

Survey of Configuration Management Tools

Thomas Delaet
Wouter Joosen

Report CW 485, April 2007



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Survey of Configuration Management Tools

Thomas Delaet

Wouter Joosen

Report CW 485, April 2007

Department of Computer Science, K.U.Leuven

Abstract

Over the last years, there has been an enormous boost in the number of solutions available for automating system configuration tasks. However, there is no generally accepted taxonomy framework that serves as a basis for evaluating configuration management solutions. In this report, we present such a taxonomy. We also apply this taxonomy to a sample of configuration management tools.

Keywords : configuration management, network management, system administration

CR Subject Classification : C.2.3, K.6.4

Survey of Configuration Management Tools

Thomas Delaet, Wouter Joosen
DistriNet, Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
{thomas,wouter}@cs.kuleuven.ac.be

April 18, 2007

Contents

1	Taxonomy Framework	1
1.1	Introduction	1
1.2	Configuration Management Taxonomy	1
1.2.1	Abstraction Level	2
1.2.2	Specification Language	3
1.2.3	Consistency	5
1.2.4	Distributed Management	7
2	Bcfg2 Evaluation Report	9
2.1	Abstraction Level	9
2.2	Specification Language	9
2.2.1	Usability	9
2.2.2	Domain Coverage	9
2.2.3	Grouping mechanism	9
2.2.4	Multi-level Specification	10
2.3	Consistency	10
2.3.1	Dependency Modeling	10
2.3.2	Conflict Management	10
2.3.3	Workflow Management	10
2.4	Distributed Management	10
2.4.1	Federated Management	10
2.4.2	Distributed Translation	10
2.5	Acknowledgements	10
3	Bladelogic Evaluation Report	11
3.1	Abstraction Level	11
3.2	Specification Language	11
3.2.1	Usability	11
3.2.2	Domain Coverage	11
3.2.3	Grouping mechanism	11
3.2.4	Multi-level Specification	11
3.3	Consistency	12
3.3.1	Dependency Modeling	12
3.3.2	Conflict Management	12
3.3.3	Workflow Management	12
3.4	Distributed Management	12
3.4.1	Federated Management	12
3.4.2	Distributed Translation	12
3.5	Acknowledgements	12

4	Cfengine Evaluation Report	13
4.1	Abstraction Level	13
4.2	Specification Language	13
4.2.1	Usability	13
4.2.2	Domain Coverage	13
4.2.3	Grouping mechanism	13
4.2.4	Multi-level Specification	14
4.3	Consistency	14
4.3.1	Dependency Modeling	14
4.3.2	Conflict Management	14
4.3.3	Workflow Management	14
4.4	Distributed Management	14
4.4.1	Federated Management	14
4.4.2	Distributed Translation	14
5	Firmato Evaluation Report	15
5.1	Abstraction Level	15
5.2	Specification Language	15
5.2.1	Usability	15
5.2.2	Domain Coverage	15
5.2.3	Grouping mechanism	15
5.2.4	Multi-level Specification	15
5.3	Consistency	16
5.3.1	Dependency Modeling	16
5.3.2	Conflict Management	16
5.3.3	Workflow Management	16
5.4	Distributed Management	16
5.4.1	Federated Management	16
5.4.2	Distributed Translation	16
6	LCFG Evaluation Report	17
6.1	Abstraction Level	17
6.2	Specification Language	17
6.2.1	Usability	17
6.2.2	Domain Coverage	17
6.2.3	Grouping mechanism	17
6.2.4	Multi-level Specification	17
6.3	Consistency	18
6.3.1	Dependency Modeling	18
6.3.2	Conflict Management	18
6.3.3	Workflow Management	18
6.4	Distributed Management	18
6.4.1	Federated Management	18
6.4.2	Distributed Translation	18
6.5	Acknowledgements	19
7	Microsoft SMS Evaluation Report	20
7.1	Abstraction Level	20
7.2	Specification Language	20
7.2.1	Usability	20

7.2.2	Domain Coverage	20
7.2.3	Grouping mechanism	20
7.2.4	Multi-level Specification	20
7.3	Consistency	21
7.3.1	Dependency Modeling	21
7.3.2	Conflict Management	21
7.3.3	Workflow Management	21
7.4	Distributed Management	21
7.4.1	Federated Management	21
7.4.2	Distributed Translation	21
8	Netdirector Evaluation Report	22
8.1	Abstraction Level	22
8.2	Specification Language	22
8.2.1	Usability	22
8.2.2	Domain Coverage	22
8.2.3	Grouping mechanism	22
8.2.4	Multi-level Specification	22
8.3	Consistency	22
8.3.1	Dependency Modeling	22
8.3.2	Conflict Management	23
8.3.3	Workflow Management	23
8.4	Distributed Management	23
8.4.1	Federated Management	23
8.4.2	Distributed Translation	23
9	Opsware Evaluation Report	24
9.1	Abstraction Level	24
9.2	Specification Language	24
9.2.1	Usability	24
9.2.2	Domain Coverage	24
9.2.3	Grouping mechanism	24
9.2.4	Multi-level Specification	25
9.3	Consistency	25
9.3.1	Dependency Modeling	25
9.3.2	Conflict Management	25
9.3.3	Workflow Management	25
9.4	Distributed Management	25
9.4.1	Federated Management	25
9.4.2	Distributed Translation	25
10	Puppet Evaluation Report	26
10.1	Abstraction Level	26
10.2	Specification Language	26
10.2.1	Usability	26
10.2.2	Domain Coverage	26
10.2.3	Grouping mechanism	26
10.2.4	Multi-level Specification	26
10.3	Consistency	27
10.3.1	Dependency Modeling	27

10.3.2 Conflict Management	27
10.3.3 Workflow Management	27
10.4 Distributed Management	27
10.4.1 Federated Management	27
10.4.2 Distributed Translation	27
11 IBM Tivoli Evaluation Report	28
11.1 Abstraction Level	28
11.2 Specification Language	28
11.2.1 Usability	28
11.2.2 Domain Coverage	28
11.2.3 Grouping mechanism	28
11.2.4 Multi-level Specification	28
11.3 Consistency	29
11.3.1 Dependency Modeling	29
11.3.2 Conflict Management	29
11.3.3 Workflow Management	29
11.4 Distributed Management	29
11.4.1 Federated Management	29
11.4.2 Distributed Translation	29
12 Conclusion	30

Abstract

Over the last years, there has been an enormous boost in the number of solutions available for automating system configuration tasks. However, there is no generally accepted taxonomy framework that serves as a basis for evaluating configuration management solutions. In this report, we present such a taxonomy. We also apply this taxonomy to a sample of configuration management tools.

Chapter 1

Taxonomy Framework

1.1 Introduction

Over the last years, there has been an enormous boost in the number of solutions available for automating system configuration tasks. However, there is no generally accepted taxonomy framework that serves as a basis for evaluating configuration management solutions. In this chapter, we define such a taxonomy.

The taxonomy framework we propose in section 1.2 is illustrated with a comparison of a sample of configuration management tools in the following chapters. We summarize the evaluation reports in chapter 12.

1.2 Configuration Management Taxonomy

Our approach in developing a configuration management taxonomy is both top-down and bottom-up. Top-down, we identify user requirements for an advanced configuration management solution. Bottom-up, we look at the key features of existing solutions.

