Much of the EDG technology is still at the design stage, and therefore difficult to evaluate. However, it is based on a proven architecture and appears to offer a pragmatic solution to many of the problems currently being faced by Grid fabric managers.

A number of difficult problems have been acknowledged as being beyond the scope of this project, in particular handling the temporal dependency aspects of configuration change, and the interaction with user processes.

3.3 HP Utility Data Center

HP's Utility Data Center¹ (UDC) [udc02] is a commercial product that centres on the automated management of server infrastructure, using the concept of “programmable infrastructure”. Hardware elements such as server nodes, network switches, hardware firewalls and storage appliances are physically wired together into some “fixed” infrastructure configuration. The UDC management software is then able to configure selected combinations of these components into *virtual server farms* using *virtual wiring*. The goal is that once a physical infrastructure is established, it can be reconfigured and re-provisioned many times under software control, allowing for rapid, flexible and repeatable use of resources. This is done primarily through virtualisation of the network fabric and storage subsystems.

3.3.1 Configuration tasks

The primary task of the UDC is to create and manage virtual server farms. A UDC management station, known as a UDC *controller*, uses *farm descriptions* to represent the configuration of virtual server farms. These descriptions include:

- the servers to be allocated
- which operating system images should be loaded on them, and what storage is required for each
- how the servers are connected
- what network tiers are required, and how the firewalls between them should be configured

The descriptions are a declarative statement of the desired resources, their configuration and connection. Descriptions can be created using a graphical “drag and drop” user interface.

3.3.2 Automated configuration

The creation of farms from descriptions is fully automatic. Farms may also have *policies* associated, and can be managed to sustain compliance with these policies. Also, many forms of failure can be automatically handled by patching replacement resources into the virtual farms.

3.3.3 Design

The design of the UDC centres on the concept of *virtual wiring*.

¹The use of the American English spelling for “Center” is deliberate

The UDC Controller configures VLANs that connect the various hardware elements together. Storage elements are soft-connected to each server as required (e.g. by configuring a SCSI to fiber-channel multiplexor). Virtual server farms are secured from one another by virtue of the security properties of VLANs.

The UDC concept accommodates multiple server, networking and storage architectures, from multiple vendors, and supports multiple operating systems. However, hardware must be qualified to in order to fit within the UDC framework, as it must be controllable to some extent by the UDC controller.

Farm descriptions can be saved for future use, and the storage associated with a particular farm can be checkpointed and retained. This means it is very easy to remove a farm, and then reinstate it at a later date in its previous state. Similarly, farm descriptions may be changed while they are in operation, for example to add or remove capacity in response to changing workloads.

3.3.4 Summary

HP's UDC represents a distinctive approach to the management of server infrastructure.

3.4 LCFG

LCFG (for “local configuration system”) is an automatic configuration and installation framework developed by the Division of Informatics of the University of Edinburgh [And94]. It is now also used as part of the test-beds for the European DataGRID project [dat]. LCFG has evolved since 1994 and the following section describes the latest version, LCFGng [AS02]. The LCFG framework provides a means to describe and realize an entire site’s configuration. Its main emphasis is on the declarative specification of configuration and the management and separation of its diverse aspects. A set of declarative descriptions of the target configuration is held in a central repository, and used by individual nodes to get the intended operating system, applications, and general configuration information.

In its current form, LCFG only supports nodes running Linux, but the concepts and mechanisms are extensible to other OS environments.

3.4.1 Configuration tasks

The LCFG technology supports the following configuration tasks:

Individual node installation Each node is associated with an individual file (the “lcfg profile” describing the operating system and the software packages to be installed on it. LCFG is able to use either a floppy disk, a CD-ROM or PXE as means to provide a minimal operating system to bootstrap the installation process. An LCFG component is then installed which treats the further installation of the operating system almost as it would do with any other application. The only difference is a final reboot done on the new operating system leaving the node ready to be managed by LCFG.

Node application management One LCFG component installed on each managed node is responsible for maintaining the software on this node. It is able to consult the list of packages the node should contain, and ask for missing packages. Reversely, it can be notified whenever the configuration it is responsible to maintain has been changed centrally, in which case it downloads the new profile.

Disconnected operations support Nodes do not need to be permanently connected to the central server to be correctly managed. Any disconnected node, for example a laptop, can request for update as soon as it is reconnected to the network. However, scheduled updates, particularly those involving a reboot of the machine, might not always come at a practical time for the users of these nodes. Therefore LCFG allows that type of node to be configured at the user’s convenience.

Diversity support Each configuration aspect, for example the support for a wheel on a mouse, is handled by a particular component installed on the node. It is able to interpret a part of the individual configuration of this node, and it performs the necessary configuration actions. Therefore handling diversity comes down to writing new components able perform the new configuration actions required by a new aspect of the configuration.

3.4.2 Automated configuration

LCFG is intended to configure large farms of possibly heterogeneous machines in an automatic and customizable way. It provides a declarative description of the configuration which is dynamically implemented on the individual machines. The system is designed to be flexible and extensible. Each individual machine configuration may be built out of standard ones, but also gives users the liberty to decide which packages should be managed or not, and even to write new simple components to manage new configuration aspects. A rather unique feature is the strong emphasis put on the loose binding from a node to the central server. It allows for easy support of disconnected operations.

3.4.3 Design

The configuration of the complete fabric is held in a central repository, as a collection of “source files” expressed using the custom prototype-based language. These source files are declarative descriptions of every possible aspect of the fabric configuration. Each individual machine has a source file, possibly referencing other source files. A central compiler can build, for each of the configured nodes, an individual XML file containing all the configuration information that may be needed by a client node. It is sent to the client over HTTP. On the client side, this XML file can be read by components. They are generally simple shell scripts, and perform configuration actions such as creating configuration files specific to a given application, or starting and stopping daemons. The LCFG philosophy is to keep the operations performed by these components as simple as possible, thus making them easy to write and extend.

3.4.4 Summary

LCFG is a working solution that has been proven to scale up to a thousand machines. It provides a simple and quite powerful declarative language to describe configuration of the fabric. It supports disconnected operations as well as possible customization of machines by users. The framework is quite extensible because the emphasis has been put on simplicity for the design of both components and declarative configuration files. The whole process still remains more or less static, in the sense that a centrally held configuration description is enforced onto the managed nodes. The latest version of LCFG has begun experimenting with new facilities, such as fabric-wide queries or handling of dynamic parameters.

3.5 Microsoft System Management Server

System Management Server (SMS) is a commercial application suite developed by Microsoft Corporation. It provides an automated solution to application and desktop configuration management for fabrics consisting of personal computer nodes running 32-bit versions of the Microsoft Windows operating system. This document describes the functionality of Microsoft System Management Server version 2.0.

3.5.1 Configuration tasks

Configuration tasks performed by SMS are:

Operating System installation. SMS allows existing nodes running older releases of Microsoft Windows to be upgraded to either Microsoft Windows 2000 or Microsoft Windows XP. It does not provide facilities for igniting new machines which lack either a suitable operating system or the appropriate SMS modules. Nor does it provide functionality for performing low-level configuration of resources (for example, re-partitioning a hard drive).

Application deployment The notion of application distribution adopted by SMS is very flexible and includes options for deploying commercial and in-house software applications, implementing operating system upgrades, distributing and modifying configuration files or registry entries, and running command line scripts. This provides the machinery for performing dynamic configuration of resources within SMS.

Before deployment, applications must be packaged as Software Distribution Packages. This can easily be done by importing information from other distribution formats (such as InstallShield), creating a package manually (based on a template), or generating a package automatically using a “before and after” snapshot approach (augmented with dependency information collected by monitoring an active copy of the application).

Once created, Software Distribution Packages are transmitted to distribution points within the fabric (called Client Access Points (CAP)). Each CAP then advertises the package to suitable client modules within its jurisdiction.

Inventory discovery SMS can collate actual configuration information from client nodes, providing a highly detailed profile of both the hardware and software that is available within the fabric. SMS uses the Web-based Enterprise Management (WBEM) architecture to describe resources within the fabric. Individual nodes store their own configuration information which is transmitted to an SQL database server on request. Reports and complex queries may then be created on this database.

