# TemplateTree II: The Post-Installation Setup Tool

*Tobias Oetiker* – ISG.EE, Swiss Federal Institute of Technology

## ABSTRACT

After installing an OS distribution, a computer is generally not yet properly adapted to go into production at the local site. Security policies must be implemented, local services configured, and standard application settings deployed. Solutions to this problem range from unpacking a tar archive in the root directory to sophisticated tools like Cfengine [3].

This paper presents TemplateTree II, a highly modular approach for solving the post-installation problem which uses Cfengine as a transport mechanism.

## Introduction

System management of an IT-infrastructure with more than just a few machines, requires an overall management concept and a lot of automation to preserve the sanity of all involved. This has been discussed at length, for example, in the Infrastructure Paper [1] by Traugott and Huddelston and and also a few years back by Rémy Evard at LISA 1997 [2].

One of the problems to tackle is the post installation procedure of freshly installed machines as well as feeding and care once they are in operation. This paper presents a modular solution to this problem. Another related problem is software distribution which is not covered in this paper (see [8] for our solution to this problem).

## The Problem

Integrating a machine into the local environment requires that configuration files are replaced with customized versions for the local site, or even that whole new software is added to the machine. Files like /etc/services, /etc/inetd.conf, and /etc/mail/sendmail.cf come to mind, but also third party packages like AFS (Andrew File System), SSH (Secure Shell) or Postfix.

In principle this problem is quite easy to solve: Setup some sort of Master Server which holds a copy of all the changes you want to apply to a client and let the client update itself by copying all the changes from the Master Server. The Infrastructure Paper calls this machine the 'Gold Server.' Manage the material on the Master Server using CVS and you even have reproducibility and accountability.

Unfortunately, most often not all the clients are the same, so there must be a method to define what material from the Master Server should go to each client. A simple approach taken by many, is to write a special customization script which is run on each machine and figures out what should be done depending on information it gleans from the machine at runtime.

One tool which takes this approach to a new level is Mark Burgess' Cfengine [3]. Essentially it provides a highly specialized language for describing machine configurations, so instead of writing customization scripts in Perl or bash it is now possible to write them in a language created for this very purpose. I am comparing TemplateTree II mostly with Cfengine because it is the most widely used tool in this area.

At first we thought that Cfengine would solve all our problems, but unfortunately we found three main areas where it did not fit our requirements:

**Modularity and meta configuration**: Because our setup is quite diverse with many different configurations we wanted a system where we can build configurations based on individual components. Each component would expose certain configurable parameters which allow to tune the component to the setup it is going to be used in. One could talk about an additional abstraction layer. Cfengine has no concept for modularization apart from its ability to use include files. The include files can not define a configuration interface for themselves.

**Documentation**: The best system can be difficult to understand if no documentation goes along with it explaining the reasoning behind the less obvious configuration decisions. All you get in Cfengine configuration files are embedded comments.

**Disk-less client support**: We use disk-less clients whenever possible. Because all client filesystems are stored on the server we can update them even when the client is not running. Cfengine has been designed for running locally, this means that writing a Cfengine configuration file for our setup is more complex than necessary. Especially, there is no support to change the 'root' directory within a Cfengine configuration file.

Despite the missing features we found Cfengine to be a great tool for the task of actually doing the necessary modifications on the target machines. It can serve both as a network transport and as an all dancing and singing file handling tool. We decided to implement a configuration system on top of Cfengine which outputs Cfengine configuration files. This allows us to use Cfengine as a back-end system without having the problems outlined above.

## Concept of a Solution

TemplateTree II addresses all the problems mentioned above. It provides a post-install host configuration system based on *Feature-Packs*.

### Modularity Through Feature-Packs

Modifying a freshly installed machine to fit the local requirements normally consist of several loosely coupled tasks. These tasks are, for example: linking the machine into the local user authentication system, configuring the machines mail transfer agent, adding the latest OpenSSH distribution and turning off unnecessary services. In the context of TemplateTree II these independent tasks are called *Feature-Packs*.

Feature-Packs are self-contained in the sense that you can mix and match Feature-Packs from a central repository. Several *system management domains*[1] could share a single repository. This is similar to the *independent packages* approach chosen for the SEPP software distribution System [8] or the Classes of Synctree [4] or even Cfengine configuration include files.