The user requirements for a configuration management solutions can all be summarized as “minimizing configuration errors”. As stated previously, configuration management solutions exist because humans make mistakes. Solutions that automate manual practices can help to avoid mistakes. This is especially true in environments that are growing more complex due to increasing scale, heterogeneity, changing requirements and unpredictable behaviour. An advanced configuration management solution should provide mechanisms to deal with these four evolutions.

The rest of this section defines four categories of classification criteria dealing with abstraction levels, the specification language, consistency and distributed management.

1. **Abstraction Level:** The language used by a configuration management solution can be classified at different levels of abstraction, ranging from high-level end-to-end requirements, to low-level bit-configurations.
2. **Specification Language:** In this section, we discuss four specification language criteria dealing with usability, domain coverage of a configuration management solution, the grouping mechanism and specifications at multiple abstraction levels.
3. **Consistency:** In this section, we discuss three criteria that ensure consistency in a computer infrastructure: modeling dependencies, conflict management and workflow management.
4. **Distributed Management:** Distributed management deals with federated management and distributes translation of configuration specifications.

1.2.1 Abstraction Level

The language used by a configuration management solution can be classified at different levels of abstraction, ranging from high-level end-to-end requirements, to low-level bit-configurations. The basic task of all configuration management solutions is to automate manual system administration. To achieve this, a configuration management solution offers a user interface that makes abstraction of manual system administration. The possible levels of abstraction can be classified as follows.

1. **End-to-end requirements:** End-to-end requirements are typical non-functional requirements [33]. They describe service characteristics that the computing infrastructure must achieve. Figure 1.1 shows an example of a performance characteristic for a mail service. Other types of end-to-end requirements deal with security, availability, reliability, usability, . . . There are currently no configuration management solutions that provide an interface at the level of end-to-end requirements. In [30], an approach based on first-order logic is proposed for expressing end-to-end requirements.
2. **Instance distribution rules:** At the abstraction level of instance distribution rules, the distribution of instances in the network is specified. We will frequently use the concept of instances in this article. We define an instance as a unit of configuration specification that can be decomposed in a set of parameters. Examples of instances are mail servers, DNS clients, firewalls and web servers. A web server, for example, has parameters for expressing its port, virtual hosts and supported scripting languages. In Figure 1.1, the instance distribution rule prescribes the number of mail servers that need to be activated in an infrastructure. To the best of our knowledge, no current configuration management solution can deal with instance distribution rules. The need for such a language is uttered in [5] and [2].
3. **Instance configurations:** At the level of instance configurations, each instance is an implementation independent representation of a configuration. An example of a solution at this level is Firmato [6], that allows modeling firewall configurations independent of the implementation software used.
4. **Implementation dependent instances** The level of implementation dependent instances specifies the required configuration in more detail. It describes the configuration specification in terms of the contents of software configuration files. In the example in Figure 1.1 a `sendmail.cf` file is used to describe the configuration of mail server instances. Solutions like LCFG [4] and Puppet [24] operate at the level of implementation dependent instances.
5. **Configuration files:** At the level of configuration files, complete configuration files are mapped to a device or set of devices. In contrast to the previous level, this level has no knowledge of the contents of a configuration file. Many configuration management solutions, like IBM Tivoli [19], Bcfg2 [14], Cfengine2 [8] and Bladelogic [20] use this abstraction level.
6. **Bit-configurations:** At the level of Bit-configurations, disk images or diff's between disk images are mapped to a device or set of devices. This is the lowest level of configuration specification. Bit-level specifications have no knowledge of the contents of configuration files or the files itself. Examples of management solutions that operate at this level are imaging systems like Partimage [32], g4u [17] and Norton Ghost [36].

Figure 1.1 shows the six abstraction levels for configuration management, illustrated with an email setup. The illustration in Figure 1.1 is derived from an example discussed in [5]. The different abstraction levels are tied to the context of configuration management. In the context of policy languages, the classification of policy languages at different levels of abstraction is often done by distinguishing between high-level and low-level policies [37, 29]. The distinction of what exactly is a high-level and low-level policy language is rather vague. In many cases, high-level policies are associated with the level that we call end-to-end requirements, while low-level policies are associated with the implementation dependent instances level. We believe that a classification tied to the context of configuration management gives a better insight in the different abstraction

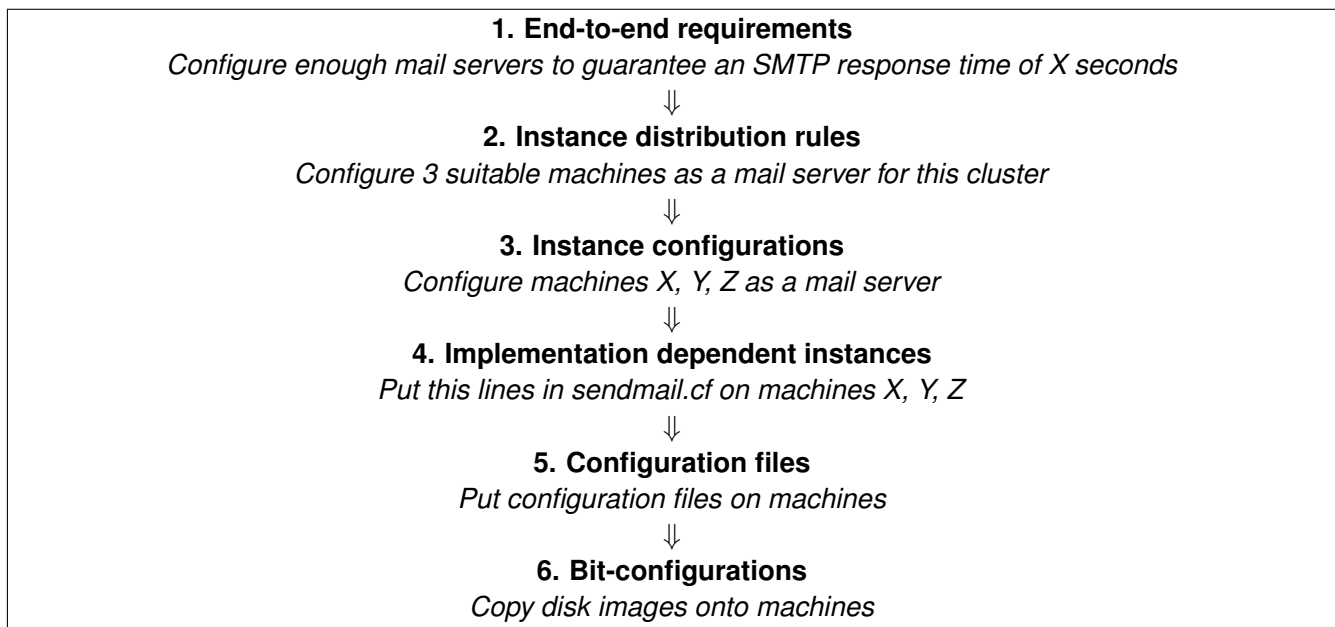


Figure 1.1: An example of different abstraction levels of configuration specification for an email setup.

levels used by configuration management solutions. In conclusion, a configuration management solution automates the deployment of configuration specifications. From the level of bit-configurations, deployment is simply copying bit-sequences to disks, while deploying configurations specified as end-to-end requirements requires a much more complex process.

