A survey of system configuration tools

Thomas Delaet *Wouter Joosen*
Bart Vanbrabant
DistriNet, Dept. of Computer Science
K.U.Leuven, Belgium
{*thomas.delaet, wouter.joosen, bart.vanbrabant*}*@cs.kuleuven.be*

Abstract

We believe that informed choices are better choices. When you adopt a system configuration tool, it implies a significant investment in time and/or money. Before making such an investment, you want to pick the right tool for your environment. Therefore, you want to compare tools with each other before making a decision. To help you make an informed choice, we develop a comparison framework for system configuration tools. We evaluate 11 existing open-source and commercial system configuration tools with this framework. If you use our framework, you will make a better choice in less time.

1 Introduction

When you adopt a system configuration tool, it implies a significant investment in time and/or money. Before making such an investment, you want to know you have picked the right tool for you environment. Therefore, you want to compare tools with each other before making a decision.

Since there exist a lot of tools with different goals, characteristics and target users, it is a difficult and time-intensive task to make an objective comparison of system configuration tools. Moreover, people using a tool already made a significant investment in that tool (and not others) and as a consequence are involved in that tool. But they themselves have difficulty comparing their "own" tool to other tools.

To help you make an informed choice, we developed a comparison framework for system configuration tools. In addition to more subjective or political decision factors, this framework can help you with the more objective factors when selecting a system configuration tool that is right for you. The framework consists of four categories of properties.

1. Properties related to the input specification
2. Properties related to deploying the input specification
3. Process-oriented properties
4. Tool support properties

We evaluated 11 existing open-source and commercial system configuration tools with our framework. This paper contains a summary of these evaluations. The full evaluations are available on our website at `http://distrinet.cs.kuleuven.be/software/sysconfigtools`. You can comment on these evaluations, provide suggestions for modifications or add your own evaluations.

The remainder of this paper is structured as follows: We start with the description of the framework in Section 2. Next, we summarize our findings for the 11 tools we evaluated in Section 3. Section 4 answers the questions on how to choose a tool and how to evaluate another tool using the framework. In Section 5, we use our framework and the evaluations to analyze the gaps in the state of the art. Section 6 concludes the paper.

2 The comparison framework

Every system configuration tool provides an interface to the system administrator. Within this interface, the system administrator expresses the configuration of the devices managed by the tool. The tool uses this specification as input and enforces it on all machines it manages. This conceptual architecture of a system configuration tool is illustrated in Figure 1.

In Figure 1, the system administrators inputs the desired configuration of the devices managed by the tool. This input it stored in a repository. The tool uses this input to generate device-specific profiles that are enforced on every managed device. The translation agent is the component of the tool that translates the system administrator input to device-specific profiles. The deployment

agent is the component of the tool that runs on the managed device and executes the generated profile.

Our comparison framework contains properties for both the specification of the input and the enforcement phase. The third type of properties that are present in our comparison framework are meta-specification properties: how does a tool deal with managing the input specification itself? The last type of properties deal with tool support: How easy is it to adopt the tool?

2.1 Specification properties

2.1.1 Specification paradigm

We define the specification paradigm of a tool by answering two questions:

1. Is the input language declarative or imperative?
2. Does the tool use a GUI-based or command-line user interface?

Tools that use a declarative input language enable to express the desired state of the computer infrastructure. The runtime of the tool compares this desired state with the configuration on every managed device and derives a plan to move to the desired state. In the system configuration literature, this process is described as *convergence* [1]. A system configuration tool that supports *convergence* has the additional benefit that divergences from the desired state are automatically corrected.

Tools that use an imperative input language distribute, schedule and deploy scripts written in its imperative input language on the managed devices. For an imperative script to work reliable, all possible states of the managed devices need to covered and checked in the script. Moreover, the system configuration tool must also keep track of what scripts are already executed on every device. An alternative is to make all the operations in the script idempotent.

Let us contrast the practical differences between an imperative and a declarative language. Suppose a system administrator does not want file `/etc/hosts_deny` to be present on a device.

In a declarative language, the system administrator must ensure that the file is not included in the model or explicitly define that the file must not exist.

In an imperative language, the system administrator must first write a test to verify if `/etc/hosts_deny` exists. If the file exists, another instruction is needed to remove the file. If the system administrator does not write the first test, the action fails if the file was already removed.

Orthogonal on the choice of declarative or imperative specification language is the choice of user interface: does the tool use a command-line or graphical user interface?

Command-line interfaces typically have a steeper learning curve than graphical approaches but, once mastered, can result in higher productivity. Command-line interfaces also have the advantage that they can be integrated with other tools through scripting. In contrast, system administrators are typically quicker up to speed with graphical approaches [12].

2.1.2 Abstraction mechanisms

A successful configuration tool is able to make abstraction of the complexity and the heterogeneity that characterises IT infrastructures where hardware and software of several vendors and generations are used simultaneously [3]. Making abstraction of complexity and heterogeneity is very similar to what general purpose programming languages have been doing for decades.

Abstraction from complexity is an important concept in programming paradigms such as object orientation. In object orientation, implementation details are encapsulated behind a clearly defined API. Encapsulation is a concept that is valuable for modeling configurations as well. Responsibilities and expertise in a team of system administrators are not defined on machine boundaries, but based on subsystems or services within the infrastructure, for example: DNS or the network layer. Encapsulation enables experts to model an aspect of the configuration and expose a well documented API to other system administrators.

Modern IT infrastructures are very heterogeneous environments. Multiple generations of software and hardware of several vendors are used in production at the same time. These heterogeneous "items" need to be configured to work together in one infrastructure.

Based on how a system configuration tool's language deals with complexity and heterogeneity, we define six levels to classify the tool. These levels range from high-level end-to-end requirements, to low-level bit-configurations. [3] inspired us in the definition of these levels.