### Splitting Configuration and Code

The modifications necessary to make a machine fit into the local setup may be similar across many machines. Nevertheless, some differences between machines will exist. If a single Feature-Pack should be able to cater for all these situations it must itself expose a configuration interface. For a mail server you want to be able to set the local mail domain, for an automounter setup the automount maps may differ between departments. In publishing, one of the hot topics is separating content from design [9]. In TemplateTree II we might call it separation of configuration from code.

Obviously this is not new. Most software packages support some sort of configuration file and you do not have to recompile emacs to change the size of a font. Therefore, in our case we might talk about a unified configuration level which sits above the normal application configuration files.

[1]A system management domain is a a set of machines managed by a single system management group. This can be a single server with a few clients or a complex setup with many servers and clients. Size does not and should not matter from a technical point of view. TemplateTree II will fit both. What we have here is more a political and organizational question. I avoided to use the word infrastructure, because its definition of encompassing all machines of a whole organization is not really appropriate in our institution with thousands of machines and many independent system management groups.

TemplateTree II implements such a meta configuration level. This has two advantages: First, a single Feature-Pack can be deployed across many different machines in various configurations. Second, the relevant meta configuration information is kept separate from the Feature-Packs and is therefore more manageable.

### Collaboration

With TemplateTree II it is possible to maintain a central repository of Feature-Packs. A group of system managers can work together keeping them up to date. Each maintains a number of Feature-Packs in the central store, specializing in some areas. When it comes to defining what Feature-Packs to use in a certain system management domain, each of the participating managers has his full freedom, as to which Feature-Packs he wants to use and how he wants to configure them. This potential for collaboration is quite similar to SEPP [8].

### Centralized Management

Having a way to easily customize machines is not enough. We also need to manage the configuration information in an efficient way. We wanted an efficient method for writing a single configuration file per management domain. A single configuration file for all machines is more efficient to maintain and has less redundancy than a system with large amounts of configuration data. Configuration files of other tools in the same problem space like Cfengine [3] tackle this problem by implementing whole scripting languages in their own right. In the case of Ressmans paper from LISA 2000 a SQL database [10] is holding all the necessary information. For TemplateTree II we chose a configuration file centering on which Feature-Packs to apply to which group of machines and how to configure the Feature-Packs. This gives us all the configuration freedom we need while still being quite simple because the complexity is locked away into the Feature-Packs, while the configuration information remains in the central configuration file.

### Documentation

System management concepts and tools differ largely from site to site, so there is no official book for folks to read in order to get up to speed on working in our environment. To make sure the documentation gets written and updated as part of our daily routine, we tightly integrate facilities for documentation into all our system management tools (see [8]).

Being able to turn a machine from "freshly installed" into "useful workstation or server" in a short time, is nice. But this is only half the bill when either something fails or when several people work together on the task. The other half is having good documentation regarding the changes done to a machine. Not only do we want to know what has been changed but also why it has been done. TemplateTree II defines a mandatory documentation standard for all Feature-Packs. TemplateTree II integrates the documentation into the Feature-Pack itself. Following the ideas of literate programming [6], it is possible to automatically create a big POD[2] file, documenting all the Feature-Packs. This means that when you want to use a Feature-Pack you will get full documentation about what the Feature-Pack does, how you can use it, and any special points to observe when applying it to your setup.

### Disk-Less Clients or "No Magic Please!"

A major feature of Cfengine is, that you can write configuration files which react to the setup and current state of the local machine. In his Computer Immunology paper [5] Burgess uses this facility to illustrate how a self healing mechanism for computers could be implemented. Our setup contains many disk-less clients where we build the client filesystem on the server even before the client boots for the first time. This means TemplateTree II must be able to run without access to the machine it is customizing. Therefore all the information it needs is available in its configuration files.

Cfengine provides facilities for monitoring machines and even for reacting to certain problems. The scope of TemplateTree II is more focused.[3] Its only purpose is to modify the configuration of a machine to make it fit the local requirements. This task is completely controllable. No evaluation of the status of a certain machine is necessary in order for TemplateTree II to do its work. We know what machines we have and how they are configured.