1.2.2 Specification Language

In this section, we discuss four criteria that characterize the specification language of a configuration management solution. These criteria are (1) usability, (2) domain coverage of a configuration management solution, (3) the grouping mechanism and (4) specifications at multiple abstraction levels.

1. **usability:** A critical factor in a solution's adoption success is its user interface for specifying configurations and for monitoring configurations [26, 12, 25, 2]. In both cases, we distinguish between language based approaches and graphical user interfaces.
2. **domain coverage:** A configuration management solution can be a limited or general-purpose solution in terms of supported platforms and functionality
3. **grouping mechanism:** To avoid replication, shared configuration specifications need a grouping mechanism.
4. **multi-level specification:** The automated translation of configuration specifications to deployed configurations can be complemented with user defined specifications at multiple abstraction levels.

All four of these criteria are largely independent of a solution's abstraction level. We define the specification language of a solution as the representation of a solution's abstraction level as discussed in the previous section.

Usability

A critical factor in a solution's adoption success is its user interface for specifying configurations and for monitoring configurations [26, 12, 25, 2]. In both cases, we distinguish between language based approaches and

graphical user interfaces.

1. **Configuration Specification User Interface:** The configuration specification can be created by editing text files or by a point-and-click interface. In the former case, a configuration management solution defines a language for specifying configurations. In the latter case, a GUI or web-based interface is provided. Solutions like Cfengine, Bcfg2 and LCFG use a language-based approach, while Netdirector, Bladelogic and IBM Tivoli are examples of solutions that use a graphical approach. Language based approaches typically have a steep learning curve. However, some system administrators prefer language based approaches because they can, once mastered, provide a higher productivity. Another advantage of language based approaches is that they define open interfaces. The language definition of solution X can be used as output format of solution Y. This is more difficult with graphical user interfaces. The advantage of graphical approaches is its accessibility.
2. **Configuration Monitoring User Interface:** A configuration management solution can provide an integrated (graphical) monitoring system and/or define a (language-based) interface for other solutions to check the state of an infrastructure. A language-based interface has the advantage that multiple monitoring systems can be connected with the configuration management solution. A monitoring system allows the user to check the current state of the infrastructure and the delta with the configuration specification. Remember that one of the evolutions sketched in the introduction is that computer infrastructure show unpredictable behaviour due to bugs, viruses and vulnerabilities. Therefore, we can not assume that a configuration specification, once deployed, does not change anymore. That is way a configuration management solution needs to provide a mechanism to monitor the current state of an infrastructure and the delta with the configuration specification.

Domain Coverage

A configuration management solution can be limited or general-purpose solution in terms of supported platforms and functionality. Comparing the functionality and platforms present in an infrastructure with the supported functionality and platforms of a configuration management solution is crucial to select the appropriate (set of) configuration management solutions for an infrastructure.

1. **Supported functionality:** A solution like Firmato is limited to firewall configurations, while solutions like Cfengine2, LCFG, Bcfg2 are general-purpose solutions.
2. **Supported platforms:** Microsoft SMS is limited to the Windows platform, while many open-source solutions focus on Unix-systems. Only few solutions (for example Firmato) also allow to specify configurations for network devices like routers and switches.

Grouping Mechanism

To avoid replication, shared configuration specifications need a grouping mechanism. In many infrastructures, some properties are shared between a subset (or multiple overlapping subsets) of devices [5]. For example, devices need the same DNS client configuration, authentication mechanism, shared file systems, . . .

Multiple implementations of grouping mechanisms are possible: LCFG [4] uses a preprocessor to group shared properties, Bcfg2 [14] defines a group construct similar to a Unix group that can have members and Cfengine [8] uses the concept of classes that can be combined with boolean logic to select common sets of configuration specifications.

Examples of advanced grouping mechanisms that avoid replication even more are automatic definition of groups and recursive groups.

1. **automatic definition of groups:** In Cfengine, a device inherits a predefined set of classes based on environmental data like its architecture, operating system, kernel, This predefined set of classes can be augmented with user defined classes.

2. **recursive groups**: Bcfg2 allows devices to be a member of one or more groups (comparable to the Unix group construct), but groups can also be a member of other groups.

Multi-level Specification

The automated translation of configuration specifications to deployed configurations can be complemented with user defined specifications at multiple abstraction levels. When a configuration management solution's language implements one of the higher abstraction levels in Figure 1.1, the details of the configuration specification are added by the solution itself. Indeed, more abstract specifications are less detailed specifications. At first, it seems that a solution has to make an engineering trade-off between providing an abstract specification language, close to business requirements on one side and providing the solution's users with a language that allows detailed specifications. However, a solution can also support for multi-level specifications by providing an interface to specify extra details at each level in the translation process [34]. For example, a solution with an abstraction level of instance distribution rules (Figure 1.1) provides automatic translation of instance distribution rules to deployed configurations. When such a solution supports multi-level specifications, the automated translation can be complemented with a specification at the level of, for example, implementation dependent instances. Such a multi-level specification can be useful when the automated translation does not give satisfactory results.

Since many solutions operate at the lower two levels of figure 1.1, they have little need for multi-level specification. A solution at a higher-level, like LCFG [4] uses templates for translating key-value pairs into configuration files. Since LCFG's templates can be modified easily, they allow for multi-level specification.

1.2.3 Consistency

In this section, we discuss three criteria that ensure consistency in a computer infrastructure: modeling dependencies, conflict management and workflow management.

1. **dependency modeling**: Modeling dependencies is, like the grouping mechanism of Section 1.2.2, a mechanism for minimizing replication in the configuration specification.
2. **conflict management**: Because a configuration specification can contain conflicting definitions, a configuration management solution should provide a mechanism to deal with conflicts.
3. **workflow management**: Workflow management deals with planning and execution of (composite) changes in a configuration specification.

As stated previously, downtime due to configuration errors has a significant effect upon the cost of managing an infrastructure. Since it is impossible for system administrators to oversee all consequences of a change in the configuration of an infrastructure, automated solutions are needed for enforcing consistency of the overall configuration. Support for consistency is not only a matter of consistent configuration for an individual system. It also concerns the interactions between different systems on a network.

Dependency Modeling

Modeling dependencies is, like the grouping mechanism of Section 1.2.2, a mechanism for minimizing replication in the configuration specification. When dependencies are made explicit, a solution can support automatically changing configurations that depend on each other. For example, when the location of a DNS server changes and the dependency between the DNS server and clients is modeled in the configuration specification, a configuration management solution can automatically adapt the client configurations to use the the new server. Again, dependency modeling reduces the chance of introducing errors in the configuration specification.

In Section 1.2.1, we defined an instance as a unit of configuration specification that can be decomposed in a set of parameters. Examples of instances are mail servers, DNS clients, firewalls and web servers. A web server, for example, has parameters for expressing its port, virtual hosts and supported scripting languages. Based on this definition, we can classify dependencies in three categories: (1) dependencies between instances, (2) dependencies between parameters and (3) dependencies between a parameter and an instance.

1. **Instance dependencies** represent a coarse grained dependency between instances. Instance dependencies can exist between instances on the same device, or between instances on different devices. An example of the former is the dependency between a DNS server instance and the startup system instance on a device: if a startup system instance is not present on a device, the DNS server instance will not work. An example of dependencies between instances on different devices is the dependency between DNS servers and its clients.
2. **Parameter dependencies** represent a dependency between parameters of instances. An example of this is a CNAME record in the DNS system: every CNAME record also needs an A record.
3. **Parameter - instance dependencies** are used to express a relation between an individual parameter and an instance. For example a mail server instance has a dependency that there needs to be an MX record in the DNS server instance that refers to the mail server.

