A framework for integrated configuration management tools

Bart Vanbrabant, Wouter Joosen
{bart.vanbrabant, wouter.joosen}@cs.kuleuven.be
iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium.

***Abstract*—IT infrastructure management is becoming increasingly complex due to the increasing size and complexity of IT infrastructures. This complexity leads to errors and subsequently to failure. Configuration errors caused by operator errors are a significant contributor to downtime. These errors are often introduced by failing to update all dependent configuration parameters. Interdependencies between parameters are often not explicitly documented and exist at all layers in an infrastructure. Existing configuration management tools fail to capture all relevant relations and as such are prone to introducing inconsistencies. In this paper we present the Infrastructure Management Platform (IMP), a framework for building integrated configuration management tools. IMP models all relevant interdependencies between parameters to support consistency when updating a configuration. IMP is an integrated management framework, managing relations between configuration parameters across all layers. Additionally it reuses existing management interfaces and deployment agents to facilitate integrated management. We have validated IMP in several case studies by implementing reusable configuration modules, and have demonstrated that tools built on top of IMP can significantly increase the abstraction level in which an infrastructure is configured.**

I. Introduction

IT infrastructures grow larger and more complex because of the growing importance of IT in our society. Operators need to manage these infrastructures to ensure it functions correctly with as little downtime as possible. This is a growing challenge because of the scale, heterogeneity and the rate of change in contemporary IT infrastructures; there is no margin for error.

The complexity of managing IT infrastructures is caused by relations and interdependencies between applications, services and subsystems in an IT infrastructure. The actual configuration of a real or virtualized infrastructure is determined by many parameters, in fact in the order of thousands. Only a fraction of these parameters can be chosen freely by the operators. All other parameters are either a duplicate of an other parameter or they are derived from other dependent parameters. An example stems from IP address allocation: only the last octet of an IP address can be freely chosen between 1 and 254 in a class C network, the other three octets are dependent on the network address. The set of managed parameters and their interdependencies result in a complex configuration parameter space which is in practice controlled by operators. Whenever a configuration parameter changes, operators need to ensure the updating of all dependent configuration parameters. Inconsistent parameters in the infrastructure lead to service failure. Research has revealed that configuration errors are a big contributor to service failure and mean time to recovery (e.g. [1], [2]).

Configuration management tools assist operators in automating management of their infrastructures. They are generic tools to configure and manage an infrastructure or application, in contrast to custom scripting. The research in this paper is inspired by the observation [3] that these tools do not operate at a high enough abstraction level to manage an entire infrastructure. Configuration management tools such as Puppet, Cfengine, Chef, ... manage an infrastructure in function of primitive operating system resources such as files, services, packages and users. The current state of the art in these tools provide mechanisms to encapsulate complexity and heterogeneity but are limited to a single host. To cope with these deficiencies operators often need to resort to custom scripting to manage an entire infrastructure with management tools [4].

Recently, cloud computing infrastructures such as IaaS have increased the above mentioned complexity, although IaaS typically adds additional layers of abstraction to provide homogeneous virtual machines on heterogeneous systems. One of the key advantages of IaaS is the pay-per-use concept. But this also means that an infrastructure frequently needs scaling up and scaling down. This requires full automation of the configuration and infrastructure management.

In this paper we present our Infrastructure Management Platform (IMP). IMP manages an infrastructure in an integrated fashion. It realizes this integration by providing the support to model all relevant relations between configuration parameters. These relations ensure that every configuration parameter is only specified once in the configuration. All duplicated or derived configuration parameters are modeled explicitly by specifying relations. IMP delivers the following contributions to achieve this ambitious goal:

1) A modeling language to capture all relevant relations between infrastructure parameters at all layers in the configuration and encapsulating them in reusable configuration modules. Configuration modules provide a domain model and implementations for domain concepts. The entire configuration model represents the desired state of each parameter in the infrastructure.
2) An architecture which is extensible with transformation and export plug-ins. Transformation plug-ins enable parameter manipulation in an imperative programming language, enabling complex configuration models while keeping the core modeling language concise. Export plug-ins provide a connection to management interfaces of applications and services and a connection to existing tools. It ensures that IMP can leverage on existing interfaces and tools to deploy configuration changes.