1. **End-to-end requirements**: End-to-end requirements are typical non-functional requirements [23]. They describe service characteristics that the computing infrastructure must achieve. Figure 2 shows an example of a performance characteristic for a mail service. Other types of end-to-end requirements deal with security, availability, reliability, usability, ... One example of an approach that deals with end-to-end requirements is given in [17]. [17] uses first-order logic for expressing end-to-end requirements.

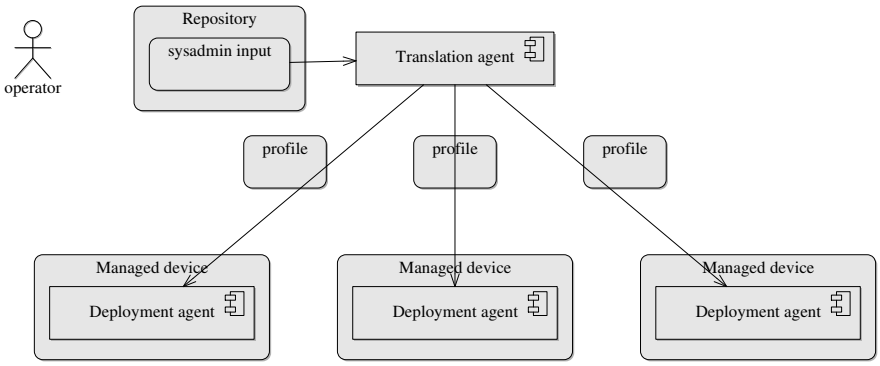

Figure 1: A conceptual architecture of system configuration tool.

2. **Instance distribution rules**: Instance distribution rules specify the distribution of instances in the network. 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 2, the instance distribution rule prescribes the number of mail servers that need to be activated in an infrastructure. The need for such a language is explicited in [3] 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 tool at this level is Firmato [6]. Firmato allows modeling firewall configurations independent from 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 2 a sendmail.cf file is used to describe the configuration of mail server instances.

5. **Configuration files**: At the level of configuration files, complete configuration files are mapped on a device or set of devices. In contrast with the previous level, this level has no knowledge of the contents of a configuration file.

6. **Bit-configurations**: At the level of Bit-configurations, disk images or diffs 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 tools that operate on this level are imaging systems like Partimage [21], g4u [9] and Norton Ghost [24].

Figure 2 shows the six abstraction levels for system configuration, illustrated with an email setup. The illustration in Figure 2 is derived from an example discussed in [3]. The different abstraction levels are tied to the context of system configuration. 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 [16, 25]. 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 system configuration gives a better insight in the different abstraction levels used by system configuration tools.

In conclusion, a system configuration tool automates the deployment of configuration specifications. At the level of bit-configurations, deployment is simply copying bit-sequences to disks, while deploying configurations specified as end-to-end requirements is a much more complex process.

2.1.3 Modularization mechanisms

One of the main reason system administrators want to automate the configuration of their devices is to avoid repetitive tasks. Repetitive tasks are not cost efficient. Moreover, they raise the chances of introducing errors. Repetitive tasks exist in a computer infrastructure because there are large parts of the configuration that are shared between a subset (or multiple overlapping subsets) of devices ([3]). For example, devices need the same DNS client configuration, authentication mechanism, shared file systems, ... A system configuration tool

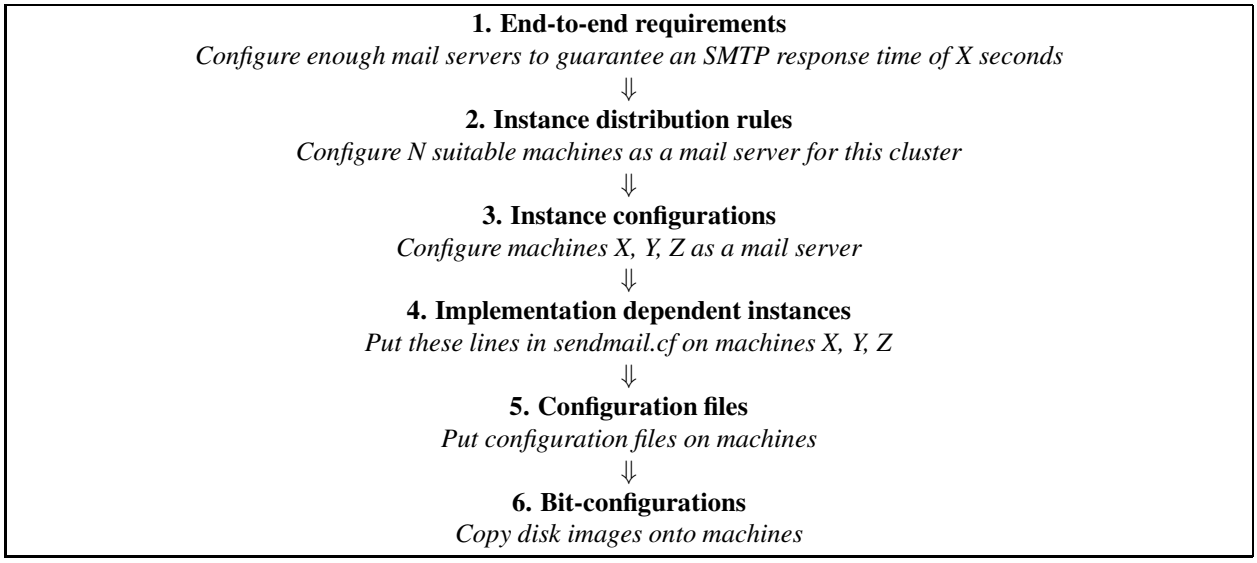

Figure 2: An example of different abstraction levels of configuration specification for an email setup.

that supports the modularization of configuration chunks reduces repetition in the configuration specification.

In its most basic form, modularization is achieved through a grouping mechanism: a device A is declared to be a member of group X and as a consequence inherits all system configuration chunks associated with X. More advanced mechanisms include query based groups, automatic definition of groups based on environmental data of the target device and hierarchical groups.

