Comparison of Configuration File Parsers and Systems

Seminar aus Programmiersprachen WS2014

Roman Decker, B. Sc.
Göttweigersiedlung 5
3161 St. Veit/Gölsen
e0828572@student.tuwien.ac.at

ABSTRACT

In the configuration management field, various solutions exist to automatically configure a set of network nodes. Especially when it comes to editing existing configuration files that are in a potentially unknown state, *file editing* capabilities of these systems are in demand. This paper focuses on these capabilities especially. The configuration file needs to be parsed, possibly altered, and written back to disk. This paper first goes into some popular configuration file formats, then presents some parsers that are able to understand these formats before it analyses three popular configuration management systems.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software configuration management*

General Terms

Management

Keywords

configuration file parsers, configuration management

1. INTRODUCTION

File editing is an important aspect of configuration management systems tightly coupled to file parsing. It is important to understand the connection these systems have to configuration file parsers in order to reasonably evaluate them.

In order to compare various configuration file parsers, we must first select the file formats to parse. To do this, two main criteria are chosen: To adhere to a practice-oriented approach, file formats are selected according to their popularity within the software engineering/system administration community. To create a coherent link between the configuration management systems described in the second part of this report, file formats are further selected with regard to them being supported by the chosen systems.

Particularly on windows systems, `INI`-Files are a prominent choice for configuration storage. In the Java-community, a popular format for storing configuration is the `.properties`-file [8]. While `XML` is not ideal for editable configuration files [15], it is still a popular choice because its hierarchical structure makes it a good medium for configuration exchange.

For the configuration management system section of this paper, three popular systems were chosen: CFEngine, Puppet and Chef. CFEngine is one of the first configuration management systems [6]. Mark Burgess, most noted for his work in policy-based configuration management and creator of CFEngine, has produced a number of scientific papers backing CFEngine's practical use [4], [5] [3].

Alongside CFEngine, other configuration management systems such as Bcfg2, Puppet and Chef have arisen. Other solutions exist, but these projects are considered to be the "big four" of configuration managment [13]. Especially Puppet is relevant in the context of this paper, as it provides direct editing of various configuration file formats via the configuration-management library "Augeas". Because of this, Puppet is selected for further inspection in this paper.

2. CONFIGURATION FILE FORMATS

This section explains the file formats studied in this paper, giving a detailed description of their syntax and use. Table 1 gives an overview of the studied configuration file formats.

2.1 Properties of configuration files

Configuration file formats may be categorized by a diverse set of features, such as their ability to support comments alongside the actual configuration data, case-sensitivity of keys or their inherent structure.

The following criteria have been chosen to differentiate the chosen configuration file formats:

- Structure (Tree-Like, simple Key-Value, etc.)
- Support for comments
  - Placement of comments
- Conservation of formatting
  - Case-sensitivity in keys
  - Whitespace in and around values

Comparison of Configuration File Parsers and Systems
Seminar aus Programmiersprachen
185.307, SE, 2014W
Roman Decker, Matrikelnr.: 0828572
e0828572@student.tuwien.ac.at

2.2 INI-file format

The INI file format is a popular configuration file format dating back as early as Windows 3.1 [12]. Probably due to its simplicity, no standard exists for the file format. An INI-file is basically a list of key-value pairs. These pairs can (optionally) be organized into *sections*. In the actual file, configuration entries are restricted to one line, the typically case-insensitive key and its value are separated by an equal-sign, sections are marked by putting the respective section-name in square brackets and are valid until the next section-definition or end of file is reached. Comments may be specified on separate lines by prefixing the line with a semicolon or a number sign ("#"). A sample INI-file can be seen in Listing 1.

Listing 1: A sample .ini-file

```
1 [foosection]
2 foo=foovalue
3 bar=barvalue
4
5 ; Here goes the bar section
6 [barsection]
7 bla=4711
```

INI has spawned a number of dialects, that largely correlate with this format and only bear subtle differences. An example for such a format would be *Subversion*'s configuration file format which allows the number sign ("#") as a comment-starting symbol.

2.3 Properties-file format

.properties-files pose another very simple way for applications to store their configuration. They are essentially a plain list of key-value pairs. Other than the aforementioned INI-files, .properties-files do not support sectioning. Keys and values can be separated in various ways, either by a double-colon, an equal-sign or simply by one or more whitespace characters (other than a newline). By suffixing a line with a backslash, values may span multiple lines. Comments are also possible on a one-line basis by starting the line with a number sign ("#") or exclamation mark. Listing 2 shows a sample properties file.