We have prototyped IMP and evaluated it using case studies to manage real infrastructures and applications that are hosted at an IaaS provider. Each of these cases are standalone tools that manage certain aspects of an infrastructure. These tools are modular and share modules with partial configuration models. The cases are: 1) A tool to manage the networking devices, firewalls and the network configuration of hosts in an enterprise. 2) A tool to manage an IPv4, IPv6 or dual stack network from the same configuration model and migrating them from IPv4 over dual stack to IPv6 only. 3) A tool to manage a component based multi-tier Java EE application on an IaaS platform and scale it autonomic using a transparent performance model. The evaluation shows that IMP encapsulates configuration knowledge in reusable configuration modules and allows integration of scripts to automate specific configuration tasks.

The remainder of the paper is structured as follows: first we elaborate on the problem statement and elicit requirements for a framework for integrated configuration management tools in section II. In section III we describe the architecture and design of the framework. Section IV discusses the actual implementation of the prototype. In section V we provide a short overview of three case studies that have been built on IMP. In section VI we discuss related work and we conclude in section VII.

II. Problem elaboration and requirements

Operators deal with the complexity of managing IT infrastructures by adding custom scripting to the tools they have to their disposal. Operators used to resort to custom scripts to automate certain aspects of the daily operations of their infrastructure. These scripts keep the interdependencies in the infrastructure up to date. For example, a script that adds a new host to the network, configures the DHCP and DNS servers, adds firewall rules and authorises the host to use printers. These scripts are brittle and therefore prone to interference because of the lack of modularity when changes are made to the infrastructure. Additionally they are in general not reusable for other infrastructures.

Configuration management tools provide a more generic tool than scripting. These tools provide a management interface to the operators. They range from setting parameters on resources deployed on hosts to more advanced tools that can build up abstractions on top of setting parameters. In ”A survey of system configuration tools” [3] we compared 11 existing commercial and open-source configuration management tools. We identified several shortcomings in these tools. A key shortcoming is the lack of support for integrated configuration management. Since we did the survey, tools such as CFengine and Puppet made effort to raise the abstraction level but still lack the capabilities to raise the abstraction level to the infrastructure level for true integrated configuration management. Additionally these tool cannot handle heterogeneity. For example, they only support UNIX servers, Windows machines or Cisco network devices. To reduce parameter duplication operators again resort to custom scripting to integrate all the different tools required to manage an entire infrastructure.

The complexity of managing a large IT infrastructure can only be reduced when all relations between configuration parameters are captured as relations in a model so parameter duplication is no longer necessary. Additionally this is also required to fully automate adding and removing servers from an infrastructure as required by cloud computing.

Research into humans and automation and more specifically into human factors of system administration has provided additional requirements for management tools [4]–[7]. This research makes a distinction between the mental model an operator has from the infrastructure and the situation awareness. A mental model is a persons cognitive representations of how a system operates, allowing him to anticipate future events and formulate plans. For operators this is the architecture, all the physical properties (devices, connections, . . .) and the configuration parameters. In contrast, situation awareness is more concrete and tactical. It provides the current state of the infrastructure. Both are required for successfully managing and troubleshooting an infrastructure.

The mental model is more long term and static. A management tool should assist an operator in building this mental model. It is important to have as much information about the *model* of the infrastructure in the same tool, at all levels of abstraction. Operators build up situation awareness of their infrastructure using inspection tools such as monitoring tools and dashboards. A management tool can assist in setting these inspection tools up and ensuring that the configuration of the inspection tools stays up to date with the current model of the infrastructure.