If some configuration must be done locally and while the client is running, there is always the option of applying a specialized boot script to the client or to add an appropriate cron-job.

[2]POD is a very simple documentation format widely used in the Perl community. It can be converted into Man, HTML, LaTeX, and other formats. We use it for most of our technical documentation (see [12]).

[3]Monitoring the health of the system is left to a specialized tool in this area (Gossips [11]). Note that using TemplateTree II does not prevent you from using cfengine as an immunological agent. It only means that TemplateTree II will not make use of these aspects of cfengine.

### Getting the Modifications to the Client

TemplateTree II uses Cfengine as a transport mechanism for moving and applying files to the target machine. We decided to use Cfengine because it provides all the file tackling equipment required for what we intend to do in one simple binary. It also allows us to use all the neat features of Cfengine like the Cfengine daemon or its ability to only copy those files which have changed or to do a dry-run in order to test a new configuration.

TemplateTree II outputs a single Cfengine configuration file per management domain. This configuration file contains all the information necessary for configuring each individual machine, as well as the root directories of the disk-less clients.

## Architecture

Figure 1 shows the main components of TemplateTree II. The configuration is stored in three main configuration files:

**system.conf** defines where TemplateTree II should look for other components of the system, which operating systems your installation supports, and which attributes must be known for each host managed by the system. The system configuration does not have to be changed under normal circumstances and is therefore kept in its own file.

**site.desc** contains structured information about how hosts should be arranged into groups, which Feature-Packs should be installed on which group of hosts, and how each feature should be configured.

**host.list** holds a simple table with a set of basic attributes for each host. The Feature-Packs can draw upon this information in addition to specific 'per pack' configuration parameters.

The actual files which have to be applied are stored in a repository of *Feature-Packs*. A Feature-Pack is a directory containing all the files which must be applied plus a file called META which describes how to apply the files.

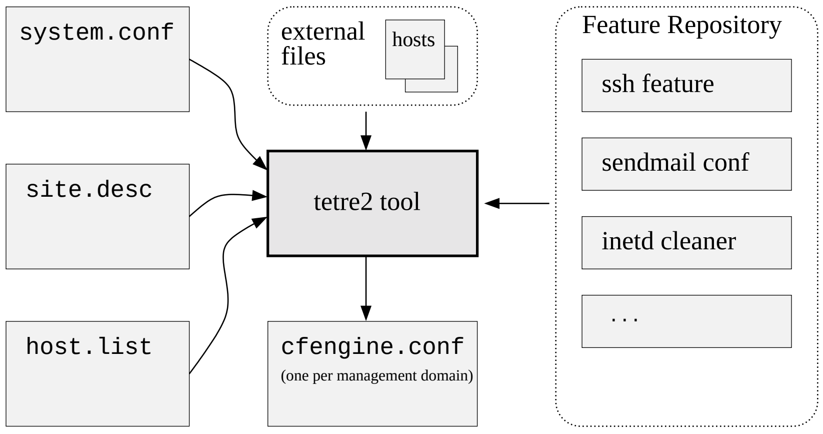

**Figure 1**: The components of TemplateTree II.

Based on the configuration files and information taken from the selected Feature-Packs, the tetre2 utility builds a cfengine.conf file.[4] The Cfengine configuration will contain references to files from the Feature-Packs repository if whole new files must be copied as part of the post-installation process. Apart from this, the generated Cfengine configuration will not depend on TemplateTree II anymore. Together with the Cfengine daemon it is possible to update the files on any machine regardless whether it shares a common filesystem with the machine where TemplateTree II is installed or not. It only needs the Cfengine binary to work and network access to the Cfengine daemon.

For large changeable files like /etc/hosts a special facility is provided to keep these files separate from the Feature-Packs. A Feature-Pack can in fact define that it wants to use such a file. TemplateTree II will then make sure that this file is provided.

## Using TemplateTree II

### Configuration Splitting

Probably the most complex task when starting to use TemplateTree II is to split your setup into a set of Feature-Packs. The goal is to devise a scheme which allows to use different combinations of your Feature-Packs to cater for all the special needs at your site.

Our approach is to use four different types of Feature-Packs:

**Application**: This is for essential applications like OpenSSH or Xinetd which are not part of the basic OS distribution, but which we think should be.

