REPORT NO. xxxx/xxxx

An Intepretive Case Study in Configuration Management Systems

Ulf Eliasson
Karl Johansson

Department of Applied Information Technology
IT UNIVERSITY OF GÖTEBORG
GÖTEBORG UNIVERSITY AND CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2008

An Intepretive Case Study in Configuration Management Systems

Ulf Eliasson Karl Johansson

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

Abstract

Several studies in the recent decade has shown that operator mistakes are the most common cause of Internet service failures. Among these, misconfiguration is the most common one. Several configuration management systems exist to alleviate this problem. However, there is little or no amount of systematic analysis of research on said systems. Using data from five case studies of different configuration management systems, we analyzed core differences and similarities in approaches to configuration management. We find that Configuration Management Systems address either the problem of automating installation and configuration, or that of analyzing and verifying configuration correctness. Qualitatively we find that there is a relationship between these two approaches and the maturity level of software and services.

KEYWORDS: `Configuration, managability`

Contents

Chapter 1

Introduction

The notion of controlling the evolvement of software configuration items have been subject for studies since the 1980's. The need to keep revision histories, have the ability to audit the history of a configuration item, and the organization of software components into versioned libraries are bundled under the term *Software Configuration Management* [Dart, 1991]. At the same time as systems grow larger more complex, so does the amount of adaptational data. Adaptational data is generally the system parameters and associated values that determine the behavior of the system. These sets of adaptational data, *System Configurations*, needs to be managed in the same manner Software Configuration Items are [Dolstra and Hemel, 2007]. Furthermore, the European Union has recognized the growing complexity of systems, specifically Service Oriented Architecture (SOA) as an area in need of research, including automated setup and management procedures for said systems [Fitzgerald and Olsson, 2006]. Oppenheimer et al. point out in their study from 2003 that improvement in the maintenance tools and systems used by service operations staff would decrease the time to diagnose and repair problems. The results published in their article show that failure in two out of three Internet services are caused by configuration errors [Oppenheimer et al., 2003]. These articles show that there is a need for further studies in this area.

At the start of the study we were contacted by Erlang Training and Consulting Ltd. (ETC), who asked us to participate in a project where the customer had the need for a system that performed analysis and verification of system configuration data. Even if several commercial products handled the analysis and verification of the configuration data, no one product contained all of the desired features. During the initial literature review, we selected four similar case studies. The LCFG system, as described by Paul Anderson [Anderson and Scobie, 2000] is a system that automates the installation and configuration of Unix machines. The principles of the system is centered around a high level configuration and policy language, in which the system administrator expresses configuration data. Strider, described in [Wang et al., 2003], is a tool for identifying problems in the windows registry, using state-based analysis to identify probable causes of failure. The NixOS take on configuration management presented in[Dolstra and Hemel, 2007] is an operating system built around a Linux kernel. The system implements a functional method for handling installation and configuration of software packages. Dolstra's and Hemel's study is strongly centered around the idea that

the management of program binaries and adaptational data is tightly coupled. Finally we have reviewed the cfengine system presented in [Burgess, 2005]. This is a policy-based agent framework that uses a stochastic view of system configuration.

However, there was a lack of systematic analysis of shared and different aspects in these systems. Thus, from a practical perspective, the purpose of this study is to supply ETC and their customer with the system needed. From a research perspective, however, we will provide a systematic review of the core differences and similarities between five case studies – one of which we developed ourselves – and analyze the problems they are aimed at solving, as well as problems identified in related studies that are not addressed by the systems under study. We have designed our study as an interpretive case study, as presented by Walsham in [Walsham, 1995].

Chapter 2

Related Research

The amount of related research that analyse and compare configuration management systems is not overwhelming, and most studies and systems seem to be centered around systems that automate the installation and configuration of UNIX based systems. The following paragraphs present our review of four case studies of different systems that addresses the System Configuration process in one or more ways.

2.1 Nix

Dolstra et al. describes the Nix configuration management system, which is developed at the Department of Information and Computing Sciences at Utrecht University, in [Dolstra and Hemel, 2007], and argue that other configuration management systems have an *imperative* approach to system configuration in that they depend on and modify the state of the system. Nix, on the other hand, uses a *functional* approach to configuration management. In the paper, the authors argue for having configuration data generated from pure functions. These functions handle the entire process of installing and configuring software packages, which makes it central part of the operating system. This functional approach makes configuration data deterministic and reproducible. In order to provide traceability, all configurations are saved, and are never overwritten by reconfigurations or re-installations. This approach also enables system roll backs.