Human factors research in system administrations and operators themselves also argue for command line tools, ASCII configuration files and APIs to management interfaces [6], [8]. This is required to be able to reproducible configure a system and automate configuration tasks.

To conclude the requirements for a management tool that realises truly integrated management are:

- Allow arbitrary levels of abstraction without limiting to hosts, subsystems or datacenters.
- Relations between entities in the network need to be first class citizens.
- Support the system administrator in building a mental model of the infrastructure.
- Enable modularity and reuse of configuration knowledge.
- Allow interfacing with existing management interfaces and management tools for deploying configuration changes.
- Deal with custom scripting by enabling operators to include scripts in the framework in a structured manner with an easy to use API.
- Use text input and allow operators to customize the tool using scripting.

III. Architecture and design of IMP

IMP is conceived as a platform to build integrated configuration management tools. Tools built on IMP consist of modules that each encapsulate configuration knowledge about a service, subsystems or application. Each module provides a partial configuration model that consists of a domain model and implementations that define domain concepts in function of other domain concepts. Additionally a module contains export and transformation plug-ins and templates to extend IMP. A conceptual architecture of IMP is shown in Figure 1. Tools built on IMP consist of one or more modules required to manage the parts from an infrastructure or manage the infrastructure in its entity. In this section we elaborate on modeling the configuration of an infrastructure, transforming high level configuration models to a deployable configuration and how IMP interfaces with existing tools.

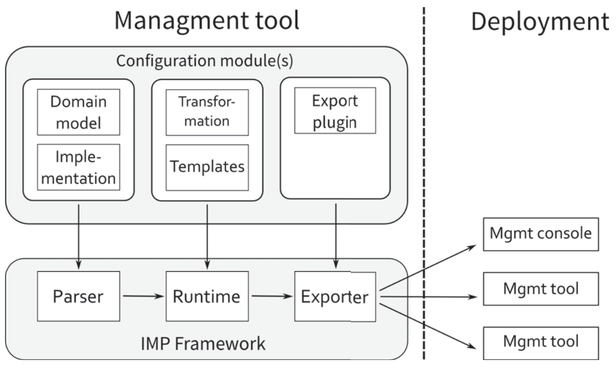

Figure 1. The architecture of IMP

A. Domain model and implementation

IMP contains a modeling language that decouples modeling the domain model from its implementation. Domain models in IMP define domain concepts such as a web or application server. Interdependencies between entities are modeled explicitly. They can be navigated bidirectionally and have an explicit multiplicity. These relations are required to model all configuration parameters only once. Each entity can have attributes that can contain strings, numbers and booleans. Similar to object oriented designs, entities can inherit from other entities.

The configuration model also implements domain concepts in function of other domain concepts providing abstraction layers. This mechanism encapsulates configuration knowledge, dependencies, relations, . . . behind a clear interface that is defined by the domain model. For example the domain concept WebServer is implemented in function of configuration files, services and software packages. By providing encapsulation, abstractions can be built up in the configuration model itself. This is important so an IMP configuration model can capture all relevant relations between configuration parameters at all levels of abstraction in one integrated model.

For each domain concept one or more implementations can be defined. For example, providing multiple implementations for a webserver using different server software. Or dealing with the issues of inconsistent service, package and configuration file naming on different platforms. Only one default implementations can exist next to multiple implementations that are selected based on a condition. When no condition matches the default implementation is used.

The separation between domain model and implementations allows for configuration modules to be developed that provide a standards based domain model without implementation. For example based on standards of the DMTF such as CIM [9]. Other modules can then provide implementations for these domain models.

During evaluation of the configuration model IMP verifies constraints to validate the consistency of the configuration model. First, the typing of the configuration model is verified. The attributes of domain concepts have types such as number, bool or string. Second, relations have a type and have a multiplicity that is enforced by IMP. Additionally IMP ensures that a double binding is maintained. Third, a configuration model developer can also define constraints on numbers and strings using expression or regular expressions.