An additional property of a modularization mechanism is whether it enables third parties to contribute partial configuration specifications. Third parties can be hardware and software vendors or consultancy firms. System administrators can then model their infrastructure in function of the abstractions provided by the third-party modules and reuse the expertise or rely on support that a third party provides on their configuration modules.

2.1.4 Modeling of relations

One of the largest contributors to errors and downtime in infrastructures are wrong configurations [19, 20, 22] due to human error. An error in a configuration is commonly caused by an inconsistent configuration. For example, a DNS service that has been moved to an other server or moving an entire infrastructure to a new IP range. Explicitly modeling relations that exist in the network helps keeping a configuration model consistent.

Modeling relations is, like the modularization property of Section 2.1.3, a mechanism for minimizing redundancy in the configuration specification. When relations are made explicit, a tool can automatically change configurations that depend on each other. For example, when the location of a DNS server changes and the relation between the DNS server and clients is modeled in the configuration specification, a system configuration tool can automatically adapt the client configurations to use the new server. Again, modeling relations reduces the possibility of introducing errors in the configuration specification.

To evaluate how well a tool supports modeling of relations, we describe two orthogonal properties of relations: their granularity and their arity.

1. **granularity**: In Section 2.1.2, 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 relations in three categories: (1) relations between instances, (2) relations between parameters and (3) relations between a parameter and an instance.

   (a) **Instance relations** 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 (for example: /etc/init.d), the DNS server instance will not work. An example of dependencies between instances on different devices

is the dependency between DNS servers and their clients.

(b) **Parameter relations** 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.

(c) **Parameter - instance relations** are used to express a relation between an individual parameter and an instance. For example a mail server depends on the existence of an MX record in the DNS server.

Note that it depends on the abstraction level of a tool which dependencies it can support. The two lowest abstraction layers in Figure 2, configuration files and bit-configurations, have no knowledge of parameters and as a consequence, they can only model instance dependencies.

2. **arity**: Relations can range from one-to-one to many-to-many relationships. A simple one-to-one relationship is a middleware platform depending on a language runtime. A many-to-many relationship is for example the relation between all DNS clients and DNS servers in a network. A system configuration tool can also provide support facilities to query and navigate relations in the system configuration specification. An example that motivates such facilities for navigating and querying relations involves an Internet service. For example, a webservice runs on a machine in the DMZ. This DMZ has a dedicated firewall that connects to the Internet through an edge router in the network. The webservice configuration has a relation to the host it is running on and a relation to the "Internet". The model also contains relations that represent all physical network connections. Using these relations, a firewall specification should be able to derive firewall rules for the webservice host, the DMZ router and the edge router [6].

An extra feature is the tool's ability to support the modeling of constraints on relations. We distinguish two types of constraints: validation constraints and generative constraints.

1. **validation constraints** are expressions that need to hold true for your configuration. Because of policy or technical factors, the set of allowable values for a relation can be limited. Constraints allow to express these limitations. Examples of such limitations are:

   - A server can only serve 100 clients.
   - Clients can only use the DNS server that is available in their own subnet.
   - Every server needs to be configured redundantly with a master and a slave server.

2. **generative constraints** are expressions that leave a degree of freedom between a chunk of configuration specification and the device on which this chunk needs to be applied. Languages without support for generative constraints need a 1-1 link between a chunk of configuration specification and the device on which is needs to be applied. Languages with support for generative constraints leave more degrees of freedom for the tool. An example of a generative constraint is: "One of the machines in this set of machines needs to be a mail server".

2.2 Deployment properties

2.2.1 Scalability

Large infrastructures are subject to constant change in their configuration. System configuration tools must deal with these changes and be able to quickly enforce the configuration specification, even for large infrastructures with thousands of nodes, ten thousands of relations and millions of parameters.

Large infrastructures typically get more benefit of using a higher level specification (see Figure 2). However, the higher-level the specification, the more processing power is needed to translate this high level specification to enforceable specifications on all managed devices. System configuration tools must find efficient algorithms to deal with this problem or restrict the expressiveness of the system configuration tool.

2.2.2 Workflow

Workflow management deals with planning and execution of (composite) changes in a configuration specification. Changes can affect services distributed over multiple machines and with dependencies on other services [3, 18].

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.

A last aspect of workflow management is non-technical: if the organizational policy is to use maintenance windows for critical devices, the tool must understand that changes to these critical devices can influence the planning and execution of changes on other devices.

2.2.3 Deployment architecture

The typical setup of a system configuration tool is illustrated in Figure 1. A system configuration tool starts from a central specification for all managed devices. Next, it (optionally) processes this specification to device profiles and distributes these profiles (or the full specification) to every managed device. An agent running on the device then enforces the device's profile. For the rest of this section, we define the processing step from a central specification to device profiles as the translation agent. The agent running on every device is defined as the deployment agent.

System configuration tools differentiate their deployment architecture along two axises: 1. the architecture of the translation agent and 2. whether they use pull or push technology to distribute specifications .

1. **architecture of translation agent**: Possible approaches for the architecture of the translation agent can be classified in three categories, based on the number of translation agents compared to the number of managed devices: centralized management, weakly distributed management and strongly distributed management [15].

   (a) **centralized management** is the central server approach with only one translation agent. When dealing with huge networks, the central server quickly becomes a bottleneck. This is certainly the case when a system configuration tool uses a high-level abstraction, as the algorithm for computing a device's configuration will become 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 tools by replicating the server and providing a shared policy repository for all servers. Another possible realization of this approach is organizing translation agents hierarchically.

   (c) **strongly distributed management** systems use a separate translation agent for each managed device. The difficulty with this approach is enforcing inter-device relations because each device is responsible for translating its own configuration specification. As a consequence, devices need to cooperate with each other to ensure consistency.