Note that it depends on the abstraction level of a solution which dependencies it can support. The two lowest abstraction layers in Figure 1.1, configuration files and bit-configurations, have no knowledge of parameters and as a consequence, they can only model instance dependencies.

Solutions that support dependency modeling, typically do this by using references (LCFG and Puppet). Firmato uses an Entity-Relationship diagram for modeling dependencies. These approaches are limited to modeling parameter dependencies. Other solutions, like Bladelogic, only allow to model instance dependencies instances by defining a relative order between instances.

Conflict Management

A configuration specification can contain conflicting definitions, so a configuration management solution should have a mechanism to deal with conflicts. Despite the presence of grouping mechanisms and dependency modeling, a configuration specification can still contain errors, because it is written by a human. In case of such an error, a conflict is generated. Conflicts can be classified in two types: application specific conflicts and contradictions in the policy specification, also called modality conflicts [27].

1. **application specific conflicts**: An example of an application specific conflict is the specification of two Internet services that use the same TCP port. In general, application specific conflicts can not be detected from the policy specification. Examples of research on application specific protocols can be found in [18] and [11] where conflict management for IPSec and QoS policies is described.
2. **modality conflicts**: An example of a modality conflict is the prohibition and obligation to enable an instance (for example a mail server) on a device. In general, modality conflicts can be detected from the policy specifications.

When a configuration specification contains rules that cause a conflict, this conflict should be detected and acted upon. A solution like Bcfg2 can detect modality conflicts but has only one associated action: signal an error. LCFG allows to detect both application specific conflicts and modality conflicts. When a conflict is detected in LCFG, it allows to specify a set of actions to execute in order to resolve the conflict.

Workflow Management

Workflow management deals with planning and execution of (composite) changes in a configuration specification. If configuration specification were static, it would not be necessary to use a configuration management solution. It is the high number of changes that makes the configuration management problem difficult to handle. Changes can affect services distributed over multiple machines and with dependencies on other services [5, 31]. One aspect of workflow management is state transfer. The behavior of a service is not only driven by its configuration specification, but also by the data it uses. In the case of a mail server, the data are the mail spool and mailboxes, while web pages serve as data for a web server. When upgrading a service or transferring a service to another device, one has to take care that the state (collection of data) remains consistent in the face of changes. Another aspect of workflow management is the coordination of distributed changes. This has to be done very carefully as not to disrupt operations of the computing infrastructure. A change affecting multiple machines and services has to be executed as a single transaction. For example, when moving a DNS server from one device to another, one has to first activate the new server and make sure that all clients use the new server before deactivating the old server. For some services, characteristics of the managed protocol can be taken into account to make this process easier. For example, the SMTP protocol retries for a finite span of time to deliver a mail when the first attempt fails. A workflow management protocol can take advantage of this characteristic by allowing the mail server to be unreachable during the change.

In [15] a mechanism based on a version control system is proposed for Bcfg2. [38] discusses a more advanced algorithm and applies it to VLAN management.

1.2.4 Distributed Management

Distributed management deals with federated management and distributes translation of configuration specifications.

1. **federated management:** Federated management implies the distribution of management responsibilities.
2. **distributed translation:** The distributed translation criterion represents the network model used to translate and deploy configuration specifications.

The nature of the subjects that need to be managed¹ is inherently distributed, but the configuration management solution itself can be distributed as well [31]. The most simple management case is where one person is responsible for managing the whole computer infrastructure without any connection to an external network or communication with other computer systems. This is not likely to occur very often. In many cases, there is some form of distributed management.

Federated Management

Federated management implies that management responsibilities are distributed. Management responsibilities can be distributed based on sets of machines, functional characteristics, or a combination of both [5]. An example of such a combination can be someone who is responsible for dealing with the web and mail infrastructure for the sales department of an organization. When federated management is supported, it is the task of the configuration management solution to assemble different configuration specifications and resolve potential conflicts. When supporting multiple managers, a mechanism needs to be present for authenticating managers and an authorization mechanism is needed for limiting a manager's powers.

Netdirector and Bladelogic are two solutions that support distributed specification. Netdirector allows to link groups of devices and functionality with a system administrator while Bladelogic uses a role based approach for specifying who is allowed to do what kind of changes.

¹an infrastructure of computing devices

Distributed Translation

The distributed translation criterion represents the network model used to translate and deploy configuration specifications. A configuration management solution typically consists of two agents: a translation agent and a deployment agent.

1. **translation agent:** Distributed Management approaches can be classified in three categories, based on the number of agents compared to the number of managed devices: centralized management, weakly distributed management and strongly distributed management [28].
 - (a) **centralized management** is the central server approach with only one translation agent. This approach is used by solutions like LCFG, Bcfg2 and Netdirector. When dealing with huge networks, the central server quickly becomes a bottleneck. The latter is certainly the case when a configuration management solution uses a high-level abstraction since the algorithm for computing a device's configuration becomes complex.
 - (b) **weakly distributed management** is an approach where multiple translation agents are present in the network. This approach can be realized for many centralized management solutions by replicating the server and providing a shared policy repository for all servers. Another possible realization of this approach is organising translation agents hierarchically.
 - (c) **strongly distributed management** systems use a separate translation agent for each managed device. Cfengine uses a strongly distributed model. The difficulty with this approach is enforcing inter-device dependencies because each device is responsible for translating its own configuration specification, devices need to cooperate with each other to ensure consistency.
2. **deployment agent:** In all approaches, each managed device contains a deployment agent that can be push or pull based. In the case of a pull based mechanism, the software agent needs to contact the translation agent to fetch the translated configurations. In a push based mechanism, the translation agent contacts the deployment agent. Deployment agents also have to be authenticated and their capabilities for fetching policies or configurations have to be limited. Configurations often contain sensitive information like passwords or keys and exposing this information to all deployment agents introduces a security risk. Cfengine uses public-key cryptography for authentication deployment agents and allows to limit the files an agent can fetch. Bcfg2 uses certificates or a shared key to authenticate deployment agents. There is no explicit authorization mechanism present in Bcfg2 but the translation agent will only serve files that are part of a device's configuration.

Chapter 2

Bcfg2 Evaluation Report

Bcfg2 [13, 16, 14] is a tool developed at Aragonne National Laboratory with Narayan Desai as primary author. Bcfg2 is implemented in Python.

2.1 Abstraction Level

Bcfg2 operates at the configuration files level. It provides an XML-based language for specifying configurations. The specification contains references to the literal configuration files that need to be deployed. Other things that can be specified are symbolic links, packages, services, directories and permissions that need to be set. In theory, Bcfg2 could be extended to operate at a higher specification level by using its plugin mechanism.

2.2 Specification Language

2.2.1 Usability

Configuration Specification User Interface a language-based interface (with XML as metalanguage) is used for specifying policies.

Configuration Reporting User Interface Bcfg2 has very good web-based reporting capabilities. On client connection, the server gathers statistics and collects the statistics in a database which is accessible from the web-frontend. The web-frontend allows to check how many clients have out-of-sync specifications, query for configuration file diffs, . . .