3.5.2 Automated configuration

A fabric managed by SMS may be organised in a hierarchical manner, with aspects of configuration management being devolved to lower tiers in the hierarchy—this reduces

the workload on the central server. Furthermore, SMS is a very modular technology and different modules of the software may be installed to different nodes in the fabric. These properties all add to the scalability of the technology.

In contrast, SMS does not have an explicit concept of a target configuration for the fabric and individual configuration changes, in the form of Software Distribution Packages, are deployed according to a schedule prescribed by an administrator. The target configuration of a node may only be determined as the sum of all the configuration changes scheduled to be applied to it. This makes determination of the system configuration problematic. Furthermore, since SMS does not provide facilities for importing a bare node into the fabric—we infer that the configuration changes needed to bring a new node into line with the configuration of its peers must be initiated manually by an administrator.

Based on this evidence, we infer that SMS is well-suited to fabrics with a high-level of uniformity (that is, little diversity) for which configuration changes are applied infrequently.

SMS is able to monitor the distribution process of a software application and provide information to the administrator regarding the success or failure of the configuration changes deployed to each target nodes.

Target nodes may be grouped together into collections, allowing configuration changes to be applied to all members of a collection in a single action. This provides a mechanism for establishing the consistency of the individual nodes which form a cluster.

3.5.3 Design

SMS is a dynamic configuration tool which advertises configuration changes that are then pulled down from a distribution point (that is, a CAP) by target clients. These configuration changes occur in an asynchronous manner, and SMS provides feedback regarding the progress in deployment of each configuration change. The configuration of the fabric is implicitly held in a declarative form, though is not easily viewable by a user. The technology offers minimal abstraction in its model.

3.5.4 Summary

SMS is a technology designed to reduce the workload of distributing and managing software applications within a large fabric. We infer that it is best suited to an environment in which node diversity is low and configuration changes are administered infrequently. It provides comprehensive facilities for inventory discovery, though the lack of an explicit target configuration limits the possibilities for automated configuration maintenance. SMS does not provide facilities for igniting bare nodes or automatically bringing nodes into line with the configuration of their peers. The technology is highly modular allowing different aspects of the configuration task to be deployed to different nodes. Furthermore, the hierarchical representation of the fabric implemented by SMS, allows responsibility for different parts of the fabric to be devolved to lower levels in the tier, reducing the workload of the central server.

For more information about SMS, consult the Microsoft web page [[Cor02](#)].

3.6 NPACI Rocks

NPACI Rocks is an open source product developed by the San Diego Supercomputer center [PMP01]. It provides a semi-automated Linux solution to the installation of front-end and back-end nodes for clusters aimed at scientific computation.

3.6.1 Configuration tasks

NPACI Rocks offers two main facilities, individual node installation and customized distribution. Furthermore, a third-party monitoring system is integrated to the Rocks distribution.

Individual node reinstallation This is the core facility of Rocks. A node may be installed or re-installed from a bare computer using PXE. Any request from a re-booted node to a DHCP server is trapped, and the individual node configuration is then pulled from a central configuration database, as a RedHat Kickstart file. The software (including the operating system) is then downloaded from a central server through HTTP.

Building customized distribution The software configuration of individual nodes is built from a tree of dependencies between softwares modules (Linux RPMs). A Kickstart file is built on the fly for each different node, combining generic information (independent from the cluster) and the information on the cluster stored in a central database.

Monitoring Although not strictly part of Rocks itself, a third party monitoring system, Ganglia [gan], is provided. It allows a user to query the state of the cluster and each of its individual nodes.

3.6.2 Automated configuration

Rocks is designed to 'make clusters easy'. Therefore the emphasis is on single node reinstallation and on software updates tracking. Whenever a machine is reported as having failed by the monitoring system, it is reinstalled completely. A central database holds the information regarding the actual role of each of the node (front-end, and back-end for a given front-end), and which generic RedHat Kickstart file should be used for the software distribution. As previously mentioned, the combination of these allows Rocks to build, when the machine boots, a customized Kickstart file taking into account cluster-specific information.

The actual time it takes to rebuild a machine is described in [PMP01]. In the experiment described, a node in a cluster of 32 nodes downloads approximately 150 MB of data from the HTTP server and it takes around 13 minutes to build it. Therefore for such a cluster size the system scales correctly.

3.6.3 Design

The target configuration of the cluster is held in a central database. It is the combination of the roles of the nodes and their software packages. NPACI Rocks enforces this static description on the underlying fabric, by making sure each failed node is reinstalled from the ground up. Changes in the target configuration occur when the generic RedHat Kickstart files are changed manually, for example when a new version of the Linux RedHat distribution is released. In this case, each of the nodes using this changed distribution is reinstalled completely.

3.6.4 Summary

Rocks is a very robust technology designed to automate as much as possible the installation of a cluster aimed at scientific computation. NPACI Rocks addresses two main problems: the normally costly reinstallation or repair of a node that has failed, and the tracking of software updates. The approach is to consider both software to be updated and any failure inside a node as divergences from a target configuration, and the same action is applied in both cases: complete automated node reinstallation. In the first case the configuration change is triggered manually, by an administrator changing the generic Kickstart files and thus starting the reinstallation process. In the second case the change may be manual (an administrator rebooting a node manually) or automatic (failure caught by the monitoring system). This approach is easily manageable, and allows for the construction of a central repository holding the target configuration, synchronously enforced onto the fabric. Experimental cluster with new software distributions may quickly be built, and a roll back to previous software releases made whenever the whole cluster fails for undiagnosed reasons.

It is assumed that the nodes in this cluster will not need to change their own configuration during the computation, nor will the back-end nodes need to interact with anything else than the front-end nodes. As such NPACI Rocks targets a very specific and constrained type of fabric, where an 'all or nothing' approach scales reasonably well, and no real further diagnostic is needed.

3.7 NetBoot and Network Install

NetBoot and Network Install are two utilities developed by Apple Computers Incorporated, and packaged with the Mac OS X Server operating system. The technologies provide complementary solutions to the automatic management of both server and client nodes within the fabric. They only support Apple Computer's hardware running Mac OS 9 and Mac OS X. This description is based on the versions of the tools provided with Mac OS X Server Version 10.2.

NetBoot is a static configuration technology which allows one to boot a client or server node from a pristine installation disk image held on a remote server, using a cloning procedure. This cloning procedure may be augmented by pre-installation and post-installation scripts which allow for localised configuration of individual nodes. NetBoot allows an administrator to lock the configuration of a node, since the pristine installation is restored whenever the node is rebooted.

Network Install allows an operating system, operating system patches, and applications to be pulled onto a client machine from a remote server in the form of a cloning procedure. This again represents a static configuration process.

Images may be rapidly created by an administrator using a GUI environment which provides a convenient interface to Apple Package Manager and Apple Property List Editor. Once created and associated with the appropriate client nodes, no user interaction is required with the images and node configuration is completed automatically (at the next reboot). At boot time, client nodes request the correct images from a server, the location of which is established using a discovery process which is an extension of BootP and DHCP, called Boot Server Discovery Protocol. This protocol also provides necessary information for configuring low-level network properties at the client node.

Delivery of disk images is a "fire and forget" process. No monitoring of the success or failure of the delivery and configuration process is provided, although image requests are logged at the image server. Each image server can hold up to 25 images and while multiple image servers may be defined, the scalability of the technology is limited to fabrics with low amounts of diversity.

We infer that NetBoot and Network Install is a clean and simple technology best suited to small-scale fabrics, with little diversity and requiring only infrequent configuration change. For more information about these technologies, consult [[Inc02b](#), [Inc02c](#)].

3.8 Red Hat Kickstart

Kickstart [Inc02e] is a utility developed and maintained by Red Hat Incorporated, designed to automate the task of igniting/upgrading nodes with Red Hat Linux. The utility is packaged with the Red Hat Linux operating system and thus supports all hardware platforms currently supported by Red Hat Linux—this includes personal computers and a range of workstation hardware platforms. While Kickstart is intended for use with the Red Hat release of Linux, analogous utilities with similar functionality exist for other Linux platforms. For example, Suse provide a utility called AutoYast2 with their current Linux implementation. The description presented below is based on the distribution of Kickstart packaged with Red Hat Linux version 7.3.