2. **push or pull**: 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 deployment 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.

2.2.4 Platform support

Modern infrastructures contain a variety of computing platforms: Windows/Unix/Mac OS X servers, but also desktop machines, laptops, handhelds, smartphones and network equipment. Even in relatively homogeneous environments, we can not assume that all devices run the same operating system: operating systems running on network equipment are fundamentally different than those running on servers/desktops and smartphones are yet another category of operating systems.

Good platform support or interaction with other tools is essential for reducing duplication in the configuration specification. Indeed, many relations exist between devices running different operating systems. For example: a server running Unix and a router/firewall running Cisco IOS. If different tools are used to manage the server and router, relations between the router and server need to be duplicated in both tools which in turn introduces consistency problems if one of the relations changes. An example of such a relation is the the firewall rule on a Cisco router that opens port 25 and the SMTP service on a Unix server.

2.3 Specification management properties

2.3.1 Usability

We identify three features concerning usability of a system configuration tool: 1. ease of use of the language, 2. support for testing specifications and, 3. monitoring the infrastructure.

1. **ease of use of the language**: The target audience of a system configuration tool are system administrators. The language of the system configuration tool should be powerful enough to replace their existing tools, which are mostly custom tools. But it should also be easy enough to use, so the average system administrator is able to use it. Good system administrators with a good education [13] are already scarce, so a system configuration tool should not require even higher education.

2. **support for testing specifications**: To understand the impact of a change in the specification, the system configuration tool can provide support for testing specifications through something as trivial as a dry-run mode or more complex mechanisms like the possibility to replicate parts of the production infrastructure in a (virtualized) testing infrastructure and testing the changes in that testing infrastructure first [5].

3. **monitoring the infrastructure**: A system configuration tool can provide an integrated (graphical) monitoring system and/or define a (language-based) interface for other tools to check the state of an infrastructure. A language-based interface has the advantage that multiple monitoring systems can be connected with the system configuration tool. A monitoring system enables the user to check the current state of the infrastructure and the delta with the configuration specification.

2.3.2 Versioning support

Some system configuration tools store their specification in text files. For those tools, a system configuration specification is essentially code. As a consequence, the same reasoning to use a version control system for source code applies. It enables developers and system administrators to document their changes and track them through history. In a configuration model this configuration history can also be used to rollback configuration changes and it makes sure an audit trail of changes exists.

The system configuration tool can opt to implement versioning of configuration specification using a custom mechanism or, when the specification is in text files, reuse an external version control system and make use of the hooks most generic version control systems provide.

2.3.3 Specification documentation

Usability studies [4, 12] show that a lot of time of a system administrator is spent on communication with other system administrators. These studies also show that a lot of time is lost because of miscommunication, where discussions and solutions are based on wrong assumptions. A system configuration tool that supports structured documentation can generate documentation from the system configuration specification itself and thus remove the need to keep the documentation in sync with the real specification.

2.3.4 Integration with environment

The infrastructure that is managed by the system configuration tool is not an island: it is connected to other networks, is in constant use and requires data from other sources than the system configuration specification to operate correctly. As a consequence, a system administrator may need information from external databases in its configuration specification (think LDAP for users/groups) or information about the run-time characteristics of the managed nodes. A system configuration tool that leverages on these existing sources of information integrates better with the environment in which it is operating because it does not require all existing information to be duplicated in the tool.

2.3.5 Conflict management

A configuration specification can contain conflicting definitions, so a system configuration tool should have a mechanism to deal with conflicts. Despite the presence of modularization mechanisms and relations 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. We distinguish two types of conflicts: application specific conflicts and contradictions in the configuration specification, also called modality conflicts [14].

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 in the configuration specification. Examples of research on application specific protocols can be found in [10] and [7], 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 in the configuration specifications.

When a configuration specification contains rules that cause a conflict, this conflict should be detected and acted upon.

2.3.6 Workflow enforcement

In most infrastructures a change to the configuration will never be deployed directly on the infrastructure. A policy describes which steps each update need to go through before it can be deployed on the production infrastructure. These steps can include testing on a development infrastructure, going through Q&A, review by a security specialist, testing on a exact copy of the infrastructure and so on. Exceptions on such policies can exist because not every update can go through all stages, updates can be so urgent that they need to be allowed immediately, but only with approval of two senior managers. A system configuration tool that provides support for modeling these existing workflows can adapt itself to the habits and processes of the system administrators and will thus be easier to use than system configuration tools without this support.

2.3.7 Access control

If an infrastructure is configured and managed based on a system configuration specification, control of this specification implies control of the full infrastructure. So it might be necessary to restrict access to the configuration specification. This is a challenge, especially in large infrastructures where a lot of system administrators with different responsibilities need to make changes to this specification. A lot of these large infrastructures are also federated infrastructures, so one specification can be managed from different administrative domains.

Authenticating and authorizing system administrators before they are making changes to the system configuration can prevent a junior system administrator who is only responsible for the logging infrastructure to make changes to other critical software running on the managed devices.

Many version control systems can enforce access control but the level on which the authorisation rules are expressed differs from the abstraction level of the specification itself. In most systems, this is based on the path of the file that contains the code or specification. But in most programming languages and system configuration tools, the relation between the name of the file and the contents of the file is very limited or even non-existing. For example an authorisation rule could express that users of the *logging* group should only set parameters of object from types in the logging namespace. With path-based access control this becomes: users of group logging should only access files in the */config/logging* directory. The latter assumes that every system administrator uses the correct files to store configuration specifications.

2.4 Support

2.4.1 Available documentation

To quickly gain users, tools have to make their barriers to entry as low as possible. A "ten minutes" tutorial is often invaluable to achieve this. When users get more comfortable with the tool, they need extensive reference documentation that describes all aspects of the tool in detail alongside documentation that uses a more process-oriented approach covering the most frequent use cases.