Nix makes use of its own language for configuring parts of the system. A configuration statement is called a *Nix expression* and provides all information Nix needs to build and configure a software package. The generated installation is immutable, it will not be altered in any way. Changes to the original configuration statement will issue a new build of the package, that will exist in parallel to the first installation, although the system will make use of the new installation. The advantage gained from this approach is nondestructive reconfigurations – safe rollback and tracability are at all stages guaranteed for the system.

The Nix system requires that all software packages within the system makes use of the

Nix configuration management. It therefore needs to control the operating system it exists on. NixOS is a Linux based operating system developed by the creators of Nix, which is based on Nix [NixOs, 2008]. The NixOS distribution features a number of software packages adapted to the Nix configuration management system.

2.2 LCFG

The LCFG system, presented in [Anderson and Scobie, 2000], is designed to manage installation and configuration of large sets of heterogeneous UNIX based systems. It was originally developed at The Department of Computer Science at Edinburgh University for managing their network of UNIX machines. Later expansions in the use of LCFG inside and outside of Edinburgh University forced a major update to the original implementation and this version is commonly referred to as LCFGng.

The core in LCFG is its own configuration language which is used for the specifications of the different systems that should be managed. This language is an abstraction of the configuration data. The decision to use an abstract language, instead of storing the configuration files themselves in the system, was made to support concepts such as relationships between data in various files. It also helps making the system more platform independent. The language describes what the configuration of a system should look like, not the steps that need to be taken to make it so.

To facilitate the handling of many systems that might share similarities and only have few differences, LCFG supports inheritance between configuration objects. Common attributes can be described in shared files and then have files that only specify the differences. These specifications are then stored in a central database. Installation of new nodes and reconfiguration of old ones is done automatically from this database without any required manual work. This is done to support large networks of nodes which would otherwise be impractical and time consuming to manage. Reconfiguration is triggered when the configuration of a node changes due to specification updates.

Because all application and subsystems that are managed by LCFG need to be incorporated in the configuration system and language, additions have to be simple and straight forward. This is solved by having an architecture that supports modules. To include support for an application or subsystem, a module is written independently of the other existing modules.

2.3 Strider

In [Wang et al., 2003], Wang et al. presents the Strider system. Originally designed at Microsoft's research labs, Strider addresses analysis of the Windows registry, the state of a Windows based system, and uses this to identify the cause of known errors. The authors argue that Strider uses a black-box approach – rather than relying on specifications of each parameter, it makes use of behavioral information to distinguish between configurational and operational data.

The theory presented in the article is based on three principles that address verification of configurations, and identification of problem sources. *State-based analysis* is the first step of the verification process. It is the primary approach to identification of the sources of failure using mechanical and statistical methods to narrow down the state to a set of likely candidates. *Attack the mess with the mass*, which compares the state that causes the failure to one that does not, thus narrowing down the candidate set further. "The mass" is a reference to a large installation base – the existence of several states that do not cause the failure in question. The last step, *Complexity-noise filtering*, uses statistics to determine the frequency for which each parameter in the state is updated. Parameters with a high update frequency are considered operational data and not part of the system configuration, and thus filtered out, narrowing the candidate set down even further.

2.4 Cfengine

Finally, the cfengine system, presented in [Burgess, 2005], is the product of an on-going research project about distributed system configuration management at Oslo University College. The policies are written in its own operating system independent language and are a description of the intended configuration for an host [Moffett and Sloman, 1993]. They contain partially ordered list of tasks or operations for the agents running on the systems to check but not the step on how to make the configuration policy compliant. It is the most widely used system that is covered in this study. Its policies are similar to those of the LCFG.

Each managed system has its own agent running. It is the responsibility of that agent to converge the system towards the intended configuration as specified in the database. When the agent discovers that the system it is responsible for deviates from the central policy it starts taking action in bringing the system closer to an ideal healthy state. A main consideration behind this design is the view of a computer system as a stochastic environment. Instead of assuming that a configuration file on a target system only changes when the configuration management system executes an action, it is believed that a system might change unpredictably at any time.