Developers of configuration modules can define assertions over relations to validate the configuration model. For example, all network interfaces should have distinct mac-addresses. These assertions can be defined in the modeling language itself or in plug-ins. Plug-ins are scheduled after the configuration model is expanded and before the export plug-ins are executed.

B. Transformation

IMP builds up abstractions through encapsulation. During the evaluation of the configuration model the IMP runtime expands each domain concept using the available implementations. IMP starts at the highest abstraction level and expands the configuration model in layers until the configuration model is deployable using export plug-ins. The model is evaluated by the runtime that iterates over all domain concepts and expands them until all instances of domain concepts are expanded or an implementation is not available for a domain concept. This process is illustrated in Figure 2.

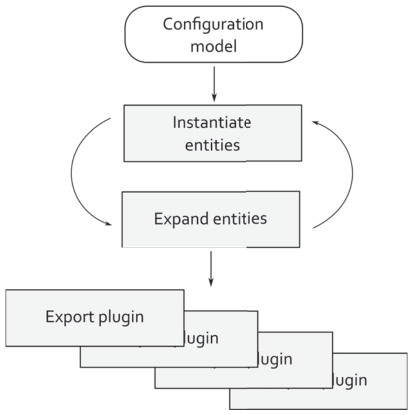

Figure 2. Process of expanding entities to the abstraction level of the interfaced configuration management tools.

C. Interfacing with existing tools

IMP interfaces with existing tools in two distinct ways. First IMP adds transformation plug-ins that define parameter transformations in an imperative programming language. Second, IMP supports export plug-ins that reuse other management interfaces. IMP leverages on existing management interfaces of platforms and the interfaces of existing configuration management tools to deploy actual configuration changes, it needs to interface with them. Each management interfaces functions at different abstraction levels. Therefore

IMP expands a configuration model to the required abstraction level.

Export plug-ins provides a native implementation for one or more domain concepts. These plug-ins interface with other tools and are distributed inside configuration modules. This means that the runtime will not expand instances of those concepts. Depending on the tool that the plug-in interfaces with, it will generate files, create database entries, etc. These plug-ins can also be used to generate documentation based on the configuration model, such as architecture or network diagrams.

Transformation plug-ins are executed during the expansion of the configuration model. A plug-in provides a declarative interface to imperative code. Each plug-in function accepts argument, has access to the current scope in the configuration model and returns a value it computed based on that scope. These functions can be simple string manipulation functions but could also interface with other tools such as a hardware inventory to lookup the MAC-address of a network interface.

Transformation functions can navigate throughout the entire configuration model that has been resolved and expanded at the time the function is scheduled for execution. The runtime only exposes a read-only version of the configuration model to the functions and secondly whenever the imperative code in a function tries to navigate the configuration model and accesses a parameter or relations that has not yet been initialized, the execution of the function is aborted and the runtime backtracks to the point where the execution of the function was started.

Transformation plug-ins are also typed. A plug-in function needs to specify the configuration model types for each parameter and the return value. Although a developer is not bound to the types in the function itself, its return value is validated to match the specified return type.

IV. Implementation

In this section we explain some of the challenges we faced during the implementation of a prototype. The implementation is developed in Python 3 using a custom DSL, plug-ins are also developed in Python.

In the implementation the export and transformation plug-ins are first class citizens. To facilitate the development of plug-ins the evaluation of the configuration model is done by building up the configuration models type system and instances in Python objects using meta-programming, creating a mirror of the configuration model in the Python runtime. Therefore the API for traversing the configuration model is in a Python function the same as in the configuration model itself. A new plug-in is defined in a Python module by adding a function decorator (which is similar to an annotation in Java) and using argument and function annotations which have been introduced in Python 3.