**OS**: Here we collect all the OS dependent changes we apply to every machine if it runs a certain OS release.

**Domain**: All machines within the same authentication domain usually share some files and configurations.

**Machine**: And finally there are always some bits and pieces of configuration which are special to an individual machine. For these we create individual, machine dependent, Feature-Packs.

The site.desc configuration file then defines which combination of Feature-Packs has to be applied to each machine.

### Anatomy of a Feature-Pack

A Feature-Pack is a directory containing all the files necessary to implement a certain functionality or behavior on the target machine. For an OpenSSH Feature-Pack this would be all the binaries for all the architectures the Feature-Pack is going to support, a startup script, and configuration files defining the site policy.

[4]The architecture of the tetre2 application allows to quite easily add other output formats apart from Cfengine configuration files like shell scripts, for example. TemplateTree II already has the ability to output POD documentation on all the Feature-Packs available in the current repository.

```
 :
 +- features
 |   |
 |   +- sendmail_config-1.0-to
 :   |   +- META
     |   +- sendmail.cf-client
     |   +- sendmail.cf-server
     |
     +- openssh-2.9.4-to
     :   +- META
         +- ...
```

**Figure 2**: A Sample Feature-Pack repository.

In addition to this free-form collection of files every Feature-Pack must contain a file called META. It describes the content of the Feature-Pack and contains all the instructions required to apply the Feature-Pack to a machine. Figure 2 shows the directory layout of a Feature-Pack repository. The META file offers six ways to expose configurable items from the Feature-Pack:

**File Sets**: The content of a Feature-Pack can be split into blocks. Each block must have a distinct name. In TemplateTree II such a block is called File Set. When applying a feature which uses File Sets, the administrator can choose which File Set to apply to which Machine.

**Operating System**: Template Tree knows for each machine it manages, which OS is running on that machine. A Feature-Pack can be configured to install different files depending on the release and type of OS it is running on.

**Substitutes**: Some files are the same for all machines except for one word, which has to be changed according to, e.g., the name of the default printer or the mail domain. A Feature-Pack can expose several named parameters which can be configured when using the Package.

**Automatic Substitutes**: The host.list file contains a number of key parameters for each machine. A Feature-Pack can access these parameters as search and replace data on files it contains.

**Magic Substitutes**: The value of a Substitute can be subjected to some perl manipulation prior to being used in a search and replace operation.

**External Files**: A package can "contain" external files. The Feature-Package only knows how to apply these external files to the target machine. The name and location of these files is configurable in the site.desc file when using the Feature-Pack.

The next page shows a sample META file. It shows how to use the functions listed above in context.

The `*** Action ***` section of this META file is very boring as it only contains `copy` instructions. TemplateTree II supports a number of other actions for creating directories, removing files and directories, generating symbolic links and, finally, a special function for assembling files.

```
*** Name ***
Sendmail Config Package

*** Version ***
1.0

*** Maintainer ***
Tobias Oetiker <oetiker@ee.ethz.ch>

*** One Line Description ***
Configure Sendmail to work properly

*** Blurb ***
This package can configure sendmail as either a normal mail server or
as a null client simply feeding mail to a central mail server.

*** Usage Info ***
This package assumes that the stock sendmail version for your OS is
already installed. It does not contain any binaries, just an
appropriate configuration file.

*** File Sets ***
client     A configuration for a forward only client
server     The works, a full blown server with built-in jacuzzi

*** Change Log ***
2000/07/02  to  Demo package created
2000/08/10  to  Added change log

*** OS Support ***
sol26
sol8
rhl62

*** Substitutes ***
mdomain  Name of the local mail domain.
mserver  Host Name of our mail relay

*** External Files ***
aliases     Your site's alias file

*** Action ***

# the server gets an alias file. Which physical file
# gets copied to the server can be configured when setting
# up the Feature-Pack in the site.desc file.

server:.*:
C aliases      /etc/mail/aliases        644  root:root

# solaris mail servers get to use the file sendmail.sol.server as
# their sendmail.cf file. While copying the file to the server,
# cfengine will do a search/replace operation for >#>mdomain<#<
# and >#>mserver<#< according to the setup in the site.desc file

server:sol.*:
C sm.sol.srv /etc/mail/sendmail.cf 644  root:root   mdomain,mserver

# the same happens for RedHat mailservers except that there is a
# different sendmail cf file.

server:rhl.*:
C sm.rhl.srv /etc/mail/sendmail.cf 644  root:root   mdomain,mserver

# for null clients we do not need a os sepcific sendmail.cf file, so
# the OS part of the file selector is set to a globally matching
# regular expression.

client:.*:
C sm.clnt   /etc/mail/sendmail.cf 644  root:root mdomain,mserver
```