Chapter 3

Research Approach

Our research design is based on the interpretive case study, presented by Walsham in [Walsham, 1995]. We have adapted this method with the modification that rather than using interviews as our main source of evidence, we have studied five separate case studies of configuration management systems, one of which we have implemented ourselves. Evidence gathered from these studies was used to determine the properties of each system, examining how they address the problems they are designed to solve. For this purpose, the case studies were sufficient to replace interviews.

An initial literature review was conducted, where we identified key concepts in system configuration management. The theory gathered in this process was used as an initial guide to design and data collection, as described by Walsham in [Walsham, 1993], and formed our initial theoretical framework. This framework was used to select four suitable case studies of configuration management systems, that are part of our field study. These four studies were complemented with our own case study, which was conducted in parallel to a development project of a configuration management system. Our roles as participant observers in this development project enhanced our study, as it allowed us to be involved in the requirement specification and design of the system under study. The theory gathered in the initial literature review was used to establish several aspects of configuration management, and the systems under study were then compared from each of these aspects. Using theory from other studies to choose these aspects also served in avoid choosing problems that are addressed by the system we developed ourselves, thus countering our bias towards our own system. The evidence used in our study is gathered from the reports and articles reviewed during the data collection phase, complemented with our experiences as participating observers in the development of a configuration management system.

The articles that were selected for further studies were selected in order to cover the following criteria: *1.* Represent systems that are covered in substantial research material. This criterion was chosen to make sure that there was sufficient documentation to provide the evidence needed. *2.* Provide features that are considered important in recent research on system configuration management. This criterion was chosen to make sure our research is a relevant contribution to research on system configuration management.

Chapter 4

DataBuild Manager

The fifth study included in this analysis is the Data Build Manager (DBM). It is a system we developed for Erlang Training & Consulting Ltd (ETC).

4.1 System Overview

The purpose of DBM is to verify the correctness of system configurations. A central database holds representations of revisions of configuration files for the target systems. When a file is imported into the system, it is transformed to an internal tree structure – the *configuration tree* (the reverse operation, generating configuration files from the stored data, is outside the scope of the system). Each value in the imported file is associated with a parameter, identified by its path in the configuration tree, which may be defined by system administrators, providers of the target system, or any other domain expert.

Parameters Each value in a configuration is associated with a formal definition of its parameter. A parameter is defined by the following fields:

- Description – Human readable description of the parameter
- Unit – Unit of the parameter value
- Default – Default value for the parameter
- Scope – Scope of the parameter – is the value common for all systems of this type, or *local* – independent thereof
- Comments – User comments of the parameter

The system allows the user to compare the values of configuration revisions on the same system or between systems. This allows the user to find differences between known

correct configurations with the current ones retrieved from the target systems. It also aids an operator in spotting the changes needed to update the system in order for it to meet a required new configuration. It can also be used for tracking down changes that introduced errors or instabilities to the system.

The comparisons can be done on different scope levels, either comparing global parameters which are shared b between all systems of a certain type. A differences between systems here is likely to be a configuration error. The second option is to compare only local parameters, parameters that are likely to be different between the target systems, and where differences on different systems are not likely to be errors The third option is to make comparison of all parameters, regardless of scope.

4.2 Anticipated work flow

The expected day-to-day work with the DBM is

1. A user uses DBM to express desired changes to the configuration of a target system, and commits the changes to the database.
2. If needed, the user compares the parameter values that are common for all systems of the same type as the target system to verify that the requested configuration changes do not result in any discrepancies.
3. The user takes steps to ensure the desired changes are carried out on the target system. An administrator who has been provided with a copy of the configuration changes will carry out the task.
4. After the changes to the actual configuration are carried out, the user imports a configuration statement retrieved from the target system into DBM, and verifies that there is no discrepancies between the desired configuration and the actual one.

4.3 Architectural consideration

Bass et al. introduces the concept *Quality Attribute* in [Bass et al., 2003]. A quality attribute is a property of a system architecture, and is generally a requirement from a stakeholder. In their book, Bass et al. lists *tactics* for architectural design, that are aimed at achieving quality attributes. A tactic, in this setting, is a description of how a quality can be achieved.

An important attribute of the system was extendability. ETC wanted to be able to extend it to handle different kinds of systems in the future, without rewriting any non-system-specific parts. This was addressed by adopting the Modifiability tactic *Anticipate the change* in order to localize modification, described in [Bass et al., 2003]. The general idea in this tactic is to anticipate where changes may occur within the system, and set as goal to reduce the number of modules that are directly affected by change.