Kickstart ignites nodes using the standard Red Hat Linux installation procedure augmented by a declarative specification of the target configuration held in a response file. Pre-installation and post-installation scripts may be added to the procedure to facilitate localised configuration of individual nodes. The response file may be stored on a diskette or, more commonly, on a remote file server (discovered using BOOTP and DHCP). Linux installation images may be read from a range of sources including: NFS, FTP, CD-ROM, HTTP, or a local hard disk drive. The Kickstart ignition procedure is implemented in a “fire and forget” manner and represents a static configuration process. Complete automation of the ignition process is not possible in the reviewed release of Kickstart, since each target node must be booted from a Red Hat Linux Boot Diskette to instantiate the process—this limits the scalability and dynamism of the technology.

The response file is commonly referred to as a “kickstart” file in associated documentation. It serves to eliminate the manual contributions normally required during the installation process. The file is stored as plain text and contains an ordered list of attribute pairs, corresponding to configuration choices normally made during a typical installation—no abstraction of this configuration definition is provided. New response files may be created using templates included with the Linux distribution, by manipulating an existing response file (imported from a node which was installed manually), or with the help a GUI shipped with Kickstart.

Kickstart tasks cover low-level aspects of configuration, including: disk partitioning, network configuration, and boot loader specification. Software applications may also be installed with the help of the Red Hat Package Manager (RPM).

Kickstart is a simple tool that reduces the administrative task of igniting multiple nodes with Red Hat Linux. We believe that the technology is best suited to small scale fabrics with low levels of diversity. Its effectiveness is best exploited when it is incorporated with other functional components into a more general configuration environment.

3.9 Rembo

The Rembo toolkit [SaR01] is a commercial product developed by Rembo Technology. It is a PXE network boot application designed to automate the large-scale, customized deployment of software images onto nodes.

Rembo relies on the presence on a PXE-enabled network interface card in the nodes it manages, and on a DHCP server on the network. When a remote node is powered up and instructed to boot from the network, a DHCP request is broadcast to obtain the boot parameters: to get an IP address, network parameters and some further information on the boot procedure that follows. A secure connection with a Rembo server can then be established and the operations to perform downloaded. Finally, the actual disk image files are downloaded from the server and a script boots the required operating system.

Rembo offers a number of useful features. Utilities are provided to easily “snapshot” disk images from existing nodes and upload them to the server. These images can then be used as a base to clone nodes, making it easy to make a local changes and propagate them to all identical machines. Rembo also optimizes the way it distributes these images. First, it only downloads the difference between the actual content of the remote node’s disk and the target image (a “diff”). Secondly, it multicasts the image on the network thus avoiding some network traffic duplication.

Rembo is OS-independent and can read and use both Windows and Linux filesystems, including allowing modification of files normally locked by an operating system. A C-like scripting language also allows powerful manipulation on these image files, such as updates to kernels or to applications. Finally, nodes may be organized into groups sharing the same startup scripts, and each node can also have its own options. The central Rembo server holds a static representation of the configuration of the software installed on the nodes it manages.

Rembo efficiently handles both scale and rapidity of change in an image, as it is very easy to upload and propagate changes with minimal impact on network traffic. Although limited (by design) to configuring the software image on each node, it does so with great flexibility.

3.10 Solaris JumpStart

JumpStart is a utility developed and maintained by Sun Microsystems Inc. and packaged with the Solaris operating system. It is designed to automate the task of installing/upgrading the Solaris operating system and Sun proprietary software on multiple nodes within a fabric. It supports Solaris (version 7.0 or greater) running on personal computers and Sun proprietary hardware. This description relates to the version of JumpStart packaged with Solaris 8.

JumpStart is a static configuration technology which ignites individual nodes using Solaris installation images stored on a remote server (or possibly, on CD-ROM). The installation process may be augmented by pre-installation and post-installation scripts. This allows Sun proprietary applications to be deployed using Solaris Package Management tools, and facilitates localised configuration of individual nodes.

To prepare a JumpStart environment, an administrator creates two files. The first file is called a “profiles” file and contains collections of responses for all nodal configuration models required for the fabric: these responses provide a declarative specification of the target configuration for a node. The second configuration file is called a “rules” file and contains a list of criteria, sufficient to attribute a configuration description to each node on the fabric. Nodes may be identified using a diverse range of properties, including hardware configuration, previous software installation, network reference number, or IP address. The “rules” file provides a very simple implementation of roles within the fabric. Low-level configuration tasks such as disk partitioning and network configuration are well supported by JumpStart.

Re-configuration of nodes is initiated by an administrator in a “fire and forget” manner—no monitoring of the deployed node ignition is provided.

We infer that JumpStart is a simple and effective tool for deploying and maintaining the Solaris operating system on small scale fabrics with low levels of nodal diversity. It is commonly included as an ignition component in a more comprehensive configuration solution. For more information about JumpStart, refer to [Inc02a].

3.11 SmartFrog

SmartFrog (SF) is a framework for configuration driven systems, developed by the Serano Project at Hewlett Packard Laboratories in Bristol. It is written entirely in Java allowing it to be implemented on any platform/operating system combination which supports a Java Virtual Machine. This description is based on the provisional specification for SmartFrog 2.0 [Gol03].

The SF framework consists of three aspects:-

- a configuration description environment, including a language along with tools for manipulating, composing, and validating instances of configuration descriptions.
- a component model that specifies the various interfaces that a component should implement in order that SF can manage its lifecycle.
- a configuration management system that use instances of configuration description and component interfaces in order to manage the lifecycle of a group of related components.

Within the framework, a system is viewed as a collection of components. These components represent such things as software, data, and hardware. Inter-related components are grouped together into a larger unit called an application. Within an application, components are tightly bound into a parent-child hierarchy. Different components in an application are able to communicate with each other through the SF framework. Different applications are usually regarded as being independent, but may locate each other through a discovery process.

3.11.1 Configuration tasks

Since SF is a framework, not an application in its own right, there is great potential flexibility for devising and implementing a whole host of configuration tasks. However, the focus of the technology is the deployment and management of software systems—especially systems that are complex and/or distributed.

Within a deployed system, applications may be initialised, implemented, and monitored throughout their lifecycle. SF applications are created by plugging together modular components—an extensive range of reusable components are provided with the SF distribution. Procedural aspects of configuration may either be incorporated directly into component descriptions, accessed by reference to other components, or initiated by calling external applications (in which case, a component would act as an interface between the SF management system and the target application).

Low-level configuration tasks are commonly implemented using third-party technologies. For example, one could write a SF component to read information regarding the installation of a new node from a database and then translate this information into a form suitable for Rembo (see Section 3.9). SF would then spawn Rembo—via a wrapper component—and provide the responses needed by Rembo to perform the remote ignition of the node.

In general, a newly ignited node would be loaded with an appropriate SF daemon during installation so that it could be immediately incorporated into the SF management domain. Feedback is provided regarding running applications, and component descriptions may be created to provide fault recovery based on events monitored in the SF management system.

3.11.2 Automated configuration

SF provides the basis on which one can build and deploy a running management system to automatically maintain the configuration of the fabric. By default, managed nodes within the fabric communicate with their peers using RMI (although, this may be replaced by other transport mechanisms). The peer-to-peer nature of these communications introduces no inherent restrictions on the scalability of the technology.

The SF description environment supports a mature syntax that allows high levels of diversity within a fabric to be described with ease. Template component descriptions may be parameterised allowing localised descriptions to be derived with minimal overheads. Furthermore, descriptions may be mutated to give context-specific instances of a template. These properties of the language encourage the use of a high-level representation of configuration aspects.

The SF deployment system supports and manages a diverse and customisable array of workflow patterns. Different applications may function in an autonomous manner, with no inherent synchronisation requirements between them. Such functionality allows fabrics that are subject to high levels of configuration change to be managed with little additional work required on the part of an administrator. Devolution of control may be achieved by replicating template server-client component relationships to provide a hierarchy of management control within the running system.