The runtime schedules the evaluation of statements in the DSL. In a first phase it defines the types so during the expansion of the model, no new types will appear which would require the entire model to be validated again. In the second phase IMP generates placeholder variables that have a symbolic reference to their dependencies. This is possible because all types, and therefore all parameters and relations are known. Only when the symbolic variable resolves to an instantiated variable the statement is evaluated. IMP iterates over all statements either until they are all evaluated or until during two subsequent loops no additional statements have been evaluated. When this occurs IMP will try to determine the cause: either an error in the configuration model or a loop in the dependencies.

Plug-ins cannot manipulate the configuration model directly because it would be to complex to schedule statements and validate the configuration model. Instead of direct access to the Python mirror of the configuration model, the Python mirrors are wrapped in a proxy so the plug-ins cannot manipulate attributes and get copies of values instead. This proxy also ensures that plug-ins are scheduled correctly. Whenever a plug-in accesses an attribute or a variable that has not been resolved, the proxy raises an exception and IMP backtracks. The access that caused the backtracking is encoded in a symbolic variable and added to the dependency list of the plug-in statement. This ensures that on the next invocation of the plug-in, at least the previous failed access will succeed.

A special plug-in is the template plug-in. This plug-in integrates the Jinja2 [10] template engine in IMP. It takes as an argument the template it needs to compile. All other configuration parameters are directly accessible because the template engine has access to the entire configuration model. The dependencies of the template on the configuration model are retrieved from the template engines compiler and added to IMP, similar to other plug-ins.

V. Evaluation

In this section we present three cases for which we implemented IMP modules to implement a tool to manage the infrastructure in each case. In this evaluation we show the reuse of modules, encapsulation of configuration knowledge and the extensibility of the architecture.

A. Case 1

The first tool manages network and security devices in an infrastructure. It hosts a 3-tier application with a presentation, application and data tier. Each tier is separated from the other tiers by firewalls. The tool manages network switches, routers and the networking configuration of the hosts in the network. Additionally it will generate firewall rules for all devices in the network based on the defined security policy.

In this case the IP configuration module provides a domain model to model all connections between services in the infrastructure. Other modules such as the jboss and mysql modules inherit from the client and server interfaces. During the model expansion templates generate firewall rules based on all client-server connections in the model. This creates a very strict policy that only allows connections that are modeled. A Python script validates each modeled connection against the security policy. This script is executed after the configuration model is expanded, but before export plug-ins are executed. If a connection violates the security policy a compilation error is issued.

B. Case 2

The second case introduces an improved network model over case 1 to manage the configuration of a dual stacked network. The tool allocates the network prefixes autonomously for IPv4, IPv6 or both. It also configures DHCP and DNS in the network. It defines the structure of the network in subnets (subnet 1 to 4), routers (router1), servers (server1) and clients (pc64). The entities in the structure of the network have a relation to an instance of the type `ip::Network`. Only on this instance network configuration parameters such as the available IP ranges for each version, the protocols for host configuration, the used routing protocols, ... are configured. These parameters and the relations between the instances that define the structure of the network are used to derive the networking configuration.

IMP includes a transformation plug-in to allocate IPv4 addresses dividing the IP range in the smallest subnet that fits the number of devices and the requests spare capacity for a subnet. This plug-in allocates IPv6 subnets based on the algorithm in ”Automated and secure IPv6 configuration in enterprise networks” [11]. The allocation of IP ranges to IP subnets is persisted and loaded on every invocation of IMP. This case also interfaces with Linux servers using Puppet and with Cisco routers using TFTP and the console.

C. Case 3

The third case is an integrated tool for managing component based applications and their execution environment on an IaaS platform and scaling these applications automatically. Figure 3 shows the entire configuration an operator is required to specify, to deploy and configure a component based Java EE application [12]. The configuration model the operator needs to develop only requires details about the IaaS to deploy the application on and the components the application consists of. Each component is allocated to a tier and its dependencies on other components. The tool generates virtual machines that are provisioned on an IaaS from a base image and configured to run as a node in a JBoss cluster per application tier. The infrastructure is monitored and a performance model is automatically build. This performance model is used to determine how many application servers per tier are required to stay within the performance boundaries defined on line 16 and 17 of Figure 3.