The resilience of the system – its ability to recover after failure – was also a requirement. This was addressed primarily by applying the *Checkpoint/rollback* tactic described by Bass et al. By letting the user make a checkpoint before critical operations it is always possible to rollback and load the checkpoint if something breaks. This can also be automated. Security of the data stored in the system was achieved by applying several Security Tactics. The integrity of the users was addressed by using the *Authenticate Users* and *Authorize Users* tactics. Each user of the system has to login before using the system, and each user has a level of privilege that determines user rights. For example a user with only read rights can read from the system while a user with read/write can both read and make changes to revisions. Finally, the integrity of the configuration data was addressed by applying the *Maintain Data Integrity* tactic, making sure that data belonging to a user can not be modified by any other user (that does not have administrative privileges).

Chapter 5

Discussion and analysis

5.1 Target system modification, platform independence and software installation

The ability to manage configuration data for all systems in an organization or network is important to the quality assurance of the system configuration process [Fitzgerald and Olsson, 2006, Oppenheimer et al., 2003]. As a result, the tool used in this process needs to either support the target system, or its extension to incorporate this support as a simple and time efficient task. Furthermore, the support for different types of target systems should be able to coexist within the same instance of the tool. From our analysis of the studies, we found that:

1. As shown in Figure 1, the Nix system requires a high degree of modification to the target system. In fact it requires that the operating system itself and all applications are installed and configured using Nix. The Strider system is designed purely for Windows, and the Windows registry is its only accepted input, with no modifications to the target system needed. DBM manages any configuration for which a parser for the target configuration file format is implemented, and requires no modification to the target system. DBM however lacks the ability to change an erroneous configuration. The two policy-based systems, LCFG and cfengine, both require agents or daemons to be installed on the target system.
2. Both Nix and Strider are OS specific tools, designed specifically for the task of supporting configuration management for their respective OS, making them insufficient in a heterogenous environment. Both LCFG and cfengine are somewhat platform independent, although both primarily focus on UNIX-like systems.
3. According to [Fisk, 1996], the process of installing and managing software packages is an important part of system configuration management. The same article further expresses the need for configuration management tools to facilitate installation and configuration of new machines. The deployment of a working service usually involves the installation and configuration of several components that the service depends upon [Dolstra et al., 2005]. This is often a time consuming and error prone

process, as dependencies might exist not only for specific software, but for specific versions of that software. Two of the systems under study support installation of software packages on remote machines: Cfengine and LCFG, one system supports software installation on the local machine: Nix, and two does not support software installation at all: DBM and Strider. Our findings from analyzing this ability in the systems under study, is that there is a relationship between platform dependence and the ability handle software installations, as illustrated in Figure 2.

Figure 1, in conjunction with Figure 2, shows that the systems under study provide either platform independence, or the ability to install new software on target systems. Furthermore, these two figures show that the ability to install software is tightly coupled with the need to make modifications to the target system. Thus, the systems that support software installation require different degree of modifications to the target system, which in turn conflicts with platform independency. It is possible that future research will provide solutions to this, for instance by emulating user behavior during software installation, instead of installing through agents, daemons or packet managers running on the target system.

5.2 Autonomy and verification

Oppenheimer et al report in [Oppenheimer et al., 2003] that at two out of the three Internet service services analyzed in their study, operator errors were the most common cause of failure, resulting in down-time. Out of these operator errors, configuration errors were the majority. Their results are supported by Zheng et al in [Zheng et al., 2007], as illustrated in Figure 3. A substantial amount of misconfigurations could be avoided with an automated configuration process – according to Zheng et al, 58% misconfigurations in their study were avoided when configuration was automated [Zheng et al., 2007]. When comparing the studies of the systems, we found that:

1. Both cfengine and LCFG are autonomous systems – any change to the central database will be automatically carried out to all affected target systems. DBM, Nix and Strider, on the other hand, all work only when invoked by a user.

2. In their study [Kycyman and Wang, 2004], Kycyman and Wang argue that prophylactic monitoring of configuration data is an important part of an self-managing configuration system. LCFG and Nix provide no analysis of the contents of actual generated configuration files whatsoever. Cfengine provides functionality to check file consistency using checksums, but does not analyze the configuration data further. Both DBM and Strider provide analysis of target system configuration data, and supports comparison to reference versions to detect inconsistencies. However, as Figure 4 illustrates, none of the autonomous systems in our study provide any substantial verification of configuration data.