Compound components packaged with the SF distribution provide transactional guarantees regarding the deployment of their children. Typically, a component will verify that all of its children have started correctly, and otherwise fail “graciously”. However, more general comment regarding the transactional characteristics of deployed configuration changes is not possible, since such characteristics are generally inherited by the system from any slave technologies which provide the procedural input. For example, Rembo being used to ignite new nodes.

With regard to security, all components live within a single trust domain that is designated by the extent of the SF deployment. All components (irrespective of their parent applications) may discover and communicate with each other, through the management system.

3.11.3 Design

Since SF is a framework, some characteristics of the design may be dictated by the structure of a particular deployment of the technology. In such cases, we can only comment on the inferred design preferences adopted by the developers.

SF is a dynamic configuration environment, aimed at scripted configuration. However, configuration deployment based on a static cloning approach may form part of (or the whole of) a particular instance of the environment. The SF model implies a declarative specification of configuration, though some component descriptions may include both declarative and procedural content.

Centralised versus distributed control is a choice to be made on a case by case basis. As is the use of synchronous versus asynchronous configuration deployment.

A key emphasis of SF is on the abstraction of configuration information, in order to provide meaningful description without the distraction of uninteresting, low-level data.

3.11.4 Summary

SF is a framework for configuration systems, capable of maintaining the configuration of large scale, diverse, and dynamic environments. The technology is implemented in Java, allowing it to be employed on any platform which supports a Java Virtual Machine. At the heart of SF is a flexible and extensible modelling environment which facilitates the roll out of configuration solutions and encourages abstraction of the configuration process to a meaningful level.

3.12 System Imager

System Imager [FP01] is a utility, developed by Bald Guy Software, for rapidly configuring nodes using a cloning procedure. It is open source software distributed under the GNU Public License, and runs on any platform with a Linux operating system. The description below is based on System Imager version 2.01.

System Imager is a static configuration tool that captures a snapshot of a golden server and replicates this image to a designated set of target nodes. Deployment is a “fire and forget” process: neither pre-validation of the target node’s suitability or verification of a successful deployment is instigated. In addition to node cloning, System Imager is able to perform low-level tasks such as network configuration and disk partitioning. For fabrics that support the Pre-execution Environment (PXE), the process may be completely automated and provides a method for unattended installation of nodes.

The philosophy of System Imager is very simple and the code is written entirely in Perl. The technology regards the golden server’s configuration as the content and structure of its local file system. Cloning is achieved by copying this content and structure to the designated target nodes (this includes replication of the golden server’s disk partitioning). It follows that, in order for the cloning process to succeed, each node to be configured must have an identical—or, at least compatible—hardware configuration with the golden server. It is the administrator’s responsibility to ensure that operations submitted are appropriate for each prescribed target node. During the cloning process, images are pulled by the target nodes from an image server. Multiple images may be applied to each node and, for efficiency, subsequent to the initial cloning operation, deployment operations only transmit file system changes. By default, the administrator interacts with the technology through a command line interface, though a GUI environment could be added if required.

In summary, System Imager is a simple utility for cloning a large number of identical Linux-based nodes. The simplicity and open source nature of the software allows the utility to be modified according to local requirements or packaged into a more comprehensive configuration solution. However, the lack of validation necessitates that a user exercises caution when deploying configuration images.

3.13 Systracker

Systracker [YSS00] is a tool for assisting with the system administration of both individual systems and clusters. It is designed to track changes that are made to system configuration parameters, so that these changes can be reinstated in the case of, for example, a system failure.

Systracker is configured to monitor a specified set of configuration files, directories and software packages (using RPM). It checks for changes to any of these objects at arbitrary intervals, such as every night. Any changes in text files are stored using RCS (Revision Control System), and only a single copy of any changed software package is retained, in order to save space.

Each set of changes represents a checkpoint that can be saved to a safe place, such as a central repository. The saved checkpoints can be used to roll back the system to any specified state, or to rebuild a system that has crashed by reapplying a complete set of configuration changes.

3.14 Tivoli Configuration and Operations Manager

Tivoli Configuration and Operations Manager (TCOM) is a commercial application suite developed and maintained by IBM which provides a comprehensive set of configuration management tools designed to simplify the administration of large-scale, multi-site enterprises. TCOM supports a heterogeneous fabric and may be deployed on most Unix operating system/hardware configurations as well as personal computers running 32-bit versions of Microsoft Windows and IBM OS/2.

3.14.1 Configuration tasks

The core capabilities of TCOM are:-

Operating system installation and low-level configuration (currently targeted at Windows and OS/2 client nodes). For such nodes, customised OS installation may be automated using a network-based installation approach augmented by a response file manually created in advance².

Application deployment covers a diverse set of configuration tasks within TCOM, including: installation of applications, update of an operating system and applications, distribution of data files, registry updates (where applicable), and distribution of system configuration files. Configuration changes are distributed in the form of a Software Distribution Profile which is pulled by target client nodes from a designated server.

A number of techniques exist for the creation of new profiles. For a commercial application, TCOM can import installation information from common third-party installation tools (such as Microsoft Installer) to create a default profile. Alternatively, a profile can be created manually, by duplicating and modifying an existing profile. Finally, a profile can be created using a “before and after” snapshot approach.

Inventory discovery allows actual configuration information to be gathered from managed resources and compared against the target configuration. Actual configuration information is transmitted by client nodes on request and is collected by local inventory discovery servers. Inconsistencies between the target and actual configuration can then be corrected automatically.

Details regarding actual configuration information are held on a relational database at a designated server and may be mined by users with complex queries.

Failure recovery of deployed configuration changes is implemented by TCOM, which monitors the progress of deployed profiles and instigates corrective action in the event of a failure.

²At the time of writing, OS installation requires a boot diskette to be manually inserted at the target node or nodes. However, in the future, the procedure is likely to utilise PXE in order to automate the ignition of client nodes running Windows, OS/2, and versions of Unix.

3.14.2 Automated configuration

TCOM can effectively automate most aspects of fabric configuration, handling scheduling considerations for change procedures and responding to common failure events.

In order to provide efficient scalability, configuration management services may be distributed across the fabric by manually creating a resource hierarchy and deploying tasks within it. Policy regions may be created within this hierarchy, allowing common aspects of configuration to be shared by a number of nodes. Authority for different aspects of configuration may be delegated to different users or user groups, who are then provided by TCOM with the appropriate privileges.

While significant functionality is provided for implementing higher level configuration change, low-level aspects of nodal configuration, such as operating system ignition, are currently restricted to Microsoft Windows and IBM OS/2 systems. Responsibility for low-level tasks, such as disk partitioning, is delegated to proprietary operating system installation tools.

Disconnected operations are supported and TCOM is aware of the concept of a volatile connection—for such a node, configuration tasks can be scheduled for times when it is connected to the fabric. Facilities are also provided to allow the owner of an individual node to have input as to the scheduling of changes.

3.14.3 Design

The target configuration of the fabric is implicitly held by TCOM in declarative form in a distributed database. This target configuration (and the set of actions required to achieve it) may be browsed and modified by an authorised user in a preview environment. However, the mechanics of the representation are hidden and no distinction between language and model is apparent.

Managed resources may be collected into policy regions, simplifying the task of administering large numbers of similar or identical node configurations. However, only modest levels of abstraction may be applied to a nodal configuration description—this limits the scope for meaningful definition of large scale entities such as clusters, and inhibits the possibility for applying rapid reconfiguration.

The implementation of configuration change is handled by an application called the Configuration Change Manager (CCM), which advertises procedures in order to bring the members of each policy region into line with their target configuration. The delegation, to the CCM, of control over the change procedure coupled with the division of the fabric into policy regions, allows changes to be made dynamically and asynchronously. Once deployed, configuration tasks are monitored by the CCM until completion and corrective action may be instigated in the event of a failure.