This case uses plug-ins to automate specific parts of the configuration. In this case this is the allocation of services to virtual machines, because the underlying IaaS does not allow control of virtual machine placement anyway. Transformation plug-ins are also used to configuration parameters such as the IP-addresses of the VM’s, based on the configuration data provided by the IaaS. In this case IMP interfaces with Puppet for VM configuration and with boxgrinder for VM provisioning and IaaS interfacing. Additionally IMP also communicates with the JBoss management interface to configure the application servers.

D. Conclusion

Each of the cases raise the abstraction level of the configuration of the infrastructure. Additionally all configuration parameters are specified only once avoiding configuration

```
1  # define the IaaS environment to deploy the
2  # application on
3  cloud = paas::IaaS(dns1 = "10.0.0.1", dns2 = "10.0.0.2",
4      domain = "local")
5
6  # define the application and the ear
7  app = paas::JEEApp(name = "epub", ear =
8      file("epub/epub-ear.ear"), paas = cloud)
9
10 # scale the application up when the responsetime
11 # goes over 100ms
12 app.scale = paas::Scale(reponsetime = 100,
13     spare = 0.1, hysteresis = 0.05, average = 600)
14
15 # create the tiers
16 web-tier = paas::WebTier(name = "web", paas = cloud,
17     jee_app = app)
18 business-tier = paas::AppTier(name = "business",
19     paas = cloud, jee_app = app)
20 backend-tier = paas::AppTier(name = "backend",
21     paas = cloud, jee_app = app)
22 data-tier = paas::DataTier(name = "database",
23     paas = cloud, jee_app = app)
24
25 # add the components and allocate them to the tiers
26 newspaper-war = paas::WebApp(name = "newspaper",
27     context = "newspaper", requires = [newspaper-ejb],
28     tier = web-tier)
29 newspaper-ejb = paas::EJB(name = "newspaper",
30     tier = business-tier, requires = [cms-ejb])
31 cms-ejb = paas::EJB(name = "cms", tier = backend-tier)
32
33 # add a datasource and add it to the components in
34 # the backend tier
35 paas::DataSource(name = "diginews", datatier = data-tier,
36     apptiers = backend-tier)
```

Figure 3. Instantiation of a component based application and deploy it on an IaaS platform in case 3.

errors due to inconsistent configuration parameters. Figure 4 depicts the number of instances that are created in each case and the amount of attributes and relations that are set. This provides an indication of the size of each case.

Each of the cases use scripts for specific transformations and to interface with existing tools. This is shown in Figure 4 by the lines of model code and the lines of code for the plug-ins included in the used modules. This table also show the amount of case specific configuration model code. This is a first indication of the level of reuse realised by IMP.

Metric	Case 1	Case 2	Case 3
IMP model code lines	1736	490	1313
IMP plug-in code lines	1408	1489	1456
Case specific lines	687	191	74 / 74
Managed hosts	22	70	13 / 34
Domain concept instances	960	413	1225 / 3301
Relations between instances	2322	846	2888 / 7868
Instance attributes set	2816	1239	4550 / 12470
Compilation time	10.1s	4.9s	14.2s / 1m9.2s

Figure 4. Metrics from each of the cases. Case 3 provides the metrics for the lower and the upper bound of hosts during scaling of the experiment.

In Figure 5 a table is depicted that lists all configuration modules we developed on IMP and in what cases we used each module. We only added new configuration parameters to modules but kept it compatible with the other cases by providing default values for new parameters. In the table the modules with a ◇ indicate that we developed a new version of the module that is not compatible with the other cases.

For the IP module this is because we switched from an IPv4 module to a module that support dual stacked networks. The jboss module in case 1 only supports fixed web and application tiers. The new version supports an arbitrary amount of tiers. Figure 5 gives an indication of the configuration reuse possible with IMP.