Thus, documentation is an important factor in the adoption process of a tool.

2.4.2 Commercial support

Studies [13] show that the need for commercial support varies amongst users. Unix users don't call support lines as often as their Window-colleagues. The same holds true for training opportunities. In all cases, the fact that there is a company actively developing and supporting a tool helps to gain trust amongst system administrators and thus increases adoption.

2.4.3 Community

In our online society, community building is integral part of every product or service. Forums, wiki's and social networks can provide an invaluable source of information that complements the official documentation of a tool and introduces system administrators to other users of their preferred tool.

2.4.4 Maturity

Some organizations prefer new features above stability, and others value stability higher than new features Therefore, it is important to know what the maturity of the tool is: Is it a new tool with some cutting edge features and frequent syntax changes in its language or a well-established tool with infrequent updates?

3 System configuration tools comparison

In this section we provide a summary of our evaluation of eleven tools. These tools consist of commercial and open-source tools. The set of commercial tools is based

Tool	Version
BCFG2	1.0.1
Cfengine 3	3.0.4
Opscode Chef	0.8.8
Puppet	0.25
LCFG	20100503
BMC Bladelogic Server Automation Suite	8
CA Network and Systems Management (NSM)	R11.x
IBM Tivoli System Automation for Multiplatforms	4.3.1
Microsoft Server Center Configuration Manager (SCCM)	2007 R2
HP Server Automation System	2010/08/12
Netomata Config Generator	0.9.1

Table 1: Version numbers of the set of evaluated tools.

on market research reports [8, 11] and consists of BMC Bladelogic Server Automation Suite, Computer Associates Network and Systems Management, IBM Tivoli System Automation for Multiplatforms, Microsoft System Center Configuration Manager and HP Server Automation System. For the open-source tools we selected a set of tools that were most prominently present in discussions at the previous LISA edition and referenced in publications. This set of tools consists of BCFG2, Cfengine3, Chef, Netomata, Puppet and LCFG.

Due to space constraints we limit the results of our evaluation to a summary of our findings for each property. The full evaluation of each tool is available on our website at `http://distrinet.cs.kuleuven.be/software/sysconfigtools`. We intend to keep the evaluations on this website in sync with major updates of each tool. For this paper we based our evaluation on the versions of each tool listed in Table 1.

3.1 Specification properties

3.1.1 Specification paradigm

Language type Cfengine, Puppet, Tivoli, Netomata and Bladelogic use a declarative DSL for their input specification. BCFG2 uses a declarative XML specification. Chef on the other hand uses an imperative ruby DSL. LCFG uses a DSL that instantiates components and set parameters on them. CA NSM, HP Server Automation and MS SCCM are like LCFG limited to setting parameters on their primitives.

User interface As with the language type, the tools can be grouped in open-source and commercial tools. The open-source tools focus on command-line interface while the commercial tools also provide a graphical interfaces. Tools such as Cfengine, Chef and Puppet provide a web-interface that allows to manage some aspects with a graphical interface. In the commercial tools all management is done through coommand-line and graphical interfaces.

3.1.2 Abstraction mechanisms

3.1.3 Modularization mechanisms

Type of grouping All tools provide a grouping mechanism for managed devices or resources. HP Server Automation, Tivoli and Netomata only provide static grouping. CA NSM and BCFG allow static grouping and hierarchies of groups. LCFG supports limited static, hierarchical and query based grouping through the C-preprocessor. Bladelogic supports static, hierarchical and query based groups. Cfengine and Puppet use the concept of classes to group configuration. Classes can include other classes to create hierarchies. Cfengine can assign classes statically or conditionally using expressions. Puppet can assign classes dynamically using external tools. Chef and MS SCCM can define static groups and groups based on queries.

Configuration modules BCFG, HP Server Automation, MS SCCM and Netomata have no support for configuration modules. Bladelogic can parametrise resources based on node characteristics to enable reuse. Tivoli includes sets of predefined policies that can be used to manage IBM products and SAP. LCFG can use third party components that offer a key-value interface to other policies, CA NSM provides a similar approach for third party agents that manage a device or subsystem. Cfengine uses bundles, Chef uses cookbooks and Puppet uses modules to distribute a reusable configuration specification for managing certain subsystems or devices.

3.1.4 Modeling of relations

BCFG, CA NSM, HP Server Automation and MS SCCM have no support for modeling relations in a configuration specification. Bladelogic can model one-to-one dependencies between scripts that need to be executed as a prerequisite, these are instance relations. Cfengine supports one-to-one, one-to-many and many-to-many relations between instances, parameters and between parameters and instances. On these relations generative constraints can be expressed. Chef can express many-to-many dependency relations between instances. Tivoli can also express relations of all arities between instances and parameters and just like Cfengine express generative

constraints. LCFG can express one-to-one and many-to-many relations using spanning maps and references between instances and parameters. Netomata can model one-to-one network links and relations between devices. Finally Puppet can define one-to-many dependency relations between instances. The virtual resource functionality can also be used to define one-to-many relations between all instances.

3.2 Deployment properties

3.2.1 Scalability

The only method to evaluate how well a tool scales is to test each tool in a deployment and scale the number of managed nodes. In this evaluation we did not do this. To have an indication of the scalability we searched for cases of real-life deployments and divided the tools in three groups based on the number of managed devices and a group of tools for which no deployment information was available.

less than 1000 BCFG2

between 1000 and 10k LCFG and Puppet

more than 10k Bladelogic and Cfengine,

unknown CA NSM, Chef, HP Server Automation, Tivoli, MS SCCM and Netomata,

3.2.2 Workflow

BMC Bladelogic and HP Server Automation integrate with an orchestration tool to support coordination of distributed changes. Cfengine and Tivoli can coordinate distributed changes as well. MS SCCM and CA NSM support maintenance windows. Distributed changes in Puppet can be sequenced by exporting and collecting resources between managed devices. BCFG2, LCFG, Chef and Netomata have no support for workflow.