3.14.4 Summary

TCOM is a large-scale, multi-site configuration tool, capable of handling diverse hardware and software components. Configuration changes are advertised by a dedicated service (the CCM) and pulled by client nodes in an asynchronous manner. TCOM is capable of: deploying software, igniting a subset of operating systems onto client nodes, performing hardware/software inventories, and responding to common configuration change failures. It stores both the target and actual configuration status in a declarative fashion using a distributed database structure: although no high level representation of configuration status information is apparent. There is limited support for managing clusters of nodes as a single entity, though we infer that the underlying philosophy is to regard the fabric as a collection of individual nodes.

For general information about TCOM, consult [Red99, Cor00]. For specific details of software deployment, see [Red00, Cor01b]. Further information about inventory services can be obtained from [Cor01a].

3.15 ZENworks for Desktops

ZENworks for Desktops (Zfds) is a commercial application suite developed and maintained by Novell Incorporated. It is a desktop management tool which automates the process of deploying, managing, and metering software and hardware components within a fabric. Zfds is targeted at personal computers and is available for 32-bit versions of Microsoft Windows and Novell Netware operating systems. Zfds requires the Novell e-Directory Services application to be installed on the fabric. The description below is based on ZENworks for Desktops version 3.2.

3.15.1 Configuration tasks

Zfds provides the following core capabilities:

Operating system installation and pre-installation tasks. Using the Pre-execution Environment (PXE), Zfds is able to ignite a new node, partition the hard drive appropriately, and install a target operating system in a completely automated manner. Once ignited, specific configuration changes may be distributed to the newly acquired node in order to bring its configuration into line with its peers.

Related atomic units of configuration change—called default policies—are collected into distribution units called Policy Packages which may be deployed dynamically to client nodes.

Software deployment is the principal function of Zfds. The term “software” may refer to commercial applications, in-house software, software patches/updates, collections of data files, or desktop configuration changes. Applications are packaged by an administrator either by migrating instructions from a third-party installation tool (for example, Microsoft Installer), by manually duplicating and modifying an existing package, or by using a “before and after” snapshot approach.

Once an application is deployed, Zfds monitors the state of the installation and is able to instigate corrective action for common problems such as corrupted registry entries. Problems which Zfds is unable to resolve completely are reported to the administrator using e-mail.

Software metering is also supported by Zfds. This allows an administrator to monitor, restrict, or licence the use of application software which is under the control of Zfds.

Inventory Discovery Zfds provides facilities to allow a complete inventory of hardware and software available within the fabric to be created and queried. Each client node holds information relating to the locally installed hardware and software, which is transmitted to the inventory server on request.

3.15.2 Automated configuration

Zfds provides a high level of automation for all aspects of operating system and software deployment and maintenance. Configuration changes—in the form of policy packages—

are pushed to client nodes from the server according to a schedule prescribed by an administrator. To reduce the load on the server, deployment of configuration changes is achieved through a tiered distribution scheme: the master server pushes a single copy of the configuration change onto a second level of distribution servers which then each have responsibility for the deployment of the package onto nodes within their jurisdiction. Zfds also employs techniques to improve the load balance of network traffic during configuration deployment, attempting to ensure that policy packages are delivered to client machines from the nearest available distribution server.

Zfds allows an administrator to lock down a user's desktop, reducing the potential for configuration rot due to unsolicited interference or external changes. In addition, Zfds monitors the health of managed nodes and can automatically correct faults, such as invalid registry entries or corrupted system files.

Zfds provides comprehensive support for disconnected operation, allowing policy packages and applications to be cached on volatile connections for off-line configuration management. Alternatively, configuration changes may be prescribed to occur when a volatile node is next connected to the fabric.

3.15.3 Design

Zfds utilises NeD to store the hierarchical topology of the managed resources which form the fabric in a directory structure. Resources which may be managed with Zfds include both users and nodes, and the structure of the fabric directory may be browsed or amended using a simple GUI environment. Resources within the directory may be loosely linked together into groups allowing collections of similar resources to be configured from a small set of policies, thus improving the scalability of the technology.

Zfds is a dynamic and asynchronous configuration tool which ignites new nodes using a cloning process (for speed and simplicity). Further configuration of nodes is achieved using a scripting process: this involves collecting relevant policies into a package and associating this to a node through NeD. Policy packages provide a minimal level of abstraction to the process.

3.15.4 Summary

The ZENworks for Desktops (Zfds) product suite provides a centralised desktop management tool for personal computers and servers running Microsoft Windows or Novell Netware. The focus of the technology is to efficiently distribute and maintain software applications within a large fabric. Configuration change is performed in a dynamic manner. Zfds is able to lock down client desktops and user accounts, in order to control the usage of software/hardware and reduce the speed of configuration rot.

The philosophy of the technology is to regard the fabric as a collection of individual nodes and no mechanism exists for collecting groups of nodes as a cluster (in order to deploy an application task onto it). Administration of different aspects of the fabric is centralised on a Novell e-Directory (NeD) server, inhibiting possibilities for devolving responsibility for the management of different aspects of the configuration. An administrator interacts with

the target configuration using a graphical user interface with little abstraction of resources and tasks.

Configuration changes are administered asynchronously using a scripting approach, following a schedule prescribed by an administrator. Comprehensive low-level configuration of personal computer nodes is supported and PXE-enabled nodes can be ignited in a completely automated manner using a cloning procedure.

For more information about ZENworks for Desktops, see [Inc02d]. For specific information about preboot services, see [Inc01b]. Details of Novell e-Directory services is provided in [Inc01a].

3.16 Other technologies

There are many examples of configuration technologies that we have not discussed in this report. These technologies have characteristics and functionality that are largely already represented in the packages which have been examined, and thus a detailed investigation would not add significant value to the survey. These technologies include:

AutoYAST2 An automated installation tool for the Suse Linux operating system with similar functionality and scope to Red Hat Kickstart.

EDWIN A technology for installing and maintaining large numbers of Microsoft Windows systems. The system is currently used within Edinburgh University. It uses Norton Ghost [gho02] for base installation, and Lanovation Picturetaker [pic02] for deployment of applications and software updates.

FAI A Linux network installation utility [fai02], with similar functionality to Kickstart and JumpStart.

Norton Ghost A disk imaging tool for Microsoft Windows-based personal computers. See [gho02] for more details.

Novadigm Radia A commercial application that automates the process of distributing and installing software packages on Windows-based nodes [rad02].

LANDesk A desktop management technology for Microsoft Windows, with similar functionality and scope to Novell ZENWorks. For more details, see the LANDesk Software Incorporated website [lan02].

Isconf A tool for maintaining system images by logging changes made to a “golden server” and applying them in the same order to other machines [isc02, TH98]. This is designed explicitly to support deterministic ordering of configuration changes thus avoiding inconsistency problems which may occur when changes are applied out of sequence (see [TB02]).

SoluCorp Linuxconf A freely available framework for writing Linux administration modules. See [Sol] for more details.

SUE An open-source, script-based system for configuring and maintaining Unix nodes. For more information about SUE, see [CER].

LUI A Linux network installation utility [IBM], with similar functionality to Kickstart.

Chapter 4

Conclusions

This report examines a number of configuration technologies, illustrating a range of different approaches to dealing with large-scale configuration management. As noted earlier, the survey is not intended to be exhaustive, but the range chosen is broadly representative of the current state-of-the-art. These technologies evade a simple, well-separated classification, but we make some general observations in the following discussion.

4.1 Summary of the Technologies

We consider different technologies in terms of an evolution from technologies that assist with low-level configuration of single nodes, through technologies that provide more extensive support for collection of nodes, towards technologies that better support diversity and ongoing management of configuration change.

4.1.1 Basic operations: low-level installation and configuration

Installing an operating system and application packages onto a node, and configuring it correctly is time consuming and error-prone. Hence, there are several technologies to automate or simplify the process of fully installing an operating system plus selected application packages on a “blank” node, usually accompanied by the ability to do at least some node-specific customization (setting hostnames, etc.). This customisation includes configuration that is specific to a node’s identity, as well as the choice of which application packages to install. Several of these technologies, e.g. Solaris JumpStart (section 3.10), Red Hat KickStart (section 3.8), are specific to a particular operating system. Others, such as Rembo (section 3.9), support multiple operating systems.