Module	Case 1	Case 2	Case 3
cisco	√	√	-
dhcp	√	√	-
dns	√	√	-
fw	√	-	-
hbase	-	-	√
httpd	√	-	√
ip	√	◇	√
jboss	√	-	◇
mcollective	-	-	√
mysql	√	-	√
mysql-proxy	-	-	√
nagios	√	-	√
net	√	√	-
nginx	-	-	√
opentsdb	-	-	√
paas	-	-	√
redhat	√	√	√
shorewall	√	-	-
tcollector	-	-	√
vm	-	-	√
std	√	√	√

Figure 5. Reuse of IMP modules in configuration tools.

VI. RELATED WORK AND FUTURE DIRECTIONS

The related work for IMP is situated in the space of commercial and opensource tools and in research. In ”A survey of system configuration tools” [3] we compared 11 tools that represent the state of the art. The conclusions of this survey relevant to this work are: creating better abstractions, support true integrated management and better modeling of the infrastructure. These conclusions are still valid for the current versions of these tools.

IMP can reuse existing domain models and interface with the agents that implement management interfaces. Incentives such as CIM [9] from the DMTF provide a domain model for configuration management tools. Additionally these standards also define management interfaces such as WS-MAN [13].

Narian et al. [14] present a tool based on constraint satisfaction and model finding. This tool provides a higher level input language for a configuration model and combines model finding with other methods to speed up resolving the model. The authors validated the tool in specific configuration tasks such as IP address assignment. The tool does not have facilities to model abstractions or extending the tool with scripting. A future addition to IMP could use model finding for the allocation of a specific set of parameters in a configuration model. For example, replace the IP allocation algorithm in case 2 with model finding.

Pattern-based Composite Application Deployment [15] describes the deployment of configuration changes based on a configuration model. IMP requires declarative management interfaces or tools for deployment. The presented approach can use existing imperative scripts or actions for generating a deployment workflow. The same authors propose a model driven approach [16] for configuration management tools. Their prototype employs a GUI to create the models and either the GUI or a Java API for the transformations. Their main requirement is separation of concerns for the different stakeholders in deploying and managing applications in a datacenter. This differs from our work in the following aspects: Our approach is more related to object oriented development and agile methodologies, while their approach is in the tradition of model driven development. This also manifests in a second difference. The architecture of IMP enables scripting within the framework. These scripts can be used to automate specific parameter transformations. The work of Eilam et al. needs domain specific search heuristics for automatic model transformations which are non-trivial to develop. Additionally IMP uses text based input which operators prefer over GUI based tools to create the models.

Rodosek proposes a methodology [17] for modeling services instead of devices. He also proposes to use an object oriented approach combined with UML. His work is complementary to our work as he proposes a methodology that is compatible with IMP.

Agrawal et al. [18] describes the design of a CIM compatible policy language. This language is also declarative in the sense that it is event-condition based. Our approach is a state based approach, where we model the desired state of the infrastructure. The deployment agents determine how to progress the current state of the infrastructure to the desired state. For this we use existing agents which only consider a single subsystem or device. In future work we intend to improve this approach by coordinating the changes in the entire infrastructure so they occur in the correct order. For example, ensuring that redundant load-balancers are restarted in sequence.

VII. CONCLUSION

In this paper we have introduced IMP, a framework to develop integrated configuration management tools. An IMP-based tool is built up from reusable configuration modules that provide a partial domain model and configuration implementations for the domain concepts. IMP focuses on modeling the interdependencies between parameters in a configuration model by specifying relations. Because all relevant relations between configuration parameters are made explicit, a change to one parameter can automatically trigger a process that updates all dependent parameters.

In the evaluation section, we have illustrated that IMP can manage real world infrastructures in an integrated manner. IMP also enables encapsulating configuration knowledge in reusable configuration modules. These modules contain a model of the configuration of the infrastructure and configuration specific plug-in scripts. We have also demonstrated that IMP interfaces elegantly with existing tools and interfaces.