Listing 2: A sample .properties file

```
1 bar.foo = 22
2 # this is a comment
3 foo.bar.another = 4711
4
5 multiline = this is a longer entry \
6       spanning multiple \
7       lines
```

2.4 XML files as configuration storage

XML provides an inherent hierarchical structure to organize configuration entries. A wide variety of tools supports XML, such as *XPath* to query and select elements and attributes, Schema-Validation to easily validate configuration files or *XSLT*[1] to transform data from one format to another. As XML-Elements are typically case sensitive, this property propagates to configuration files using this format. Listing 3 depicts a sample XML-file.

[1]E**X**tensible **S**tylesheet **L**anguage

Table 1: Comparison of chosen file formats

Format	Structure	Comments	Key case
INI	Key-Value, sections	Single line	Insensitive
Properties	strict Key-Value	Single line	Sensitive
XML	Tree	Arbitrary [3]	Sensitive

2.4.1 XML-Comments

Comments in XML are denoted by the comment tag, which starts with "<!--" and ends with "-->". Any text in between will be considered a comment. However, the string "--" must not occur within the comment.

The fact that XML-comments are just another XML-Element gives them the advantage of simply being nodes in the *DOM*-Tree[2]. That makes it easier for parsers to preserve them.

3. CONFIGURATION FILE PARSERS

This section compares chosen configuration file parsers with each other. Detailed results can be found in Table 2 in Section 6.

A popular choice for parsing configuration files in Java is the Apache Commons Configuration library, which is licensed under the Apache license[4]. It supports all of the chosen configuration file formats natively and runs on Java 5 or greater.

Another viable tool for parsing configuration files of different forms is Augeas, a configuration management library licensed under LGPL. In Augeas, configuration formats are converted to an internal representation using Lenses. Augeas supports all of the chosen formats natively through stock lenses[5] that come with Augeas.

The typical task of these libraries in the context of configuration management systems is to read a configuration file, eventually adapt its content and write it back to disk ("Round-trip"). The aforementioned parser libraries are evaluated according to a list of the following criteria with regards to their round-tripping properties:

- (C1) Preserve comments
- (C2) Preserve order
- (C3) Preserve case of case-insensitive keys
- (C4) Preserve formatting

For each criteria, a specialized test-case is crafted, along with its desired outcome in case of passing the test. Listings 3 and 4 illustrate a testcase for whether an XML-Parser preserves comments when updating keys paradigmatically.

Listing 3: Test case for preserving comments

```
1 <?xml version="1.0"?>
2 <root>
3  <someelements>
4   <element>
5    <!-- This is a comment -->
6    <foo>foovalue<!-- nested --></foo>
7    <bar>barvalue</bar>
8   </element>
```

[2]**D**ocument **O**bject **M**odel
[3]Single- and multi-line comments, see Section 2.4.1
[4]http://www.apache.org/licenses/, 2015-01-13
[5]see Section 3.2 for an explanation on lenses

```
  </someelements>
</root>
```

Listing 4: Desired outcome for Listing 3

```
<?xml version="1.0"?>
<root>
 <someelements>
  <element>
   <!-- This is a comment -->
   <foo>changed<!-- nested --></foo>
   <bar>barvalue</bar>
  </element>
 </someelements>
</root>
```

3.1 Apache Commons Configuration

The Apache Commons Configuration library supports 9 configuration formats including the three formats examined in this paper. As the library is a Java framework, it is interfaced through Java classes. Each format has its own class that extends from a base configuration class[6]. Custom formats may be added by extending from that base class. Apache Commons Configuration comes packaged as a `.jar`-file, thus simple, minimalist Java-Applications were written for each test case.

3.2 Augeas

The Augeas configuration management library is written in C and comes with a variety of bindings for different languages.

Augeas' powerhorses are its so-called "lenses": Little programs written in Augeas' own language that transform a file format from and to Augeas' internal tree-representation [14]. Lenses work bidirectionally, so that writing the internal tree-representation back to its file format becomes possible. Augeas comes with a large number of stock-lenses supporting various file-formats out of the box. However, it is possible to create custom lenses for additional formats.

In order to evaluate Augeas, the `augtool` command line tool is used that comes bundled with Augeas. The workflow typically consists of loading a configuration file, applying some changes and writing the changes back to the file.