3.2.3 Deployment architecture

Translation agent Cfengine uses a strongly distributed architecture where the emphasis is on the agents that run on each managed device. The central server is only used for coordination and for policy distribution. Bladelogic, CA NSM and MS SCCM use one or more central servers. BCFG2, Chef, HP Server Automation, Tivoli, Netomata and Puppet use a central server. Chef and Puppet can also work in a standalone mode without central server to deploy a local specification.

Tool	Platform support
BCFG2	*BSD, AIX, Linux, Mac OS X and Solaris
Cfengine 3	*BSD, AIX, HP-UX, Linux, Mac OS X, Solaris and Windows
Opscode Chef	*BSD, Linux, Mac OS X, Solaris and Windows
Puppet	*BSD, AIX, Linux, Mac OS X, Solaris
LCFG	Linux (Scientific Linux)
BMC Bladelogic Server Automation Suite	AIX, HP-UX, Linux, Network equipment, Solaris and Windows
CA Network and Systems Management (NSM)	AIX, HP-UX, Linux, Mac OS X, Network equipment, Solaris and Windows
IBM Tivoli System Automation for Multiplatforms	AIX, Linux, Solaris and Windows
Microsoft Server Center Configuration Manager (SCCM)	Windows
HP Server Automation System	AIX, HP-UX, Linux, Network equipment, Solaris and Windows
Netomata Config Generator	Network equipment

Table 2: Version information for the set of evaluated tools.

Distribution mechanism The deployment agent of BCFG2, Cfengine, Chef, LCFG, MS SCCM and Puppet pull their specification from the central server. Bladelogic, CA NSM, HP Server Automation and Tivoli push the specification to the deployment agents. The central servers of Chef, MS SCCM and Puppet can notify the deployment agents that a new specification can be pulled. Netomata relies on external tools for distribution.

3.2.4 Platform support

The platforms that each tool supports is listed in Table 2.

3.3 Specification management properties

3.3.1 Usability

Usability Usability is a very hard property to quantify. We categorised the tools in easy, medium and hard. We determined this be assessing how easy a new user would be able to use and learn a tool. We tried to be as objective as possible to determine this but this part of the

evaluation is subjective. We found Bladelogic, CA NSM, HP Server Automation, Tivoli and MSCCM easy to start using. The usability of Cfengine, LCFG and Puppet is medium, partially because of the custom syntax. Puppet also has a lot of confusing terminology but tools such as puppetdoc and puppetca make up for it so we did not classify it as hard to use. We found BCFG2 hard to use because of the XML input and the specification is distributed in a lot of different directories because of their plugin system. Chef is also hard to use because of its syntax and the use of a lot of custom terminology. Netomata is also hard to use because of its very concise syntax but powerful language.

Support for testing specifications BCFG2, Cfengine, LCFG and Puppet have a dry run mode. Netomata is inherently dry-run because it has no deployment part. Chef and Puppet support multiple environments such as testing, staging and production.

Monitoring the infrastructure BCFG2, Bladelogic, HP Server Automation, CA NSM, Tivoli, LCFG, Puppet and MS SCCM have various degrees of support for reporting about the deployment and collecting metrics from the managed devices. The commercial tools have more extensive support for this. Chef, LCFG, Puppet and Netomata can automatically generate the configuration for monitoring systems such as Nagios.

3.3.2 Versioning support

BCFG2, Bladelogic, Cfengine, Chef, Tivoli, LCFG, Netomata and Puppet use a textual input to create their configuration specification. This textual input can be managed in an external repository such as subversion or git. CA NSM and MS SCCM have internal support for policy versions. The central Chef server also maintains cookbook version information. For HP Server Automation it is unclear what is supported.

3.3.3 Specification documentation

BCFG2, Bladelogic, Chef, HP Server Automation, Tivoli, LCFG, Netomata and Puppet specifications can include free form comments. Cfengine can include structured comments that are used to generate documentation. Because Chef uses a Ruby DSL, Rdoc can also be used to generated documentation from structured comments. Puppet can generate reference documentation for built-in types from the comments included in the source code. No documentation support is available in CA NSM and MS SCCM.

3.3.4 Integration with environment

BCFG2, Cfengine, Chef, Tivoli, LCFG, MS SCCM and Puppet can discover runtime characteristics of managed devices which can be used when the profiles of each device are generated. Bladelogic can interact with external data sources like Active Directory.

3.3.5 Conflict management

BCFG and Puppet can detect modality conflict such as a file managed twice in a specification. Cfengine3 also detects modality conflicts such as an instable configuration that does not converge. Bladelogic and CA NSM have no conflict management support. Puppet also supports modality conflicts by allowing certain parameters of resources to be unique within a device, for example the filename of file resources.

3.3.6 Workflow enforcement

None of the evaluated tools have integrated support for enforcing workflows on specification updates. Bladelogic can tie in a change management system that defines workflows.

3.3.7 Access control

The tool that support external version repositories can reuse the path based access control of that repository. BMC, CA NSM, HP Server Automation, Tivoli, MS SCCM and the commercial version of Chef allow fine grained access control on “resources” in the specification.

3.4 Support

3.4.1 Available documentation

Bladelogic, CA NSM and HP Server Automation provide no public documentation. IBM Tivoli provides extensive documentation in their evaluation download. BCFG2, Cfengine, Chef, LCFG, MS SCCM and Puppet all provide extensive reference documentation, tutorials and examples on their websites. Netomata provides limited examples and documentation on their website and Wiki.

3.4.2 Commercial support

Not very surprising the commercial tools all provide commercial support. But most open-source tools also have a company behind them that develops the tool and provides commercial support. LCFG and BCFG2 have both been developed in academic institutes and have no commercial support.