Thus, systems with a high degree of autonomy provide little or no verification or analysis of configuration data on target systems. However, none of the studies present any evidence of conflicts between these two properties. The reason for this is the problems each system is aimed at solving. Cfengine, LCFG and NixOS all address installation

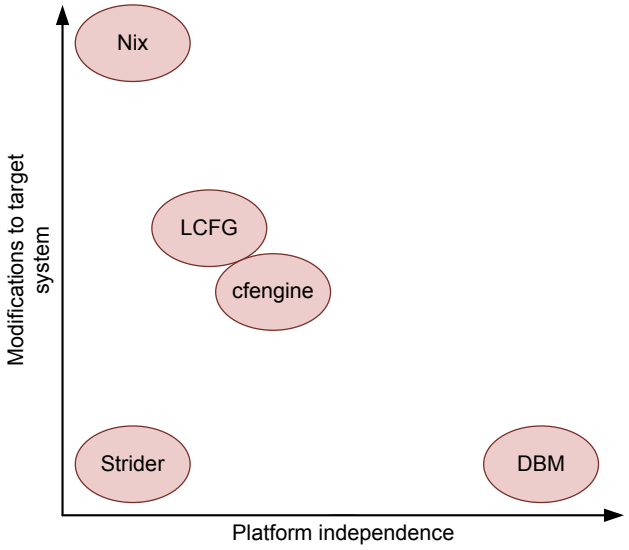

Figure 1: Platform independence, and modifications to target systems

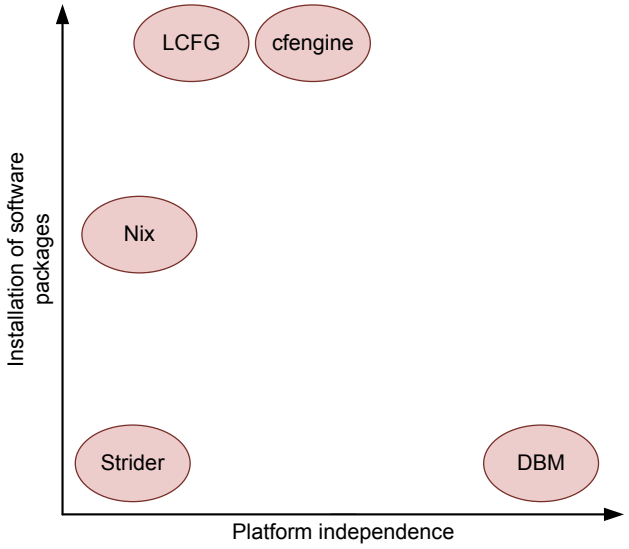

Figure 2: Platform independence, and ability to install software packages on target systems

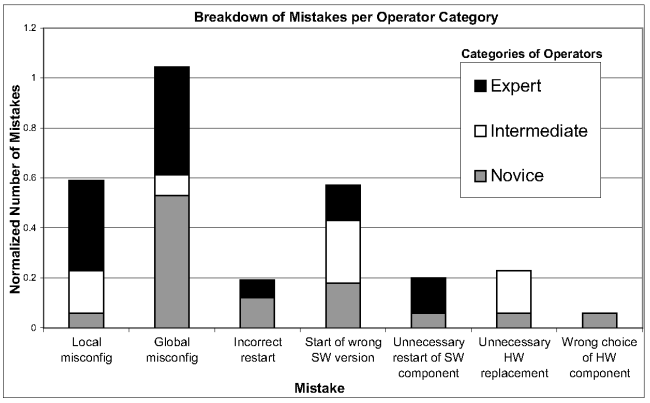

Figure 3: Operator mistakes per operator category (adapted from [Zheng et al., 2007]).

primarily, and do not identify verification as an aspect of configuration management. DBM and Strider, on the other hand, are both aimed at addressing only verification.

5.3 Discussion of future implication

As systems move towards service oriented architectures, they are often distributed across vast heterogeneous networks. This makes the process of configuration increasingly complex [Fitzgerald and Olsson, 2006]. Figure 5 illustrates the maturation of services, and in the report, the authors argue that the Second Generation Services represent current state-of-the-art in services. The vertical axis in this figure, Service Composition, is dependent on the service level of autonomy. Each generation of services in the figure is separated by a threshold that stipulates levels of:

- Quality & reliability
- Service management
- Interoperability
- Security and trust

In order to achieve the next generation, services must meet these levels. From a configuration management point of view, Service management requires autonomy – the configuration management system must be able to automatically resolve issues. On the other hand, the Quality & reliability aspect requires the configuration management system to provide analysis and verification of configuration, in order to identify problems before they become visible to the user [Oppenheimer et al., 2003, Wang et al., 2003].