A typical test run looks as depicted in Listing 5.

Listing 5: A sample testrun with augtool

```
$ augtool --noload \
          --noautoload \
          --echo \
          --root .
> set /augeas/load/xml/lens "Xml.lns"
> set /augeas/load/xml/incl "comments.
    xml"
> load
> set /files/comments.xml/root/
    someelements/element/foo/#text "
    changed"
> save
Saved 1 file(s)
```

[6]The `AbstractConfiguration` class, see `https://commons.apache.org/proper/commons-configuration/apidocs/org/apache/commons/configuration2/AbstractConfiguration.html`

Augeas was able to perform most tasks to a satisfactory degree. The best format to use in combination with Augeas is arguably `XML` which bears big similarities to Augeas' internal intermediate-representation due to its hierarchical structure. However, Augeas was also able to handle `INI`- and `Properties`-files well, with the little exception of treating all `INI`-sections and -keys case-sensitively. Augeas was not able to handle multi-line values in `Properties` files. `augtool` was able to read files with multiline-keys but wasn't able to write them.

3.2.1 Disparities

Another interesting difference between the two parsers is the way they handle duplicate sections in `INI`-files. Consider the following test case in Listing 6.

Listing 6: Section unification test case

```
[foosection]
foo=foovalue
bar=barvalue

[barsection]
bla=4711

;same section name here
[foosection]
bar = duplicate
baz = bazvalue
```

While Augeas preserved the two `foosection` sections (with a `bar` key in each section), Apache Commons Configuration unified the two sections to one single section at the location of the first occurence. The duplicate `bar` key then also appeared twice in the merged section. Listing 7 shows the output generated by Apache Commons Configuration.

Listing 7: Section unification test case output for Apache Commons Configuration

```
[foosection]
foo = foovalue
bar = barvalue
bar = duplicate
baz = bazvalue

[barsection]
bla = 4711
```

4. CONFIGURATION MANAGEMENT SYSTEMS

This section will describe three widely used configuration management systems, *CFEngine*, *Puppet* and *Chef*.

4.1 CFEngine

CFEngine is a rule-based configuration management system first conceived in 1993 by Mark Burgess. It is written in C and licensed under GPL. Its aim is to centralize configuration of a large number of heterogenous client systems in a network. Burgess summarizes its functionality as follows [6]:

- Testing and configuration of network files
- Simple automated text file editing
- Symbolic link management

- Testing and setting the permissions and ownership of files
- Systematic deletion of garbage files
- Systematic automated mounting of NFS filesystems
- Other sanity checks

What differentiates CFEngine from most configuration management systems is its stochastic philosophy of system evolution [4]. CFEngine aims to make a system *converge* to an ideal state, taking into account random entropy through user interaction. This Process of *healing* a system towards a healthy state is referred to as *immunity model*. CFEngine tries to converge this fuzzy concept [5] of a state to the ideal state given by a *Policy*. This policy (rule-)based evaluation resembles logic programming languages such as Prolog [7], [5].

CFEngine's declarative configuration language consists of one or more statements each of which follow the same structure [6]. A sample configuration snippet is shown in listing 8.

Listing 8: Sample CFEngine configuration file

```
bundle bundle_type name
{
promise_type:

classes::

"promiser" -> { "promisee1", "promisee2
    ", ... }

     attribute_1 => value_1,
     attribute_2 => value_2,
     ...
     attribute_n => value_n;
}
```

These files organize *Promises* into *Bundles*. A *Promise* is a declarative statement about the desired state of the system. A *Promise*'s use is restricted by *Classes*, boolean classifiers that describe system context [16].

Relevance to configuration file parsers.

CFEngine provides good support for editing `XML`-documents via its `edit_xml` bundle. Similarly, CFEngine also provides some `JSON`-parsing functionality for editing `JSON`-Files via an internal `JSON` parser. However, this is where custom file format support ends. Except some helper functions for handling `INI`-files, users have to resort to CFEngine's (albeit very powerful) Regular-Expression-based file editing tools. As the *nix landscape mainly uses manifold, but simple, line-based configuration formats, this appears to be a very practical approach.

Documentation.

Older versions of CFEngine did not have very good documentation. However with more recent versions, CFEngine has gotten a thorough documentation available at `https://docs.cfengine.com/latest/`.

4.2 Puppet

Puppet is another open source project dedicated to configuration management. The Ruby-application is licensed under an Apache License (older versions used to have GPL licensing).