**Listing 1**: A sample META file.

The *assemble* function allows for different Feature-Packs to each provide part of a file which then gets *assembled* on the target machine. This can be used to configure /etc/system on a Solaris system where some Feature-Pack might want to add a special driver load instruction whereas on other machines there is just the usual shared memory and stack protection configuration in there. Another usage would be the root crontab file or the inetd.conf where several Feature-Packs contribute to the contents of the file.

### The Host List (host.list)

The most simple configuration file in a TemplateTree II setup is the host.list file which contains a simple table with all the hosts of the site. It is shown in Listing 2. The third row in the sample above is for the disk-less machine called *bluehat* which uses disk-space on the server *drwho*. Some of the columns in the table like the host name and the OS are required by TemplateTree II, others are configurable through the system.conf file.

### The System Configuration (system.conf)

At the root of the TemplateTree II configuration setup is the system.conf file. It defines where the other components of the system are stored, what OSes are handled and which columns must be listed in the host.list file.

```
*** Locations ***
SiteDesc = /etc/tetre2/site.desc
HostList = /etc/tetre2/host.list
Features = /etc/tetre2/features
ExternalFiles = /etc/tetre2/extfiles
ConfServer = jobis.ee.ethz.ch
RunTimeVar = /var/cfengine

*** Operating Systems ***
sol26   Sun Solaris 2.6 Sparc
sol7    Sun Solaris 7 Sparc
sol8    Sun Solaris 8 Sparc
rh162   RedHat Linux 6.2 x86
irix63  SGI Irix 6.3 MIPS

*** Host List Config ***
HOST     Hostname
IP       IP Address
ETHER    ETHERNET Address
DEF_GW   Default Gateway
DOMAIN   DNS Domain of the Host
OS       OS of the machine
ROOT     Where is the ROOT of this machine

*** Host List Tests ***
DOMAIN   sub {return \
     "We only manage ethz domains" \
     unless $Match =~ /ethz/; 0 }
```

### The Site Description (site.desc)

With all the other parts of the system in place it is now possible to setup the Site configuration file defining which Feature-Pack should be installed on which machine and how it should be configured in the process. The site.desc file has three main sections:

**Feature Selection**: This allows to create instances of Feature-Packs. Each instance can contain several *cases*. A case lends its name to a group of configurable parameters. When applying a Feature-Pack these parameters can be activated by using the name of a *case*.

**Host Grouping**: Most hosts at a site share some common configuration. So instead of configuring each host individually you can build groups of hosts and then apply the Feature-Pack instances to the groups.

**Feature Application**: In the final section of the site.desc file the feature instances are applied to individual hosts or whole groups of hosts as defined in the previous section.

Below is a tiny Site Description sample file for illustration:

```
*** Feature Selection ***
#----------------------------------
SENDMAIL := sendmail_conf-1.0-to
#----------------------------------
  # default values
  mdomain = "ee.ethz.ch"
  mserver = "smtp.ee.ethz.ch"
  aliases -> "aliases/ee"

  # case 'mailserver' uses
  # the file-set 'server'
  mailserver:
        /server

  # case 'nullclient' uses
  # the file-set 'client'
  nullclient:
        /client

*** Host Groups ***

null = drwho bluehat

*** Host Features ***

tardis: SENDMAIL(mailserver)
@null: SENDMAIL(nullclient)
```

## Security Considerations

If a whole site is setup and configured using a centralized approach such as TemplateTree II the potential problems which arise are similar to those you get when all your fields carry the same crop. First, a

```
# HOST   IP            ROOT                         OS     DOMAIN
#--------------------------------------------------------------------
tardis  192.168.1.2   /                            Sol8   ee.ethz.ch
drwho   192.168.2.44  /                            Sol7   ee.ethz.ch
bluehat 192.168.2.12  drwho:/export/root/bluehat   Sol7   ee.ethz.ch
. . .
```

**Listing 2**: Table of hosts: host.list file .

security problem present on one host is likely to be present on all and second, if your central configuration machine gets compromised and the intruders modify the TemplateTree II setup then the malicious code could get distributed easily to all managed machines.