As we have shown, there is no system that currently provides both the Service management and quality & reliability aspects while being autonomous. It is possible that future research initiatives in configuration management should focus on combining existing research in configuration verification, like [Wang et al., 2003, Kycyman and Wang, 2004]

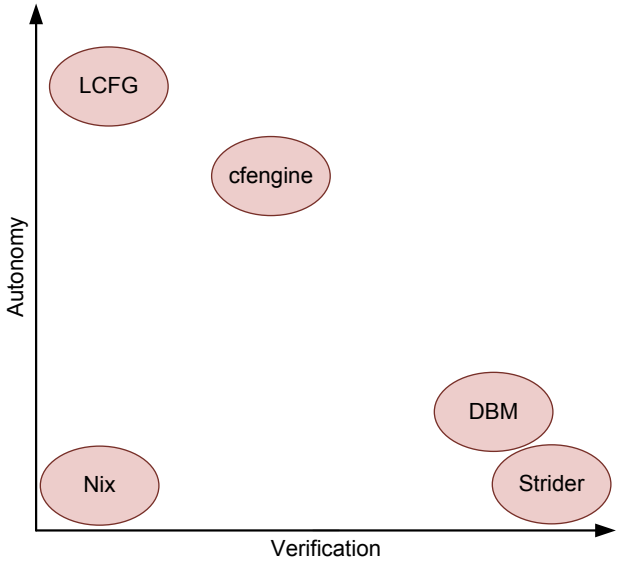

Figure 4: Illustration of autonomy and verification per system

with the more matured research on autonomous configuration of UNIX-like system [Dolstra and Hemel, 2007, Burgess, 2005, Anderson and Scobie, 2000]. Furthermore, in increasingly heterogenous networks [Fitzgerald and Olsson, 2006], platform independence will likely become an sought-for property of configuration systems, requiring further research efforts on configuration management systems.

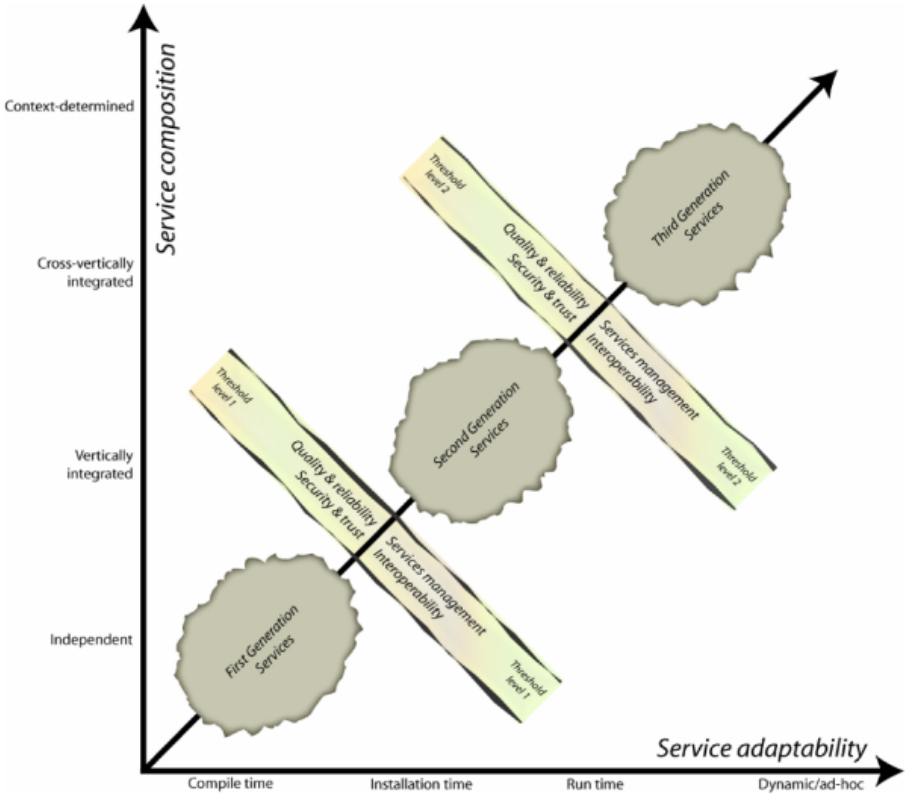

Figure 5: Relationship between autonomy, adaptability, and service maturity (Adapted from [Fitzgerald and Olsson, 2006]).