Within this category, there are two basic approaches: the use of *response files*, and the use of *gold images*.

In the case of response files, (e.g., KickStart) installation proceeds as it would if done manually, but the user choices made during installation are supplied automatically, so installation can be unattended.

In the case of gold images (e.g., Rembo), image copies are made from known, working

installations. These images are then copied onto other nodes, and can be patched after copying to accommodate node-specific configuration attributes. Patching can also be used to apply software updates to nodes. The gold image approach generally allows for quite rapid installation of nodes, since this is limited only by the speed with which the image can be deployed on the node, and not by the time taken for running a sequence of operating system and application installation and configuration tasks.

These technologies primarily focus on bringing a single node from its “bare metal”, fully unconfigured state, up to a state where they are ready to run. The technologies focus on easily configuring single, discrete nodes where there is a great deal of similarity amongst the nodes (or where there is a fairly small number of different node types), and therefore where there can be lots of reuse of the configurations.

However, this class of technology provides no real abstraction mechanism that allows multiple nodes to be thought of as one “system”, although in some cases all configurations are stored together in a single database.

4.1.2 Consolidation: dealing with multiple nodes as one

We want to be able to deal with many systems as if they were one to reduce the workload and complexity of managing large installations, and to reduce the probability of making configuration errors. We want to make configuration changes only once, and have them propagate to all nodes for which the change is necessary.

As an example, NCAPI Rocks (section 3.6) extends the capabilities of Red Hat KickStart by offering configuration support at the level of a *cluster*, meaning that the configuration of all the nodes in the cluster can be combined. It leverages the high degree of similarity typically found in cluster nodes to simplify this task, and to allow the cluster to be thought of as one system. When configuration updates are required, these are decided centrally and generally installed across all nodes in the system. Nodes in such clusters are generally dedicated to the cluster — i.e., they are not in use as desktop machines — so configuration changes can safely be determined centrally.

As with NCAPI Rocks, Rembo also provides a central configuration database (and gold image repository) which allows a set of machines to be managed centrally, where there is not too much diversity across the managed nodes.

In general, the technologies discussed so far focus on the initial configuration of a node, not on the ongoing management of changes in configuration over time.

4.1.3 Constant change: managing changing configurations

System configurations change constantly. Application packages appear, bug fixes are released, users are added, nodes are repurposed, and so on. For many installation contexts, we want a configuration system that is designed with support for such constant changes at its heart.

A very common approach adopted in many large-scale installations is to use one of the methods above to perform the initial configuration of each node, and then to use a technol-

ogy such as Cfengine (section 3.1) or SUE [CER] to manage ongoing changes to configuration parameters. A more integrated approach is offered by a technology such as LCFG (section 3.4).

LCFG — and to some extent the fabric configuration work-package of the European Data-GRID (section 3.2) which is derived from it — is capable of installing a node from the bare-metal state. However, unlike the other technologies considered thus far, LCFG is also well-targeted for managing a changing configuration over time.

The philosophy of LCFG is to specify declaratively the configuration state that is desired for each node. Each node then tracks this configuration specification, and takes actions that bring it's actual configuration into synchronization with the requested configuration. In common with a number of other approaches, this also makes it simple to rebuild a node from scratch, to its fully updated configuration, in the event of a node failure or replacement.

Three other LCFG properties are worth noting: first, it provides support by design for volatile connections, so can be used to configure nodes such as home-based machines or laptops that are only sporadically connected to the network. Secondly, it offers modularity of control and separation of concerns by grouping configuration parameters into different aspects. Finally, it provides a very fine-grained level of control on nodes, as each of them can potentially have a different configuration.

4.1.4 Doing it all: integrated packages

Technologies such as Tivoli Configuration and Operations Manager (section 3.14), Novell ZENworks (section 3.15) are representative of comprehensive commercial offerings. These technologies are capable of configuring a node from its bare-metal state, including application packages, and extend to managing a wide-spectrum of configuration changes over the life-cycle of the node. They differ in philosophy from LCFG by taking a less abstract and declarative view of the nodes being configured.

These technologies have very broad functional scope. They typically include inventory management inventory (what nodes are out there, running which packages), and detection of failures. There is often support for scheduling of configuration changes.

4.1.5 Virtualisation: flexible infrastructure provisioning

The HP Utility Data Center (section 3.3) is included in this discussion because it represents an unusual perspective on automated system configuration

The UDC's approach focuses on *virtualisation*. Assemblies of servers, network components and storage components that are physically networked together are composed into virtual server "farms" under the control of management software. A server farm can be configured very quickly by connecting the required server nodes (etc.) to a VLAN, and connecting each server to a selected disk image (and hence operating system and software configuration) in a storage array. The elements of the server farm are unaware that they are connected in this way.

The UDC approach allows virtual farms to be created, modified and removed quickly under software control. The approach (in its present form) requires hardware to be qualified for use within a UDC, so that only certain specific hardware configurations are supported. This is therefore a highly integrated approach to configuration management for specific assemblies of hardware. It's focus on security is also of note and is well-tailored to the commercial environments at which it's primarily targeted.

4.1.6 Automated, distributed applications: handling complex dependencies across nodes

One missing capability in the technologies discussed so far is the ability to configure systems where services run across a set of nodes, and where there are complex dependencies across the service components running on each node. For example, a web-service may consist of a database tier, an application logic tier and a web-server tier, each of which may be spread over multiple nodes. To realise the service correctly, the service components must be started in the right sequence, and then bound together appropriately. To ensure continuous operation, each service component must be monitored, and appropriate reconfiguration action taken if a component fails.

SmartFrog (section 3.11) is a configuration and deployment technology that focuses on this higher-level of the configuration task. It is not used for operating system installation and node ignition, but for application/service configuration and ignition. SmartFrog is able to describe applications as configurations of software components that run across a set of distributed nodes. It *realises* these descriptions by bringing up the software components in the correct sequence, correctly bound to one another, and then it manages the components as collections with pre-defined semantics through their lifecycles. Because most interesting configuration properties are consolidated into SmartFrog descriptions, it is also very easy to reconfigure applications to run differently—e.g., to increase the number of web-servers, or to locate them on more powerful nodes.

SmartFrog is distinct from the other technologies considered here in that it is able to deal with a software system considered as a single entity even though it runs over multiple, distributed nodes.

Bibliography

- [And94] Paul Anderson. Towards a hi-level machine configuration system. In *Proceedings of the 8th Large Installations Systems Administration (LISA) Conference*, pages 19–26, Berkeley, CA, 1994. Usenix.
<http://www.lcfg.org/doc/LISA8.Paper.pdf>.
- [And01] Paul Anderson. A declarative approach to the specification of large-scale system configurations. Discussion Document, February 2001.
<http://www.dcs.ed.ac.uk/~paul/publications/conflang.pdf>.
- [AS02] Paul Anderson and Alastair Scobie. LCFG: The Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002.
<http://www.lcfg.org/doc/ukuug2002.pdf>.
- [Bar01] Maite Barroso. Datagrid wp4 report on current technology. Technical report, 2001.
- [Bur95] Mark Burgess. Cfengine: a site configuration engine. *USENIX Computing systems*, 8(3), 1995.
<http://www.iu.hioslo.no/~mark/research/cfarticle/cfarticle.html>.
- [Bur01] Mark Burgess. Recent developments in cfengine. Unix.nl, 2001.
<http://www.iu.hio.no/~mark/papers/UnixNLBurgess.ps>.
- [CER] CERN. SUE. Web page.
<http://wwwinfo.cern.ch/pdp/ose/sue/doc/sue.html>.
- [Cor00] IBM Corporation. *Tivoli Management Framework User's Guide Version 3.7*, August 2000.
- [Cor01a] IBM Corporation. *Tivoli Inventory User's Guide 4.0*, 2001.
- [Cor01b] IBM Corporation. *Tivoli Software Distribution User's Guide 4.1*, 2001.
- [Cor02] Microsoft Corporation. *System Management Server 2.0 Administrator's Guide*, 2002. Available online from <http://www.microsoft.com/>.
- [CP02] Lionel Cons and Piotr Poznański. Pan: A high level configuration language. In *Proceedings of the 16th Large Installations Systems Administration (LISA) Conference*, Berkeley, CA, 2002. Usenix.