ACKNOWLEDGMENT

This research is partially funded by the Agency for Innovation by Science and Technology in Flanders (IWT), by the Research Fund KU Leuven, and by the EU FP7 project NESSoS.

References

[1] T. Dumitraş and P. Narasimhan, "Why do upgrades fail and what can we do about it?: toward dependable, online upgrades in enterprise system," in *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Springer-Verlag New York, Inc. New York, NY, USA: Springer-Verlag New York, Inc., 2009, p. 1–20.

[2] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "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*, USENIX Association. Berkeley, CA, USA: USENIX Association, 2003, p. 1–1.

[3] T. Delaet, W. Joosen, and B. Vanbrabant, "A survey of system configuration tools," in *Proceedings of the 24th Large Installations Systems Administration (LISA) conference*. San Jose, CA, USA: Usenix Association, 11/2010 2010.

[4] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker, "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*, ACM. New York, NY, USA: ACM, 2004, pp. 388–395.

[5] D. G. Hrebec and M. Stiber, "A survey of system administrator mental models and situation awareness," in *SIGCPR '01: Proceedings of the 2001 ACM SIGCPR conference on Computer personnel research*, ACM. New York, NY, USA: ACM, 2001, pp. 166–172.

[6] R. Barrett, P. P. Maglio, E. Kandogan, and J. Bailey, "Usable autonomic computing systems: The system administrators' perspective," *Advanced Engineering Informatics*, vol. 19, no. 3, pp. 213 – 221, 2005, autonomic Computing.

[7] T. B. Sheridan, *Humans and Automation: System Design and Research Issues*, ser. HFES Issues in Human Factors and Ergonomics Series. New York, NY, USA: John Wiley & Sons, Inc., 2002, vol. 3.

[8] T. A. Limoncelli, "A Plea to Software Vendors from Sysadmins - 10 Do's and Don'ts," *ACM Queue*, p. 30, 2010.

[9] DMTF, "Common Information Model (CIM) Standards," http://www.dmtf.org/standards/cim/, 08 2012.

[10] "Jinja2 template engine," http://jinja.pocoo.org/docs/, 2012.

[11] F. Beck, O. Festor, I. Chrisment, and R. Droms, "Automated and secure IPv6 configuration in enterprise networks," *2010 International Conference on Network and Service Management*, pp. 64–71, Oct. 2010.

[12] D. Van Landuyt, S. Op de beeck, E. Truyen, and P. Verbaeten, "Building a digital publishing platform using aosd," in *LNCS Transactions on Aspect-Oriented Software Development*, vol. 9, December 2010, pp. 1–34.

[13] DMTF, "Web services management," http://dmtf.org/standards/wsman, 08 2012.

[14] S. Narain, G. Levin, S. Malik, and V. Kaul, "Declarative Infrastructure Configuration Synthesis and Debugging," *Journal of Network and Systems Management*, vol. 16, no. 3, pp. 235–258, 2008.

[15] T. Eilam, M. Elder, A. V. Konstantinou, and E. Snible, "Pattern-based composite application deployment," *12th IFIPIEEE International Symposium on Integrated Network Management IM 2011 and Workshops*, pp. 217–224, 2011.

[16] T. Eilam, M. Kalantar, A. Konstantinou, G. Pacifici, J. Pershing, and A. Agrawal, "Managing the configuration complexity of distributed applications in Internet data centers," *IEEE Communications Magazine*, vol. 44, no. 3, pp. 166–177, Mar. 2006.

[17] G. D. Rodosek, "A generic model for IT services and service management," *IFIP/IEEE Eighth International Symposium on Integrated Network Management, 2003.*, pp. 171–184, 2003.

[18] D. Agrawal, S. Calo, K.-w. Lee, and J. Lobo, "Issues in designing a policy language for distributed management of it infrastructures," *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pp. 30–39, 2007.