2.2.2 Domain Coverage

Supported Functionality General-purpose

Supported Platforms Unix systems (Bcfg2 has been used on Linux, Solaris, FreeBSD and Mac OS X)

2.2.3 Grouping mechanism

Bcfg2 provides a grouping mechanism. Hosts can members of multiple groups and groups can inherit. Bcfg2 also allows dynamic groups. When a client contacts the server, the server executes probes on the client. Based on the information of those probes, the client becomes a member of one or more dynamic groups.

2.2.4 Multi-level Specification

Since Bcfg2 operates at the level of configuration files, a multi-level specification mechanism is not necessary.

2.3 Consistency

2.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies N/A

Parameter - Instance dependencies N/A

2.3.2 Conflict Management

Modality conflicts Bcfg2 detects a ambiguous specification of a configuration file. When different versions of a configuration file are assigned to a device, the version with the most specific group/individual host wins. If a configuration file is generated by multiple plugins, an error is generated.

Application specific conflicts N/A

2.3.3 Workflow Management

Bcfg2 does not currently includes a mechanism for workflow management. A basic workflow management system for Bcfg2 is proposed in [15] and is based on integration Bcfg2 with a subversion repository and a state transition specification.

2.4 Distributed Management

2.4.1 Federated Management

Devices N/A

Functionality N/A

Security N/A

2.4.2 Distributed Translation

Server Model Bcfg2 uses a centralized server approach. Client/server communication is done using XML-RPC over HTTPS. When duplicating the data repository, multiple servers can be used for dealing with scalability concerns

Communication Pull-based

Security Devices can be authenticated with a share secret or certificates. A device can only fetch its own configuration.

2.5 Acknowledgements

- Narayan Desai

Chapter 3

Bladelogic Evaluation Report

Bladelogic [20] provides an integrated solution for managing the data-center. It deals with configuration management, automated installation, server provisioning, regulatory compliance, monitoring and an application release management.

3.1 Abstraction Level

Bladelogic is situated at the level of configuration files. It offers a so called Network Shell which is a centralized console for administration and scripting. Languages supported in this Network Shell are Zsh, Perl and Python.

Those jobs are little scripts that change the configuration of a particular service. The scripts are dependent on the implementation of the service used. For example, when managing a web server, different scripts need to be written for Microsoft IIS and Apache.

3.2 Specification Language

3.2.1 Usability

Configuration Specification User Interface GUI and web-based interface for managing configuration specifications by scheduling configuration tasks (jobs) on systems.

Configuration Reporting User Interface There is a module which provides graphical reporting/inventory control capabilities.

3.2.2 Domain Coverage

Supported Functionality General-purpose. Focus on data-centers.

Supported Platforms Unix, Windows systems

3.2.3 Grouping mechanism

Bladelogic allows parametrization of policy jobs and a grouping mechanism based on characteristics such as operating system name/version, location, ...

3.2.4 Multi-level Specification

Since Bladelogic operates at the level of configuration files, a multi-level specification mechanism is not necessary.

3.3 Consistency

3.3.1 Dependency Modeling

Instance dependencies Bladelogic offers a limited form of dependency modeling at the level of jobs. Jobs can have annotations that specify a list of jobs that must be executed as a prerequisite.

Parameter dependencies N/A

Parameter - Instance dependencies N/A

3.3.2 Conflict Management

Modality conflicts N/A

Application specific conflicts N/A

3.3.3 Workflow Management

Bladelogic allows automatic rollback on job failure. The deployment of jobs can be staged or executed at a certain time.

3.4 Distributed Management

3.4.1 Federated Management

Devices N/A

Functionality There is an advanced role-based access control mechanism. Users can be assigned to a list of predefined roles like “allow bare-bones installation”, “enroll and decommission servers”, . . .

Security Authentication/Authorization are supported. Authentication can be done by using secure remote passwords, certificates, kerberos or active directory. Authorization uses the Role based access control mechanism and detailed ACL's.

3.4.2 Distributed Translation

Server Model Centralized-server approach. Separate server instances can be activated, while sharing same data repository.

Communication Push-based

Security Authorization: Since a push-based mechanism is used, clients can only see their own configuration. Authentication is done using certificates or passwords.

3.5 Acknowledgements

- Steve McCluskey (Bladelogic Inc.)

Chapter 4

Cfengine Evaluation Report

Cfengine [9, 7, 8, 10], or the configuration engine is an autonomous agent and a middle to policy language and agent for building expert systems to administrate and configure large computer networks. Mark Burgess is the primary author of Cfengine. Cfengine is developed at Oslo University College. Cfengine is implemented in C.

4.1 Abstraction Level

Cfengine language is declarative. The basic language allows to specify configurations at the level of configuration files. Cfengine has some general purpose primitives, for editing, copying, removing files and directories. Also it has a primitive for settings permissions on files and directories and for executing arbitrary scripts. Cfengine also has limited support for modelling configurations at the level of implementation dependent instances. For example: it has built in support for network configuration and NFS-client/server configurations. These built in language primitives enforce a best-practice configuration model.

4.2 Specification Language

4.2.1 Usability

Configuration Specification User Interface Language-based.

Configuration Reporting User Interface N/A

4.2.2 Domain Coverage

The language of Cfengine is not easy to extend. It is not designed to allow extensions in its policies at the level of implementation dependent instances. However, it can be used as a general purpose management tool by using its functionality at the level of configuration files.

Supported Functionality General-purpose

Supported Platforms Unix, Windows

4.2.3 Grouping mechanism

Cfengine has a very powerful class mechanisms with an associated boolean logic in the language to select which components are to be activated for which devices.

4.2.4 Multi-level Specification

Since Cfengine operates at the level of configuration files, a multi-level specification mechanism is not necessary.

4.3 Consistency

4.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies N/A

Parameter - Instance dependencies N/A

Dependencies could be simulated by using variables in the configuration specification.

4.3.2 Conflict Management

Modality conflicts N/A

Application specific conflicts N/A

4.3.3 Workflow Management

N/A

4.4 Distributed Management

4.4.1 Federated Management

No special authorization primitives are present for distributed policy specification. However, Cfengine could be used to assemble policy specifications from different sources.

Devices N/A

Functionality N/A

Security N/A

4.4.2 Distributed Translation

Server Model Cfengine architecture is completely distributed. Each managed hosts runs a policy agent that enforces the specified policy. Policies can be distributed by any available mechanism (scp, rsync, ...). Cfengine also includes a daemon (cfservd) that can be used for securely distributing policies. The decentralized execution model ensures maximum scalability/performance. Each managed device has its own autonomous agent that only needs to communicate with the policy distribution server.

Communication Pull

Security Communication between the policy distribution server (cfservd) and clients is encrypted and public-key cryptography is used for authentication. Clients can be restricted to a subset of the configuration specification.

Chapter 5

Firmato Evaluation Report

Firmato [6] is a toolkit to model the security policy of all firewalls network-wide, with explicit modelling of inter dependencies between different firewall devices. Firmato is implemented in C.

5.1 Abstraction Level

Firmato models firewall configurations independent of the firewall software used, it is thus located at the instance configuration level. A domain-specific Model Definition Language is used define firewall policies. This is then translated into an ER Model. Based on the ER-model, firewall-specific rules are generated.

5.2 Specification Language

5.2.1 Usability

Configuration Specification User Interface Language-based.