3.4.3 Community

Cfengine, Chef, Tivoli, MS SCCM and Puppet have large and active communities. BCFG2 has a small but active community. CA NSM has a community but it is very scattered. BMC, Netomata and LCFG have small and not very active communities. For HP Server Automation we were unable to determine if a community exists.

3.4.4 Maturity

Some of the evaluated tools such as Tivoli and CA NSM are based on tools that exist for more than ten years, while other tools such as Chef and Netomata are as young as two years. However no relation between the feature set of a tool and their maturity seems to exist.

4 Putting the framework to use

4.1 How do I choose a tool for my environment?

Our framework and tool evaluations can help you to quickly trim down the list of tools to the tools that match your requirements. You list your required features, see which tools support these features and you have a limited list of tools to continue evaluating. In fact, our website at `http://distrinet.cs.kuleuven.be/software/sysconfigtools` provides a handy wizard to help you with this process.

The limitation of our framework is that it can not capture all factors that influence the process for choosing a system configuration tool: 1. We limit our evaluation to system configuration and do not include adjacent processes like provisioning, 2. Politics often play an important role when deciding on a tool, 3. your ideal solution might be too pricey, or 4. other, more subjective, factors come into play.

For all these reasons, we see our framework more as an aid that can quickly give you a high-level overview of the features of the most popular tools. Based on our framework, you can decide which tools deserve more time investment in your selection process.

4.2 How do I evaluate another tool using this framework?

We welcome clarifications to our existing evaluations and are happy to add other tool evaluations on the website. Internally, the website defines our framework as a taxonomy and every property is a term in this taxonomy. We associated a description with every term which should allow you to asses whether the property is supported by the tool you want to evaluate. Feel free to contact us for an account on the website so that you can add your evaluated tool.

5 Areas for improvement

Based on our evaluations in Section 3, we identify six areas for improvement in the current generation of tools. We believe that tools who address these areas will have a significant competitive advantage over other tools. The areas are:

1. **Create better abstractions**: Very few tools support creating higher-level abstractions like those mentioned in Figure 2 on page 4. If they do, those capabilities are hidden deep in the tool's documentation and not used often. We believe this is a missed opportunity. Creating higher-level abstractions would enable reuse of configuration specifications and lower the TCO of a computer infrastructure. To realize this, the language needs to (a) support primitives that promote reuse of configuration specifications like parametrization and modularization primitives, (b) support constraints modeling and enforcement, (c) deal with conflicts in the configuration specification and (d) model and enforce relations.
2. **Adapt to the target audience's processes**: A tool that adapts to the processes for system administration that exist in an organization is much more intuitive to work with than a tool that imposes its own processes on a system administrators. A few examples of how tools could support the existing processes better:
   - *structured documentation and knowledge management*: Cfengine3 is the only tool in our study that supports structured documentation in the input specification and has a knowledge management system that uses this structured documentation. Yet, almost all system administrators document their configurations. Some do it in comments in the configuration specification, some do it in separate files or in a fully-fledged content management system. In all cases, documentation needs to be kept in sync with the specification. If you add structured documentation to the configuration specification, the tool can generate the documentation automatically.
   - *integrate with version control systems*: A lot of system administrator teams use a version control system to manage their input specification. It allows them to quickly rollback a configuration and to see who made what changes.

Yet, very few tools provide real integration with those version control systems. A tool could quickly set up a virtualized test infrastructure for a branch that I created in my configuration. I would be able to test my configuration changes before I merge them with the main branch in the version control system that gets deployed on my real infrastructure.

- *semantic access controls*: In a team of system administrators, every admin has his own expertise: some are expert in managing networking equipment, other know everything from the desktop environment the company supports, others from the web application platform, ... As a consequence, responsibilities are assigned based on expertise and this expertise does not always aligns with machine boundaries. The ability to specify and enforce these domains of responsibility will prevent that for example a system administrator responsible for the web application platform modifies the mail infrastructure setup.
- *flexible workflow support*: Web content management systems like Drupal have support for customized workflows: If a junior editor submits an article, it needs to be reviewed by two senior editors, all articles need to be reviewed by one of the senior editors, ... The same type of workflows exist in computer infrastructures: junior system administrators need the approval from a senior to roll out a change, all changes in the DMZ needs to be approved by one of the managers and a senior system administrator, ... Enforcing such workflows would lower the number of accidental errors that are introduced in the configuration and aligns the tool's operation with the existing processes in the organization.

3. **Support true integrated management**: We would like to see a tool that provides a uniform interface to manage all types of devices that are present in a computer infrastructure: desktops, laptops, servers, smartphones and network equipment. Why would this be useful? When you have one tool, with one language that can specify the configuration of all devices, every system administrator speaks the same language and thinks in the same primitives: whether they are responsible for the network equipment, the data center or your desktops. The tool can then also support the specification and enforcement of relationships that cross platform boundaries: the dependencies between your web server farm and your Cisco load balancer, dependencies between desktops and servers, dependencies between your firewall and your DMZ servers, ... The current generation of tools either focuses on a single platform (Windows or Unix), focuses on one type of devices (servers) or needs different products with different interfaces for your devices (one product for network equipment, one for servers and one for desktops).

4. **Become more declarative**: The commercial tools in our study all start from scripting functionality: the system administrator can create or reuse a set of scripts and the tool provides a script-management layer. Research and experience with many open-source tools has shown that declarative specifications are far more robust than the traditional paradigm of imperative scripting. Imperative scripts have to deal with all possible states to become robust which results in a lot of if-else statements and spaghetti-code.

5. **Take the CIO's agenda into account**: Most open-source tools in our study have their origin in academia. As a result, they lag behind on the features that are on the CIO's checklists when deciding on a system configuration tool: (a) easy to use (graphical) user interface, reporting, (b) auditing, compliance, reporting capabilities in nice graphs and (c) access control support.

6. **Know that a system is software + configuration + data**: No tool has support for the data that is on the managed machines. Take a web server as example: the web server is software, that needs configuration files and serves data. System configuration tools can manage the software and configuration but have no support for state transfer: if my tool moves the web server to another node, I need to move the data manually.