- [dat] The DataGrid Project. Web page.
<http://www.datagrid.cnr.it/>.
- [Eva97] Remy Evard. An analysis of UNIX system configuration. In *Proceedings of the 11th Large Installations Systems Administration (LISA) Conference*, page 179, Berkeley, CA, 1997.
- [fai02] Fully Automated Installation web page, 2002.
<http://www.informatik.uni-koeln.de/fai/>.
- [FP01] Dann Frazier and Greg Pratt. *SystemImager v2.0.1 Documentation*, 2001. Available online from <http://systemimager.org/manual/>.
- [gan] Ganglia Toolkit. Web page.
<http://www.millennium.berkeley.edu/ganglia/>.
- [gho02] Norton Ghost Product Datasheet from the Symantec Corporation Website, 2002.
<http://www.symantec.com>.
- [Gol03] P. Goldsack. Smartfrog: Configuration, ignition and management of distributed applications (To be published). 2003.
<http://www-uk.hpl.hp.com/smartfrog>.
- [gri02] Gridweaver Project Website, 2002.
<http://www.epcc.ed.ac.uk/gridweaver>.
- [Ham] Martin Hamilton. RedHat Linux Kickstart HOWTO. Web page.
http://metalab.unc.edu/pub/Linux/docs/HOWTO/other-formats/html_single/KickStart-HOWTO.html.
- [HP01] Matt Holgate and Will Partain. The Arusha project: A framework for collaborative systems administration. In *Proceedings of the 15th Large Installations Systems Administration (LISA) Conference*, Berkeley, CA, 2001.
- [IBM] IBM. LUI. Web page.
<http://oss.software.ibm.com/developerworks/projects/lui>.
- [Inc01a] Novell Incorporated. *Novell eDirectory 8.6 Administration Guide*, October 2001. Ref. 103-000155-001.
- [Inc01b] Novell Incorporated. *Novell ZENworks for Desktops 3.2 Preboot Services Administration*, September 2001. Ref. 103-000185-001.
- [Inc02a] Sun Microsystems Inc. *Solaris 8 Advanced Installation Guide*, 2002. Available online from <http://docs.sun.com/>.
- [Inc02b] Apple Computers Incorporated. *Mac OS X Server Technologies NetBoot*, 2002. Available online from <http://www.apple.com/server/netboot.html>.

- [Inc02c] Apple Computers Incorporated. *Mac OS X Server Technologies Network Install*, 2002. Available online from <http://www.apple.com/server/netboot.html>.
- [Inc02d] Novell Incorporated. *Novell ZENworks for Desktops 3.2 Administration*, June 2002. Ref. 103-000122-001.
- [Inc02e] Red Hat Incorporated. *Red Hat Linux 7.3: The Official Red Hat Linux Customization Guide*, 2002. Available online from <http://www.redhat.com/>.
- [isc02] Isconf.Org Website, 2002.
<http://www.isconf.org>.
- [lan02] Landesk Software Incorporated Website, 2002.
<http://www.landesksoftware.com>.
- [LFBE02] Mark Logan, Matthias Felleinsen, and David Blank-Edelman. Environmental aquisition in network management. In *Proceedings of the 16th Large Installations Systems Administration (LISA) Conference*, Berkeley, CA, 2002.
- [pic02] Lanovation Incorporated Website, 2002.
<http://www.lanovation.com>.
- [PMP01] Greg Bruno Philip M. Papadopoulos, Mason J. Katz. Npaci rocks: Tools and techniques for easily deploying manageable linux clusters. Cluster 2001, October 2001.
<http://www.rocksclusters.org/rocks-documentation/papers/clusters2001-rocks.pdf>.
- [rad02] Novadigm Radia, 2002.
<http://www.novadigm.com>.
- [Red99] IBM Corporation/ Redbook. *An Introduction to Tivoli Enterprise*, October 1999.
- [Red00] IBM Corporation/ Redbook. *An Introduction to Tivoli Software Distribution 4.0*, December 2000.
- [SaR01] Rembo Technology SaRL. *REMBO: A Complete Pre-OS Remote Management Solution REMBO Toolkit 2.0 Manual*, 2001. Available online from <http://www.rembo.com/>.
- [Sol] Solucorp. Linuxconf. Web page.
<http://www.solucorp.qc.ca/linuxconf/>.
- [TB02] Steve Traugott and Lance Brown. Why order matters: Turing equivalence in automated systems administration. In *Proceedings of the 16th Large Installations Systems Administration (LISA) Conference*, Berkeley, CA, 2002.
- [TH98] Steve Traugott and Joel Huddleston. Bootstrapping an infrastructure. In *Proceedings of the Twelfth Systems Administration Conference (LISA'98)*, December 1998.

- [udc02] HP Utility Data Center Website, 2002.
<http://www.hp.com/large/infrastructure/utilitydata/overview>.
- [YSS00] D. R. Yocum, C. Sieh, and D. Skow. A modular administration tool for linux computers. 2000.

Glossary

Definitions

For each of the following definitions we indicate the area to which they are most related (language, model or system).

abstraction (model) The process by which distracting complexity is hidden in a description.

application (system) A collection of components performing a function. The application defines the relationships between the individual component lifecycles.

aspect (system) A collection of configuration elements which share a common purpose and point of interest. When considered together, the grouping of elements within an aspect should describe a meaningful and (at an abstract level) interesting function of the fabric. Aspects should also provide a level of modularity to the configuration of the fabric.

aspect-based access control (system) The ability to define which users are permitted to control particular aspects of a node's configuration.

aspect-oriented process engineering (system) An application of aspect-oriented techniques to business process.

aspect-oriented programming (system) An extension of object-oriented programming which allows the independent specification of different aspects. Usually, the aspects affect overlapping, and widely-separated regions of the program.

asymptotic configuration (system) It may not be possible for highly dynamic or very large scale systems to provide transactional guarantees that the entire system has reached its intended configuration. In such systems individual nodes will get or apply their configuration when they are able to, and the whole configuration is then guaranteed to be reached over an indefinite period of time, hence "asymptotically".

attribute (language) A name/value pair. The value may be optional.

bootstrapping an operating system (system) The process of initialisation an operating system on a node, using a small application called the bootstrap.

class (model) A feature provided by many configuration tools for specifying groups of nodes with similar properties; the class membership may be declared explicitly, or it may be computed automatically, based on some property of the nodes, such as the platform.

cloning (system) A procedure by which the configuration of a node is replicated to one or more other nodes.

cluster (model/system) A system that consists of a collection of interconnected nodes used as a single, unified computing resource.

component (system) When referring to a software entity in a configuration management technology, a component is a software module able to either interpret configuration information into the necessary configuration files for a given operating system or application, and/or to interpret configuration actions.

For example, a component may turn parts of a Linux node description into the `/etc/hosts` file, and another may be able to start or stop the DNS server daemon.

configuration description A structured description of the intended system, application, or component to be instantiated. A configuration description may be deployed by a configuration management technology that is able to interpret it and turn it into a component/application/system inside the fabric.

Example: an individual node configuration description could be describing the software on this node and its configuration parameters.

configuration rot (system) The degradation of a system configuration due to inaction or unsolicited configuration modifications.

configuration management technology (system) An application or environment which, given the description of a target configuration, will trigger configuration actions in order to transform the fabric with the aim of reaching the desired target configuration.

compilation (language) The processes by which the configuration management technology generates the description of the intended system.

In particular the following terms all relate to language constructs or operations:

- derivation or history : the steps taken to derive a particular attribute value from the values specified in included prototypes.
- mutation : the process, other than simple overriding, which evaluates the value of an attribute by considering values from one or more other prototypes.
- overriding or overwriting : the process by which the value of an attribute is replaced by the last value included from a prototype.
- parameterization : a selection of attributes to be considered as variables for a given prototype. Any subsequent use of the value inside the same prototype is done by using a reference to the parameter.