Configuration Reporting User Interface Firmato uses a Rule Illustrator, which transforms the firewall-specific configuration files into a graphical representation of the current policy on the actual topology. Such a visualization allows a quick first evaluation of the viability of a chosen policy.

5.2.2 Domain Coverage

Supported Functionality Firewalls

Supported Platforms Unix, Windows, Network Devices

5.2.3 Grouping mechanism

Grouping construct. End-end specification of firewall policies.

5.2.4 Multi-level Specification

N/A

5.3 Consistency

5.3.1 Dependency Modeling

Instance dependencies Models relationships between services (for example: mail client and mail servers)

Parameter dependencies N/A

Parameter - Instance dependencies N/A

5.3.2 Conflict Management

Modality conflicts Errors in specification are detected.

Application specific conflicts N/A

5.3.3 Workflow Management

N/A

5.4 Distributed Management

5.4.1 Federated Management

Devices N/A

Functionality N/A

Security N/A

5.4.2 Distributed Translation

Server Model Centralized

Communication Push

Security N/A

Chapter 6

LCFG Evaluation Report

LCFG [4, 1, 3] is a configuration management tool developed at the University of Edinburgh, with Paul Anderson as lead. LCFG is implemented in Perl.

6.1 Abstraction Level

The specification language of LCFG is composed of scoped attribute-value pairs. The system is seen as a collection of components (each responsible for generating one or more configuration files). For each of these components, a number of parameters can be set by using attribute-value assignments. The specification language of LCFG operates at the implementation dependent instances level.

6.2 Specification Language

6.2.1 Usability

Configuration Specification User Interface The specification of configurations is done using plain text files.

Configuration Reporting User Interface Clients report back their current state and a web interface displays errors, warnings, current resource values etc, and well as an indication of the current levels. This information is also available from the server via a separate API if anything wants to process it further.

6.2.2 Domain Coverage

Supported Functionality General-purpose

Supported Platforms Currently, LCFG primarily supports RPM-based Linux systems and, to some degree, Solaris systems. The specification language and infrastructure used do not need any extensions for supporting other platforms, the components that generate configuration files need to be rewritten.

6.2.3 Grouping mechanism

The structuring mechanism used for avoiding replication is the standard C-preprocessor.

6.2.4 Multi-level Specification

The translation process can be manipulated by modifying templates or re-implementing components that take care of translating the generated node profiles to configuration files.

6.3 Consistency

6.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies LCFG allows to specify dependencies between different instances running on different devices by using the principles of spanning maps (a one to many dependency) or references (a one to one dependency). An example of a spanning map is the configuration of a DHCP server which has a spanning map for referring to the MAC addresses of its DHCP clients. An example of a reference is the name property of a web server which refers to the host name. These mechanisms are limited to dependencies between parameters. L

Parameter - Instance dependencies N/A

6.3.2 Conflict Management

Modality conflicts Each value can be assigned only once, but it may be modified an unlimited number of times. C Macros are used to describe common conflict resolution strategies (e.g. add value to list, remove a value to list, ...). Component authors can extend this mechanism with arbitrary Perl code which will take two conflicting values and resolve them.

Application specific conflicts LCFG also uses the concept of validation constraints, that allow to specify constraints on a parameter. Examples of such constraints are “the value given is a hostname in DNS” or “the value given is a dotted quad IP address”. If such a constraint is broken, compilation will fail.

6.3.3 Workflow Management

N/A

6.4 Distributed Management

6.4.1 Federated Management

Devices LCFG has a mechanism for merging configuration specifications from multiple sources into a coherent specification that is then translated to individual node configurations.

Functionality LCFG has a mechanism for merging configuration specifications from multiple sources into a coherent specification that is then translated to individual node configurations.

Security N/A

6.4.2 Distributed Translation

Server Model LCFG uses a classical client-server architecture, where one server is responsible for assembling and translating a centralized configuration to node configurations. These node configurations are then fetched by the clients (who run on the managed node) and translated to configuration files by per-component back-ends. There is a huge collection of existing back-ends available for common functionality like web server configuration, firewall management, startup system configuration, ... LCFG has been tested in an environment with over 1000 machines under total control. However, because of the centralized-server approach, further scaling would require introducing more servers with a shared data repository.

Communication Pull-based

Security Authentication of clients is supported. Clients can only fetch their own configurations.

6.5 Acknowledgements

- Paul Anderson
- Ed Smith

Chapter 7

Microsoft SMS Evaluation Report

Microsoft Systems Management Server (SMS) [21] is a systems management software product by Microsoft for managing large groups of Windows-based computer systems. SMS provides remote control, patch management, software distribution, and hardware and software inventory.

7.1 Abstraction Level

The primary function of Microsoft SMS is software management and patch management. It also includes functionality for running commands/scripts on remote hosts, transferring files, restarting client computers and remote login.

Since Microsoft SMS has no knowledge of the contents of managed files, it is located at the configuration files level.

7.2 Specification Language

7.2.1 Usability

Configuration Specification User Interface Microsoft SMS uses a GUI interface.

Configuration Reporting User Interface Extensive monitoring capabilities are included for asset management (hardware and software), and software metering. A graphical user interface is used.

7.2.2 Domain Coverage

Supported Functionality General-purpose

Supported Platforms Windows

7.2.3 Grouping mechanism

Applications can be activated on hosts based on subnet, user, group characteristics.

7.2.4 Multi-level Specification

Since Microsoft SMS operates at the level of configuration files, a multi-level specification mechanism is not necessary.

7.3 Consistency

7.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies N/A

Parameter - Instance dependencies N/A

7.3.2 Conflict Management

Modality conflicts N/A

Application specific conflicts N/A

7.3.3 Workflow Management

N/A

7.4 Distributed Management

7.4.1 Federated Management

Devices N/A

Functionality N/A

Security Login-based authentication.

7.4.2 Distributed Translation

Server Model Weakly-distributed management approach. A hierarchy of parent and child servers can be created. According to the documentation, one server should be able to manage up to 25000 clients.

Communication Push-based

Security Secured Authentication of devices.

Chapter 8

Netdirector Evaluation Report

Netdirector [35] is an open source configuration management tool for managing open source applications running on many variants of Linux and Solaris. Unique features include the ability to simultaneously push changes out to many different platforms. Netdirector is implemented in Java.

8.1 Abstraction Level

Netdirector deals with changing files/running scripts and is located at the configuration files level.

8.2 Specification Language

8.2.1 Usability

Configuration Specification User Interface Web-based interface.

Configuration Reporting User Interface Status-reporting in web-based user interface.

8.2.2 Domain Coverage

Supported Functionality General-purpose

Supported Platforms Unix

8.2.3 Grouping mechanism

Devices can be grouped together.

8.2.4 Multi-level Specification

Since Netdirector is located at the configuration files level, a multi-level specification mechanism is not necessary.

8.3 Consistency

8.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies N/A

Parameter - Instance dependencies N/A

8.3.2 Conflict Management

Modality conflicts N/A

Application specific conflicts N/A

8.3.3 Workflow Management

Planning execution time of changes, rollback feature.

8.4 Distributed Management

8.4.1 Federated Management

Roles can be created which are associated with devices and services. This allows for flexible, two-dimensional (devices and functionalities) authorizations.

Devices Supported.

Functionality Supported.