Chapter 6

Conclusions

In this paper we presented the results of our comparison of five case studies of different configuration management systems. In these five systems, we identified two major categories – either they are aimed at automating the installation and configuration process, or at analyzing and verifying existing configuration data in order to detect discrepancies, and to prevent failure. However, other studies have shown that there is a need for both these properties in configuration management.

Based on our findings, we propose that future studies in the field of configuration management focus on integrating these two approaches, aiming at providing both quality and reliability, as well as service manageability.

Chapter 7

Acknowledgments

We would like to thank Erlang Training and Consulting Ltd. for the opportunity to participate in the development of the DataBuild Manager, and for help with proof reading, assistance in testing, and developer support. Thanks to Jan Henry Nyström for proofreading in great detail and for useful suggestions and comments. We would also like to help Thomas Arts for helpful points and input. Last but not least we would like to thank Carl Magnus Olsson, our supervisor and guiding light in the world of research.

Bibliography

[Anderson and Scobie, 2000] Anderson, P. and Scobie, A. (2000). Large scale linux configuration with LCFG. In *Proceedings of the Atlanta Linux Showcase*, pages 363–372, Berkeley, CA. Usenix.

[Bass et al., 2003] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional.

[Burgess, 2005] Burgess, M. (2005). A tiny overview of cfengine: Convergent maintenance agent. In *Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems*. MARS/ICINCO.

[Dart, 1991] Dart, S. (1991). Concepts in configuration management systems. In *Proceedings of the 3rd international workshop on Software configuration management*, pages 1–18, New York, NY, USA. ACM.

[Dolstra et al., 2005] Dolstra, E., Bravenboer, M., and Visser, E. (2005). Service configuration management. In *SCM '05: Proceedings of the 12th international workshop on Software configuration management*, pages 83–98, New York, NY, USA. ACM.

[Dolstra and Hemel, 2007] Dolstra, E. and Hemel, A. (2007). Purely functional system configuration management. In *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 1–6, Berkeley, CA, USA. USENIX Association.

[Fisk, 1996] Fisk, M. (1996). Automating the administration of heterogeneous lans. In *LISA '96: Proceedings of the 10th USENIX conference on System administration*, pages 181–186, Berkeley, CA, USA. USENIX Association.

[Fitzgerald and Olsson, 2006] Fitzgerald, B. and Olsson, C. M. (2006). The software and services challenge. Technical report, Contribution to the preparation of the Technology Pillar on "Software, Grids, Security and Dependability" of EU:s 7th Framework Programme.

[Kycyman and Wang, 2004] Kycyman, E. and Wang, Y.-M. (2004). Discovering correctness constraints for self-management of system configuration. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 28–35, Washington, DC, USA. IEEE Computer Society.

[Moffett and Sloman, 1993] Moffett, J. D. and Sloman, M. S. (1993). Policy hierarchies for distributed system management. *IEEE JSAC Special Issue on Network Management*, 11(9).

[NixOs, 2008] NixOs (2008). Webpage. http://www.nixos.org/.

[Oppenheimer et al., 2003] Oppenheimer, D., Ganapathi, A., and Patterson, D. A. (2003). 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*, pages 1–1, Berkeley, CA, USA. USENIX Association.

[Walsham, 1993] Walsham, G. (1993). *Interpreting Information Systems in Organizations*. John Wiley & Sons, Inc., New York, NY, USA.

[Walsham, 1995] Walsham, G. (1995). Interpretive case studies in is research: Nature and method. *European Journal of Information Systems*, 4:74–81.

[Wang et al., 2003] Wang, Y.-M., Verbowski, C., Dunagan, J., Chen, Y., Wang, H. J., Yuan, C., and Zhang, Z. (2003). Strider: A black-box, state-based approach to change and configuration management and support. In *LISA '03: Proceedings of the 17th USENIX conference on System administration*, pages 159–172, Berkeley, CA, USA. USENIX Association.

[Zheng et al., 2007] Zheng, W., Bianchini, R., and Nguyen, T. D. (2007). Automatic configuration of internet services. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 219–229, New York, NY, USA. ACM.