Puppet is interfaced through the `puppet` command-line tool. It is used to manipulate a system's *resources*, i.e. Puppet's concept of an atomic configuration unit [1]. *Resources* can be manifold, for example:

- User accounts
- Files
- Directories
- Services
- etc.

Puppet then provides a Resource Abstraction Layer (referred to as RAL) which allows administrators to form declarative statements about these resources (e.g. *"make sure a user account with a specific name exists"*). These statements are written in Puppet's configuration language or domain-specific language and are saved to *manifest* files. Puppet will then apply these manifests by executing the code within them.

In order to configure a network of hosts with Puppet, one must run a Puppet master server on a host within the network. The hosts to be configured by Puppet then need to run a Puppet agent, which will periodically fetch manifests from the Puppet master and apply them.

Just like CFEngine, Puppet offers a declarative way of specifying a configuration with its own JSON-like configuration language. However, since Puppet Version 2.6, configurations may also be written in a domain-specific language, a feature Ruby seems to be built for [11].

Puppet's configuration language supports modules and classes, which enables a very modular approach. This eases maintenance considerably.

A sample puppet manifest is depicted in 9[7].

Listing 9: A sample puppet manifest

```
file {'/tmp/test1':
 ensure  => present,
 content => "Hi.",
}

notify {'/tmp/test1 has already been
    synced.':
 require => File['/tmp/test1'],
}
```

Similar to CFEngine, Puppet runs on most Unix variants and Windows. However, a functional installation of Ruby is required on the Puppet master as well as every Puppet agent. Mac OS X is not explicitly stated as being supported in the Puppet documentation, however it is listed in Puppet's installation guide under "other Unix". Furthermore, there exists documentation on how to handle Mac OS X peculiarities when rolling out a Puppet system.

Relevance to configuration file parsers.

Puppet uses the Augeas configuration management library mentioned in 3.2 for editing configuration files. Due to its tight coupling to the library, puppet has contributed to the implementation of Augeas as well.

[7]Taken from `https://docs.puppetlabs.com/learning/ordering.html`

4.3 Chef

Like Puppet, Chef is mainly written in Ruby and is licensed under an Apache License. Configuration instructions are called "*recipes*", which are in turn collected in *Cookbooks* on a chef server. Recipes are then delivered to hosts who run a chef client that will take care to configure the node.

Chef also supports a mode of operation for single host systems called `chef-solo`.

In contrast to Puppet, which provides its own configuration language to implement manifests, chef recipes are written in a Ruby domain-specific language only. This language is not declarative but imperative, which changes the viewpoint from which administrators create configuration recipes. Chef's domain-specific language acts more like a library helping Ruby code interface with configuration artifacts.

Relevance to configuration file parsers.

Other than Puppet, which allows format-aware editing of configuration files, Chef's editing capabilities are restricted to simple regular expression-based search-and-replace operations.

4.4 Comparison

While CFEngine's viewpoint of configuration management is strongly orientated towards a developer's view, Chef and Puppet are both more directed towards use by system administrators [2].

CFEngine shines when it comes to memory and CPU footprint. As it is written in C, it does not require a runtime-environment and thus has less dependencies than its counterparts Puppet and Chef. Puppet and Chef are both written in Ruby and thus require a Ruby environment to be present on every network node.

5. NIXOS' APPROACH TO CONFIGURATION MANAGEMENT

A problem all of the systems detailed in Section 4 suffer from is statefulness: When a local configuration file has to be edited, its state is potentially unknown [10].

5.1 The nix package manager

Solving this dilemma requires an altogether different approach to configuration management. The `nix` package manager is a tool that uses a purely functional approach to package management. In contrast to popular package managers (like `apt` or `dpkg`), `nix` uses a purely functional language for package configuration.

5.2 NixOS

NixOS is a Linux distribution further applying the concept of functional configuration management across an entire operating system consistently. Based on the `nix` package manager, NixOS maintains a single configuration file that is edited when the configuration needs to be changed [10]. NixOS then uses the `nix`-package-manager to build the configuration from scratch. This approach has the following advantages:

- Statelessness: Rebuilding the configuration with the same base configuration file will always produce the same result, regardless of currently installed packages.
- During upgrades, the system is not in an inconsistent state, as upgrades are atomic.
- Because configuration files are never overwritten, the system can easily be rolled back to a different configuration, even in constant time [9].
- Different versions of the same package can coexist. This enables NixOS to allow non-root-users to install packages without breaking other users' system configuration.