Security Authentication and Authorization based on role-system and logins.

8.4.2 Distributed Translation

Server Model Central server + agents on each managed system. XML-RPC + SSL is used as a communication mechanism. The server has a database which stores configuration data, authorizations and a backlog of changes made. Multiple servers can be present, allowing weakly distributed management.

Communication Push-based

Security Bidirectional authentication based on SSL certificates. Server only pushes client configuration.

Acknowledgements

- Greg Wallace

Chapter 9

Opsware Evaluation Report

Opsware [22] has two major products: 1. Network Automation System and 2. Server Automation System. The former manages all network devices like routers, switches, bridges, ... while the latter is responsible for managing Windows/Solaris/Linux servers. These two systems provide a solution for configuration management, software management and auditing.

9.1 Abstraction Level

No special-purpose language is used. All changes are in machine configuration file format. Opsware also includes a global shell interface to execute commands on a group of servers. Opsware is situated at the configuration files level.

9.2 Specification Language

9.2.1 Usability

Configuration Specification User Interface Web-interface.

Configuration Reporting User Interface Opsware has two modes of operation: deploy and audit. In deploy mode, bundles of packages/files and scripts are distributed. Disadvantage is that those two modes share no data. A deployment policy and an audit check for that deployment policy must be written and maintained separately. The finest granularity that can be achieved in deployment mode is files. No individual parameters of configuration files can be specified. Anything beyond basic functionality has to be achieved by writing shell scripts. Audits are reported in Web-interface. Visual analysis/reporting is supported.

9.2.2 Domain Coverage

Supported Functionality Server/Infrastructure

Supported Platforms Unix, Windows, Network Devices

9.2.3 Grouping mechanism

Standard grouping construct.

9.2.4 Multi-level Specification

Since Opsware operates at the level of configuration files, a multi-level specification mechanism is not necessary.

9.3 Consistency

9.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies N/A

Parameter - Instance dependencies N/A

9.3.2 Conflict Management

Modality conflicts N/A

Application specific conflicts N/A

9.3.3 Workflow Management

Opsware can be configured to require approval of administrators before a change becomes active.

9.4 Distributed Management

9.4.1 Federated Management

Role based access control for device views, actions and system actions.

Devices Supported.

Functionality Supported.

Security Authentication (logins) and authorization (role-based system) supported.

9.4.2 Distributed Translation

Server Model Centralized server (with failover support)

Communication Push-based

Security Multiple supported device mechanisms (TFTP, SSH, ...)

Chapter 10

Puppet Evaluation Report

Puppet [23, 24] lets you centrally manage every important aspect of your system using a cross-platform specification language that manages all the separate elements normally aggregated in different files, like users, cron jobs, and hosts, along with obviously discrete elements like packages, services, and files. Puppet is developed by Luke Kanies as an independent consultant (Reductive Labs). Puppet is implemented in Ruby.

10.1 Abstraction Level

The language syntax is ruby-like. The basic concept is that of a type (similar to the OO-class concept). Types can be used to model configurations, package and service information. Puppet types can be used to create implementation dependent instance configurations and (implementation independent) instance configurations. At present, it is mainly used at the implementation dependent instance configuration level.

10.2 Specification Language

10.2.1 Usability

Configuration Specification User Interface Language-based

Configuration Reporting User Interface Can create graphs of client configurations.

10.2.2 Domain Coverage

Supported Functionality General-purpose

Supported Platforms Unix-systems

10.2.3 Grouping mechanism

Classing and subclassing mechanism, configuration chunks can be parametrized. There is also an if-else structure available to select configuration chunks.

10.2.4 Multi-level Specification

N/A

10.3 Consistency

10.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies It is possible to make available certain host properties to other hosts, which can be queried by other components and other device configurations.

Parameter - Instance dependencies N/A

10.3.2 Conflict Management

Modality conflicts N/A

Application specific conflicts N/A

10.3.3 Workflow Management

N/A

10.4 Distributed Management

10.4.1 Federated Management

Devices N/A

Functionality N/A

Security N/A

10.4.2 Distributed Translation

Server Model Strongly distributed. Most of the translation computations are done at the client side.

Communication Pull-based

Security Certificate-based. Access to files can be restricted for clients based on IP address/DNS.

Chapter 11

IBM Tivoli Evaluation Report

Tivoli [19] is in the first place a brand-name for IBM's line of IT management products. Most, but not all, of Tivoli products are integrated into the IBM Tivoli Management Framework. This framework provides the basic functionalities as building blocks for other IBM Tivoli products. The term IBM uses for the functionality of its Tivoli product line is ESM - Enterprise Systems Management. The IBM Tivoli Framework has matured from a lot of separate products, that have been gradually integrated into a uniform management framework.

11.1 Abstraction Level

At its core, Tivoli has a secure file copying mechanism and script execution mechanism. Therefore Tivoli is located at the configuration files level.

11.2 Specification Language

11.2.1 Usability

Configuration Specification User Interface IBM provides different user interfaces to its management framework. The main interface is a cross-platform GUI applications that integrates most management applications. A web-based version is also available, together with command-line utilities.

Configuration Reporting User Interface The user interfaces allow to monitor the current state of the network.

11.2.2 Domain Coverage

Supported Functionality General-purpose

Supported Platforms Unix and Windows

11.2.3 Grouping mechanism

Tivoli provides facilities for grouping devices together and applying queries/file distribution/policy configuration on groups of devices. The finest granularity that can be achieved is the distribution of files and scripts.

11.2.4 Multi-level Specification

Since Tivoli operates at the level of configuration files, a multi-level specification mechanism is not necessary.

11.3 Consistency

11.3.1 Dependency Modeling

Instance dependencies N/A

Parameter dependencies N/A

Parameter - Instance dependencies N/A

11.3.2 Conflict Management

Modality conflicts N/A

Application specific conflicts N/A

11.3.3 Workflow Management

Allows planning of task executions through “activity plans”.

11.4 Distributed Management

11.4.1 Federated Management

IBM Tivoli offers an authorization system to allow delegation of management responsibilities to different administrators. The finest granularity that can be achieved is the responsibility over files and scripts.

Devices Supported.

Functionality Supported.

Security Supported (user-password logins).

11.4.2 Distributed Translation

Server Model The server runtime is a CORBA-based framework. Agents are distributed to all managed entities. Different implementations (heavy and lightweight) of these agents exists. At its core functionality, the IBM Management Framework provides a secure remote command execution and file copy facility. Multiple Tivoli servers can communicate with each other and exchange policies, allowing a weakly distributed management model.

Communication Push

Security Since a push-based mechanism is used, clients can only see their own configuration. Authentication of clients is supported.

Chapter 12

Conclusion

The sample of configuration management tools presented in the previous chapters, classified by using our taxonomy framework includes three open-source tools developed by academia (Bcfg2, Cfengine and LCFG), five tools backed by companies (Bladelogic, Microsoft SMS, Netdirector, Opsware and IBM Tivoli Configuration Manager), one research prototype (Firmato) and one tool developed by an independent consultant (Puppet).