- **referencing** : pointing to an externally-defined value, or linking to another part of the configuration description (typically by indicating the name of the attribute).

declarative description (model) A specification (collection of expressions) which describes the target configuration of the fabric (or part of the fabric) in terms of the end-state to be achieved.

default policy (model) A set of constructs which determine the default attribute values of an aspect of the fabric.

dependency (model/system) A prerequisite of a configuration change (that is a profile) on other profiles.

deployment (system) Refers to the actual process of turning a static description of a component, application, or system into a running entity on the fabric.

Specifically, in some configuration technologies, refers to the invocation of the configuration procedures contained within one or more procedural profiles. For example, the action of taking compiled software and placing it in a running state.

devolution of management (system) The process of determining and implementing an (efficient and effective) distribution of responsibility and necessary privileges between different nodes within the fabric.

Designated nodes may be allocated responsibility according to locality or task. For example, a designated node may be responsible for the complete configuration of all resources within its locality. At the other extreme, a node may be responsible for a single aspect of configuration of the fabric as a whole.

disconnected operation The ability (or inability) of a node to continue functioning when not connected to the network.

diversity (system) An adjective applied to a fabric in which considerable differences exist between the hardware makeup of and the system configuration of individual (or clusters of) nodes.

An implication of diversity is that the fabric cannot be efficiently and effectively configured using a cloning procedure alone.

dynamic validation (system) For a configuration technology, refers to the ability to check for the validity of a predicate at any point in time inside the deployed system.

Typically, the predicate would be described as a static predicate, and the configuration management technology would ensure that it is always valid at runtime (hence taking 'dynamic' configuration actions in case of inconsistencies).

dynamism (system) An adjective applied to a fabric for which frequent configuration changes are applied, either to the whole or a subset of the fabric.

An implication here is that it becomes difficult (or impossible) to ascertain the configuration of the whole fabric at any particular instance in time.

fabric (model/system) Term used to refer to the complete set of resources to be considered and managed.

fire and forget (system) A configuration technology is sometimes said to “fire and forget” if it only cares about the application of a configuration change, but does not provide any subsequent monitoring of the success or the conformity of this configuration change in the system.

golden server (system) See pristine installation.

high-level definition (language) An abstracted or implicit description of low level resources of the fabric.

idempotency (model/system) An operation which, when applied to a specific set of data, gives the same result no matter how many times it is performed.

ignition of a node (system) A unifying term that describes the process required to take a node from a baseline state—having no operating system, applications, or configuration information applied—to a state in which it can be assimilated into the fabric as a managed resource. This generally involves:

- Low-level tasks, such as: partitioning the local hard drive, applying BIOS updates, and configuring network information.
- Installing a target operating system and any client applications that may be required to manage the node within the fabric.
- Applying default policies to bring the configuration of the node into line with its peers.

implicit declaration (language) Constructs that are converted into explicit declarations later in the processing phase.

inconsistency (system) Discrepancy between the desired configuration and the actual configuration (of an individual node or group of nodes). Inconsistencies can be established by testing validation policies.

inventory (system) The process of collecting actual configuration information relating to (a part of or) the entire fabric at a specific instance in time.

inventory discovery (system) The procedure by which the actual configuration of a node (or collection of nodes) is acquired.

language A formal (normally textual) notation for describing properties of abstract entities and the relationships between them. A language definition consists of formal rules (the Backus Naur Form or “BNF”) for constructing terms (also expressions, sentences) of the language and a set of rules for interpreting these terms within an underlying formal (mathematical) system (i.e. the semantic models and semantic mappings).

model or resource model The aspects (attributes, relationships, etc) of a resource that require representation using the language. For example a node model might include CPU type and memory capacity, but would probably not include colour of casing.

node (model/system) A uniquely identifiable processing location within the fabric. Commonly, a node refers to a computer or other device, such as a printer.

The question of how to identify each individual node is non-trivial—it is often sufficient to label a node using a network identifier called the Media Access Control (MAC) address. However, some nodes have more than one such address.

In the literature, the term node is often replaced by other terms such as: computer, host, and workstation. However, we feel that the generic term “node” is more precise.

operating system ignition (system) The process implemented to install a clean node (with no software or configuration previously applied) with an operating system and other fundamental tools to allow the node to interact with the fabric.

parameterisation (language) The ability to create multiple references to a value defined elsewhere (in a single place).

policy region (model) A collection of nodes which share one or more default and/or validation policies in common. A node may belong to more than one policy region.

preboot execution environment(PXE) (system) An industry standard client/server interface that allows nodes which are not yet loaded with an operating system to be ignited from a remote location by an administrator.

The PXE code is typically held on a read-only memory chip installed on the network interface of the target node. Alternatively, the PXE code could be provided by a local boot diskette, though this would necessitate local, manual involvement in the ignition process.

pristine installation (system) A description of a node which has been determined to have a basic installation of an operating system (and possibly software). But no configuration changes applied.

This type of node would be used for a cloning procedure, in which case it is often referred to as a golden server.

procedural description (model) A specification that describes a target configuration in terms of the procedures which must be implemented to achieve it.

procedural profile (system) A description of the procedure for implementing a related set of configuration changes.

prototype (language) A collection of attributes and references to other prototypes.

prototype-based language (language) An (object-oriented) language in which new objects are created by cloning other prototypes rather than creating new instances of the class.

push or pull (system) These represent two complementary methods of deploying configuration change. In the push approach, a server initiates a configuration change on the client nodes that it manages and is responsible for the scheduling and the transmission of configuration information. In the pull model, configuration changes are advertised by a server, and each client requests configuration information when it is ready.

item[resource (model/system)] Any entity that requires description using a formal language. This may include physical objects such as nodes, storage, firewalls and networks, as well non-physical entities such as software components, data or licenses. A resource may be composite, in that a resource may be constructed from sub-resources, for example a cluster resource may be composed of a collection of other resources such as nodes.

rollback (system) The procedure by which a configuration change is reversed, restoring a node (collection of nodes) to its (their) exact configuration prior to the change.

schema (language) A set of rules defining the attributes that may appear in a particular description, and their legal values. A description is said to be validated against a schema.

service advertisement (system) The process by which an entity in the system can publish attributes inside a directory on the system.

service discovery (system) The process by which an entity in the system can query for services published under given attributes inside a directory somewhere in the system, and is returned the location of matching advertised services.

snapshot (system) A static record of the configuration and file structure of a node at a prescribed instance in time. This record provides a comprehensive description of all files, directories, and configuration resources required to replicate the exact configuration of the original node onto another.

A typical use of the snapshot technique is to build an installation package for a software component. Having taken a snapshot of a pristine installation node both prior to and subsequent to the installation of a software package, one can determine the changes that were applied by differencing the two snapshots. Theoretically, this allows the software package to be installed onto another node, simply by applying the same changes. However, this technique is highly error-prone since, in order to guarantee the correctness of the software installation, each target node must have an identical configuration to the source pristine installation.

static (system) An adjective applied to a tool which is capable of configuring a node (or group of nodes) according to a configuration description, but is not able to maintain the node(s) automatically, in the event of a later configuration change.

synchronisation (model/system) The process by which temporal dependencies within a configuration change are determined and, where possible, are resolved.

system An informal term meaning a collection of resources satisfying some implied completeness or closure condition.

system configuration (model) The sum of all of the configuration changes applied to a system during its initialisation and thereafter.

The system configuration may not represent an exhaustive description of everything that may be configured: but rather the aspects that will be configured.

transactional characteristic (model) Evaluation of a set of actions as an atomic unit, for which the system can guarantee either completion or controlled failure.

template (model) A standard configuration description (for example containing known, validated, default values) that may be used to build specialized configurations descriptions.

validation policy (model) An expression to be evaluated in order to test the compliance of a description with a predicate.

variable (language) A token or placeholder which may be assigned a value.

virtual resource (system) A logical fabric resource that is transparently interpreted by the system running on the fabric as a physical fabric resource. For example, a single physical machine (single IP address) may be virtualised by the fabric into several logical machines (each with its own IP address) that will be seen as completely distinct by the applications on top of the fabric.

volatile connection (model/system) A node which is anticipated to perform disconnected operations.