6 Conclusion

We believe that this paper and our website can help system administrators make a more informed, and as a consequence better, choice for a system configuration tool. Our framework is not a mechanical tool: you can not check off the things you need and it will give you the perfect tool for you. We see it more as one of the decision factors that will save you a lot of time in the process of researching different tools: it quickly gives you a high-level overview of the features of each tool and enables you to trim down the list of possibilities for your use case. We will keep the website at `http://distrinet.cs.kuleuven.be/software/sysconfigtools` up to date when new

versions of tools are released and are open for adding new tool evaluations to our website.

7 Acknowledgements

We would like to thank our shepherd, Eser Kandogan for his comments on the draft versions of the paper and Mark Burgess for sharing his insights in the constraints-part of our framework. We would also like thank our anonymous reviewers and all people who commented on the tool evaluations on the website.

This research is partially funded by the Agency for Innovation by Science and Technology in Flanders (IWT), by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven.

References

[1] ALVA COUCH, JOHN HART, E. G. I., AND KALLAS, D. Seeking closure in an open world: A behavioral agent approach to configuration management. In *Proceedings of the 17th Large Installations Systems Administration (LISA) conference* (Baltimore, MD, USA, 10/2003 2003), Usenix Association, Usenix Association, p. 125–148.

[2] ANDERSON, P., AND COUCH, A. What is this thing called "system configuration"? LISA Invited Talk, November 2004.

[3] ANDERSON, P., AND SMITH, E. Configuration tools: Working together. In *Proceedings of the Large Installations Systems Administration (LISA) Conference* (Berkeley, CA, December 2005), Usenix Association, pp. 31–38.

[4] BARRETT, R., KANDOGAN, E., MAGLIO, P. P., HABER, E. M., TAKAYAMA, L. A., AND PRABAKER, M. Field studies of computer system administrators: analysis of system management tools and practices. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work* (New York, NY, USA, 2004), ACM, ACM, pp. 388–395.

[5] BARRETT, R., MAGLIO, P. P., KANDOGAN, E., AND BAILEY, J. Usable autonomic computing systems: The system administrators' perspective. *Advanced Engineering Informatics 19*, 3 (2005), 213 – 221. Autonomic Computing.

[6] BARTAL, Y., MAYER, A., NISSIM, K., AND WOOL, A. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst. 22*, 4 (2004), 381–420.

[7] CHARALAMBIDES, M., FLEGKAS, P., PAVLOU, G., BANDARA, A. K., LUPU, E. C., RUSSO, A., DULAY, N., SLOMAN, M., AND RUBIO-LOYOLA, J. 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)* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 99–108.

[8] COLVILLE, R. J., AND SCOTT, D. Vendor Landscape: Server Provisioning and Configuration Management. Gartner Research, May 2008.

[9] FEYRER, H. g4u homepage. http://www.feyrer.de/g4u/.

[10] FU, Z. J., AND WU, S. F. Automatic generation of ipsec/vpn security policies in an intra-domain environment, 2001.

[11] GARBANI, J.-P., AND O'NEILL, P. The IT Management Software Megavendors. Forrester, August 2009.

[12] HABER, E. M., AND BAILEY, J. Design guidelines for system administration tools developed through ethnographic field studies. In *CHIMIT '07: Proceedings of the 2007 symposium on Computer human interaction for the management of information technology* (New York, NY, USA, 2007), ACM, ACM, p. 1.

[13] HREBEC, D. G., AND STIBER, M. A survey of system administrator mental models and situation awareness. In *SIGCPR '01: Proceedings of the 2001 ACM SIGCPR conference on Computer personnel research* (New York, NY, USA, 2001), ACM, ACM, pp. 166–172.

[14] LUPU, E., AND SLOMAN, M. Conflict analysis for management policies. In *Proceedings of the Vth International Symposium on Integrated Network Management IM'97* (May 1997), Chapman & Hall, pp. 1–14.

[15] MARTIN-FLATIN, J.-P., ZNATY, S., AND HUBAUX, J.-P. A survey of distributed enterprise network andsystems management paradigms. *J. Netw. Syst. Manage. 7*, 1 (1999), 9–26.

[16] MOFFETT, J. D. Requirements and policies. In *Proceedings of the Policy Workshop* (November 1999).

[17] NARAIN, S. Towards a foundation for building distributed systems via configuration. http://www.argreenhouse.com/papers/narain/Service-Grammar-Web-Version.pdf, 2004.

[18] OPPENHEIMER, D. The importance of understanding distributed system configuration. In *Proceedings of the 2003 Conference on Human Factors in Computer Systems workshop* (April 2003).

[19] OPPENHEIMER, D., GANAPATHI, A., AND PATTERSON, D. A. Why do internet services fail, and what can be done about it? In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems* (Berkeley, CA, USA, 2003), USENIX Association, USENIX Association, p. 1–1.

[20] OPPENHEIMER, D., AND PATTERSON, D. A. Studying and using failure data from large-scale internet services. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop* (New York, NY, USA, 2002), ACM, ACM, p. 255–258.

[21] Partimage homepage. http://www.partimage.org.

[22] PATTERSON, D. A. A simple way to estimate the cost of downtime. In *Proceedings of the 16th USENIX conference on System administration* (Berkeley, CA, USA, 11/2002 2002), USENIX Association, USENIX Association, p. 185–188.

[23] RAYMER, D., STRASSNER, J., LEHTIHET, E., AND VAN DER MEER, S. End-to-end model driven policy based network management. In *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop* (2006), p. 4.

[24] SYMANTEC. Norton Ghost Homepage. http://www.symantec.com/ghost.

[25] VERMA, D. Simplifying network administration using policy-based management. *IEEE Network 16*, 2 (Mar/Apr 2002), 20–26.