As a conclusion, we provide a tabular summary of the capabilities of each tool in table 12.1

	Bc12	Bladelogic	Cfengine	Firmato	LCFG	Microsoft SMS	Netdirector	Opware	Puppet	Tivoli
Abstraction level	5	5	5	3	4	5	5	5	4	5
Usability										
Specification Reporting	Language GUI	GUI GUI	Language	Language GUI	Language GUI	GUI GUI	GUI GUI	GUI GUI	Language GUI	GUI GUI
Domain Coverage	General Unix	General Unix Windows	General Unix Windows	Limited Unix Windows Network	General Unix	General Windows	General Unix	General Unix Windows Network	General Unix	General Unix Windows
Platforms										
Grouping mechanism	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multi-level specification										
Dependency Modeling										
Instance		Yes		Yes					Yes	
Parameter					Yes					
Parameter-Instance										
Conflict Management										
Modality conflicts	Yes			Yes	Yes					
Application specific					Yes					
Workflow Management	Yes	Yes					Yes	Yes		Yes
Federated Management										
Devices		Yes			Yes		Yes	Yes		Yes
Functionality					Yes		Yes	Yes		Yes
Security		Authentication Authorization				Authentication	Authentication Authorization	Authentication Authorization		Authentication Authorization
Distributed Translation										
Translation Agent	Centralized Pull	Centralized Push	Distributed Pull	Centralized Push	Centralized Pull	Weakly Distr. Push	Weakly Distr. Push	Centralized Push	Distributed Pull	Weakly Distr. Push
Communication	Authentication Authorization	Authentication Authorization	Authentication Authorization		Authentication Authorization	Authentication Authorization	Authentication Authorization	Authentication Authorization	Authentication Authorization	Authentication Authorization
Security										

Table 12.1: Taxonomy of a sample of configuration management solutions. An empty cell represents not supported functionality.

Bibliography

- [1] Paul Anderson. Lcfg homepage. <http://www.lcfg.org>.
- [2] Paul Anderson and Alva Couch. What is this thing called “system configuration”? LISA Invited Talk, November 2004.
- [3] Paul Anderson and Alastair Scobie. Large scale Linux configuration with LCFG. In USENIX, editor, *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, October 10–14, 2000, Atlanta, Georgia, USA*, pages 363–372, Berkeley, CA, USA, 2000. USENIX.
- [4] Paul Anderson and Alastair Scobie. LCFG - the Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002.
- [5] Paul Anderson and Edmund Smith. Configuration tools: Working together. In *Proceedings of the Large Installations Systems Administration (LISA) Conference*, pages 31–38, Berkeley, CA, December 2005. Usenix Association.
- [6] Yair Bartal, Alain Mayer, Kobbi Nissim, and Avishai Wool. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst.*, 22(4):381–420, 2004.
- [7] M. Burgess. Cfengine www site. <http://www.iu.hio.no/cfengine>, 1993.
- [8] M. Burgess. *GNU cfengine*. Free Software Foundation, Boston, Massachusetts, 1994-.
- [9] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.
- [10] Mark Burgess. Recent developments in cfengine. In *Unix.nl Conference Proceedings*, 2001.
- [11] Marinos Charalambides, Paris Flegkas, George Pavlou, Arosha K. Bandara, Emil C. Lupu, Alessandra Russo, Naranker Dulay, Morris Sloman, and Javier Rubio-Loyola. Policy conflict analysis for quality of service management. In *POLICY '05: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 99–108, Washington, DC, USA, 2005. IEEE Computer Society.
- [12] Alva Couch. Why people don't adopt configuration management tools. LISA 2004 Configuration Management Workshop Invited Talk, December 2004.
- [13] Narayan Desai, Rick Bradshaw, and Joey Hagedorn. Bcfg2 Trac homepage. <http://trac.mcs.anl.gov/projects/bcfg2>.
- [14] Narayan Desai, Rick Bradshaw, and Joey Hagedorn. System management methodologies with Bcfg2. *login: the USENIX Association newsletter*, 31(1), February 2006.
- [15] Narayan Desai, Rick Bradshaw, Joey Hagedorn, and Cory Lueninghoener. Directing change using bcfg2. In *Proceedings of the Large Installations Systems Administration (LISA) Conference*, pages 215–220, Berkeley, CA, December 2006. Usenix Association.

- [16] Narayan Desai, Rick Bradshaw, Scott Matott, Sandra Bittner, Susan Coghlan, Rémy Evard, Cory Luninghoener, Ti Leggett, John-Paul Navarro, Gene Rackow, Craig Stacey, , and Tisha Stacey. A case study in configuration management tool deployment. In *Proceedings of the Large Installations Systems Administration (LISA) Conference*, pages 39–46, Berkeley, CA, December 2005. Usenix Association.
- [17] Hubert Feyrer. g4u homepage. <http://www.feyrer.de/g4u/>.
- [18] Zhi (Judy) Fu and S. Felix Wu. Automatic generation of ipsec/vpn security policies in an intra-domain environment, 2001.
- [19] IBM. IBM Tivoli homepage. <http://www.ibm.com/software/tivoli>.
- [20] Bladelogic Inc. Bladelogic homepage. <http://www.bladelogic.com>.
- [21] Microsoft Inc. Microsoft SMS homepage. <http://www.microsoft.com/smsserver>.
- [22] Opsware Inc. Opsware network automation system and opsware server automation system. <http://www.opsware.com>.
- [23] Luke Kanies. Puppet. <http://reductivelabs.com/projects/puppet/>.
- [24] Luke Kanies. Puppet: Next-generation configuration management. *login: the USENIX Association newsletter*, 31(1), February 2006.
- [25] Eliot Lear. Network complexity: How do I manage all of this? LISA Invited Talk, November 2004.
- [26] Tom Limoncelli. What I've learned from avoiding configuration management. LISA 2005 Configuration Management Workshop Invited Talk, December 2005.
- [27] E. Lupu and M. Sloman. Conflict analysis for management policies. In *Proceedings of the Vth International Symposium on Integrated Network Management IM'97*, pages 1–14. Chapman & Hall, May 1997.
- [28] Jean-Philippe Martin-Flatin, Simon Znaty, and Jean-Pierre Hubaux. A survey of distributed enterprise network and systems management paradigms. *J. Netw. Syst. Manage.*, 7(1):9–26, 1999.
- [29] Jonathan D. Moffett. Requirements and policies. In *Proceedings of the Policy Workshop*, November 1999.
- [30] Sanjai Narain. Towards a foundation for building distributed systems via configuration. <http://www.argreenhouse.com/papers/narain/Service-Grammar-Web-Version.pdf>, 2004.
- [31] D. Oppenheimer. The importance of understanding distributed system configuration. In *Proceedings of the 2003 Conference on Human Factors in Computer Systems workshop*, April 2003.
- [32] Partimage homepage. <http://www.partimage.org>.
- [33] D. Raymer, J. Strassner, E. Lehtihet, and S van der Meer. End-to-end model driven policy based network management. In *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop*, page 4, 2006.
- [34] Edmund Smith and Paul Anderson. Toward broad-spectrum autonomic management. In *Proceedings of the The Sixth International Conference on Networking (ICN 2007)*, 2007. to be published.
- [35] Emu software. Netdirector homepage. <http://www.netdirector.org/>.
- [36] Symantec. Norton Ghost Homepage. <http://www.symantec.com/ghost>.

- [37] D.C. Verma. Simplifying network administration using policy-based management. *IEEE Network*, 16(2):20–26, Mar/Apr 2002.
- [38] Ning Wu and Alva Couch. Autonomic configuration transition planning in polynomial time. submitted, 2007.