We counter this by two measures: First, we keep the whole TemplateTree II setup in CVS which allows us to backtrack configuration problems we introduce ourself and, second, we protect the central TemplateTree II configuration server by only allowing access via secure channels. Further, the whole system gets backed up regularly and is trip-wired for easy detection of unauthorized modifications. We have also been thinking about using digital signatures on FeaturePacks but have not yet implemented such a functionality.

## Conclusion

By building on top of Cfengine a system has been devised that allows for the complete modularization of the post install process of Unix workstations. TemplateTree II allows us to perform fully customized machine setups in a very short time while maintaining full reproducibility. Because TemplateTree II works with a single top level configuration file the configuration information for the whole site is readily accessible and when changes are required, they can be quickly performed on all systems.

## Future Work

At the moment we are happy with TemplateTree II and use it as it stands. We do have some ideas, though. One would be to improve the Cfengine configuration generator to create more compact Cfengine configuration code. Currently the Cfengine configuration is about as voluminous as it can get. Smaller code would be simpler to understand and debug than the bulk we have today.

An entirely different road would be to replace Cfengine altogether and use a modified rsync server which provides a view on a virtual file system which changes its *contents* dynamically depending on the host which is sending the request. This approach would allow us to capitalize on the excellent performance and security possible with rsync/ssh.

## About the Author

Tobias Oetiker is a Senior System Manager with the IT Support Group of the Department of Information Technology and Electrical Engineering at the Swiss Federal Institute of Technology in Zurich. He is an electrical engineer by education and a system manager by vocation. His main area of interest is currently scalable system management concepts and their implementation. In his spare time Tobi likes to read, go to movies, and work on his Free (as in GNU) software projects.

## Acknowledgments

I would like to thank the following individuals for helping me with this paper: Paul Anderson, David Schweikert, Jon Stearley, and Fritz Zaucker. If you find anything especially well written or spelled properly, it must have been one of their suggestions.

## Availability

TemplateTree II is written in Perl and available under GNU GPL from http://isg.ee.ethz.ch/tools/ .

## References

[1] Steve Traugott, Joel Huddleston, "Bootstrapping an Infrastructure," http://www.infrastructures.org, *LISA*, 1998.

[2] Evard, Rémy, "An Analysis of UNIX System Configuration," *LISA*, 1997.

[3] Burgess, Mark, "Cfengine: A Site Configuration Engine," *USENIX Computing Systems,* Vol. 8, No. 3, http://www.iu.hioslo.no/cfengine, 1995.

[4] Lockard, John, Jason Larke, "Synctree for Single Point Installation, Upgrades, and OS Patches," *LISA 1998.*

[5] Burgess, Mark, "Computer Immunology," *LISA 1998.*

[6] Donald E. Knuth, *The CWEB System of Structured Documentation*, Addison-Wesley, 1993.

[7] Weisshaus, Melissa, et al., *GNU tar: An Archiver Tool*, http://www.gnu.org/manual/tar/ .

[8] Oetiker, Tobias, "SEPP – Software Sharing and Packaging System," http://www.sepp.ee.ethz.ch/, *LISA 1998.*

[9] Lamport, Leslie, "LaTeX: A Document Preparation System, User's Guide and Reference manual," Addison-Wesley.

[10] Ressmann, David & John Valdés, "Use of Cfengine for Automated, Multi-Platform Software and Patch Distribution," *LISA*, 2000.

[11] Goetsch, Victor, Albert Wuersch, Tobias Oetiker, *Gossips: The Systems and Services Monitor*, http://isg.ee.ethz.ch/tools .

[12] Wall, Larry, Tom Christiansen, "Perl POD: Plain Old Documentation," http://www.cpan.org/doc/manual/html/pod/perlpod.html .