5.2.1 *Disadvantages*

The fact that concurrent versions of the same package will be kept until no other package depends on them (even though dependants would work with different versions of the package as well), leads to a bigger storage requirement. The same reason can also cause build times to increase considerably.

Table 2: Comparison of configuration file parsers

Parser	C1	C2	C3	C4
INI				
`Apache Commons Configuration`	No	Partly[8]	Yes	Yes
`Augeas (Puppet lens`[9]`)`	Yes	Yes	No	Yes
Properties				
`Apache Commons Configuration`	Yes	Yes	n/a	Yes
`Augeas`	Yes	Yes	n/a	Yes
XML				
`Apache Commons Configuration`	Yes	Yes	n/a	Yes
`Augeas`	Yes	Yes	n/a	Yes

6. CONCLUSION

In terms of parsers, both Apache Commons Configuration and Augeas were able to perform tasks to a satisfactory degree. Apache Commons configuration lacked support when it comes to preservation of INI-comments. Augeas on the other hand was not able to write multi-line-values in `.properties`-files. Table 2 lists detailed evaluation results.

When it comes to configuration management systems, there is no real "winner". CFEngine, Puppet and Chef all have their advantages: While CFEngine clearly outperforms Puppet and Chef in terms of Footprint and performance, its counterparts are more intuitive when it comes to learning the system.

Configuration management is evolving steadily in these times, with multiple projects competing for market dominance. CFEngine remains a viable option, having the best theoretical background and a tiny footprint while being just as powerful as its competitors.

Puppet provides very good documentation and is probably the easiest tool to learn. Its configuration language is expressive and readable, which further increases productivity.

Chef has its similarities to puppet (language, license), the most important difference is the fact that Chef's recipes are

[8] see Section 3.2.1

[9] Due to the fact that `INI`-Files have various manifestations, Augeas has multiple lenses for editing `INI`-files. The testcases here were run using the "Puppet"-lens.

[10] see Section 4.2

Table 3: Comparison of configuration management systems

Configuration management system	Configuration Language	License	Platforms
CFEngine	Declarative	GPL	Linux, Mac OS X, Windows
Puppet	Declarative	Apache	Linux, Mac OS X (partial[10]), Windows
Chef	Imperative	Apache	Linux, Mac OS X, Windows

written in an imperative language. One thing to be considered for the usage of both Puppet and Chef is the need for a Ruby environment.

On the other hand, NixOS sketches an entirely different approach to configuration management. Whether this idea will prevail is still to be determined.

References

[1] Learning puppet. `https://docs.puppetlabs.com/learning/ral.html`, 2014.

[2] M. Baukes. Puppet vs. cfengine. `http://www.scriptrock.com/blog/puppet-vs-cfengine`, 2014.

[3] M. Burgess. Evaluating cfengine's immunity model of site maintenance, 2000.

[4] M. Burgess. Recent developments in cfengine. In *In Proceedings of the 2nd Unix.nl conference*, 2001.

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

[6] M. Burgess and O. College. Cfengine: a site configuration engine. In *USENIX Computing systems, Vol*, 1995.

[7] A. L. Couch, Dr. and M. Gilfix. It's elementary, dear watson: Applying logic programming to convergent system management processes. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 123–138, Berkeley, CA, USA, 1999. USENIX Association.

[8] V. S. Desinov. Overview of java application configuration frameworks. *International Journal of Open Information Technologies*, 2013.

[9] E. Dolstra. *The Purely Functional Software Deployment Model*. PhD thesis, Universiteit Utrecht, Jan. 2006.

[10] E. Dolstra, A. Löh, and N. Pierron. Nixos: A purely functional linux distribution. *J. Funct. Program.*, 20(5-6):577–615, 2008.

[11] S. Günther. Engineering domain-specific languages with ruby. 2009.

[12] M. Inc. Windows 3.1 resource kit win.ini section settings. `http://support.microsoft.com/KB/83386`.

[13] C. Lueninghoener. Getting started with configuration management. *;login:*, 2011.

[14] D. Lutterkort. Augeas - a configuration api. *Proceedings of the Linux Symposium*, 2008.

[15] M. Raab. A modular approach to configuration storage. Master's thesis, Technische Universität Wien, 2010.

[16] T. Zlatanov. Language concepts. `https://docs.cfengine.com/docs/3.5/manuals-language-concepts.html`, 2014.