A Distribution and System Compiler for Handling Heterogeneous Computer Site

Ivan Augé[1], Vincent Leligeour[1], Olivier Pons[2]

[1] ENSIIE, 18, allée Jean Rostand, 91000 Evry
(auge,leligeour)@ensiie.fr
[2] CEDRIC-CNAM, 292, rue Saint-Martin, 75003 Paris
pons@cnam.fr

Abstract. This document describes YaKa, a complete solution for deploying operating systems on large computer sites. This solution proposes a compiled approach that differs from the standard installation approaches which, first install a basic system and then update or integrate tools one by one by running scripts on the freshly installed system.
YaKa provides a language witch allows to describe a computer site as a whole and at all levels: host hardware, systems to install on the hosts, system description, links between host systems, software generation. A compiler then generates automatically the systems to be installed for each host and generates the network installation servers. The generated systems are ready to use and fully operational on their first run.
This approach has major advantages. As the compiler generates all the systems of all the hosts in a single operation, every component of a system is generated with a complete knowledge of all the other components of the system and of the network relationship between the host and its neighbours. This ensures the coherence of each piece of software with the other softwares of the system and with the environment of the host.
This dramatically decreases the duration of system installation and boot, by reducing the amount of components needing to be installed and the complexity of the startup scripts. It allows one to group together tools for providing abstract services such as the `network-user` service which links two or more tools among PAM, NFS, NIS, LDAP and SAMBA. On top of this, a compiled approach allows to do statically a lot of verifications such as checking if a required executable is present or that no dynamic library is missing on the system.
The power of the language is illustrated by its ability to describe a complete source based Linux distribution. The efficiency and reliability of the tool are demonstrated by its use for several years at ENSIIE (a postgraduate school in computer science) to manage all the hosts of the teaching network (10 servers, 100 clients with heterogeneous hardware, different operating system and software).

1 Introduction

During the past decades, the architecture of computer sites has evolved from a centralized to a distributed approach. Most sites are now composed of a large number of workstations often with several kinds of operating systems and different software configurations. Furthermore workstations tend to have heterogeneous hardware. Managing such a site and having all the computers in the wanted configuration is a challenge for the system administrator.

Although a bit primitive, manual installation using standard distribution procedure is still popular. This involves to log into each machine to copy or edit configuration file or to run custom scripts. It is very time consuming, source of error and makes it difficult to ensure the coherence of the whole site. One of the earliest solutions to automate the installation of a large number of machines is to use clones. This usually requires to manually configure some golden-copy machine and then to replicate its file system on each workstation. But this solution that guarantees identical configuration on each host and may be convenient for clusters is not practicable on heterogeneous sites. In both cases, updates that require to log into hosts to install and configure

new software are complex and time consuming. In practice this often dissuades system administrators to perform all but critical updates.

More recently, tools appeared to help the system administrator, first to install and configure workstations and then to manage them. Most of them are tightly integrated in free or commercial operating systems. For the installation, they use a file with the answers to the questions of the standard installation procedure. Then updates are achieved using the package manager of the operating system. Those softwares lack of genericity. If the site is not homogeneous in term of operating systems, system administrator needs to use several tools to perform the complete site installation. The situation is even worse if multi-boot workstations with different operating systems are required.

Furthermore, over time, the number of system and network softwares increased drastically and softwares became more and more complex. A few softwares offer new services, most of them are variant of already existing services. The system administrator has to choose which software to use for a given service and then to configure it. He spends more and more time to understand software and to configure them. This task becomes each day harder. Existing installation tools do not assist the system administrator in choosing which software to use and in configuring them.

This paper presents YaKa, a framework dedicated to the management of large computer sites heterogeneous both in software and hardware. It provides a language that permits to describe the full computer sites as a whole. Then compiling these description, it allows to generate **ready to use** operating systems and then to propagate them automatically on the complete site. the YaKa framework is freely available[4] and distributed under GNU Licence. The rest of the article is organized as follows. First we introduce the mechanism of site management and review some of the existing tools. Then we introduce our solution and describe our framework. Then we discuss the specificity of its associated language. And Finally, before concluding, we present and comment our experiments.

2 Managing a computer site

In the last decade, several tools have been offered for system management. [10,15,8,11,9,18] perform automated installation. They generally rely on a package manager such as RPM[5] (and associated tools `urpmi`, `yum`), dpkg[17](and `apt-get`), or in the Microsoft world `windows-update`[14]. Some tools[7,6,2,1] perform automated configuration. Some of these are image based, some others are package based. A few of them can support both approaches.

2.1 Package based approach

Most package based tools follow the typical scheme described Figure 1.

Database installation They use a DHCP/BOOTP/TFTP database for booting, a package database, a configuration database which contains configuration parameters and scripts for each host. Before anything else can happen, these databases must be up and populated. (1 Figure 1).

First boot A mini system is downloaded via the network using the DHCP, BOOTP and TFTP protocols (2a Figure 1). This mini system starts an installer which first detects network and hardware disks, and then gets installation parameters from the

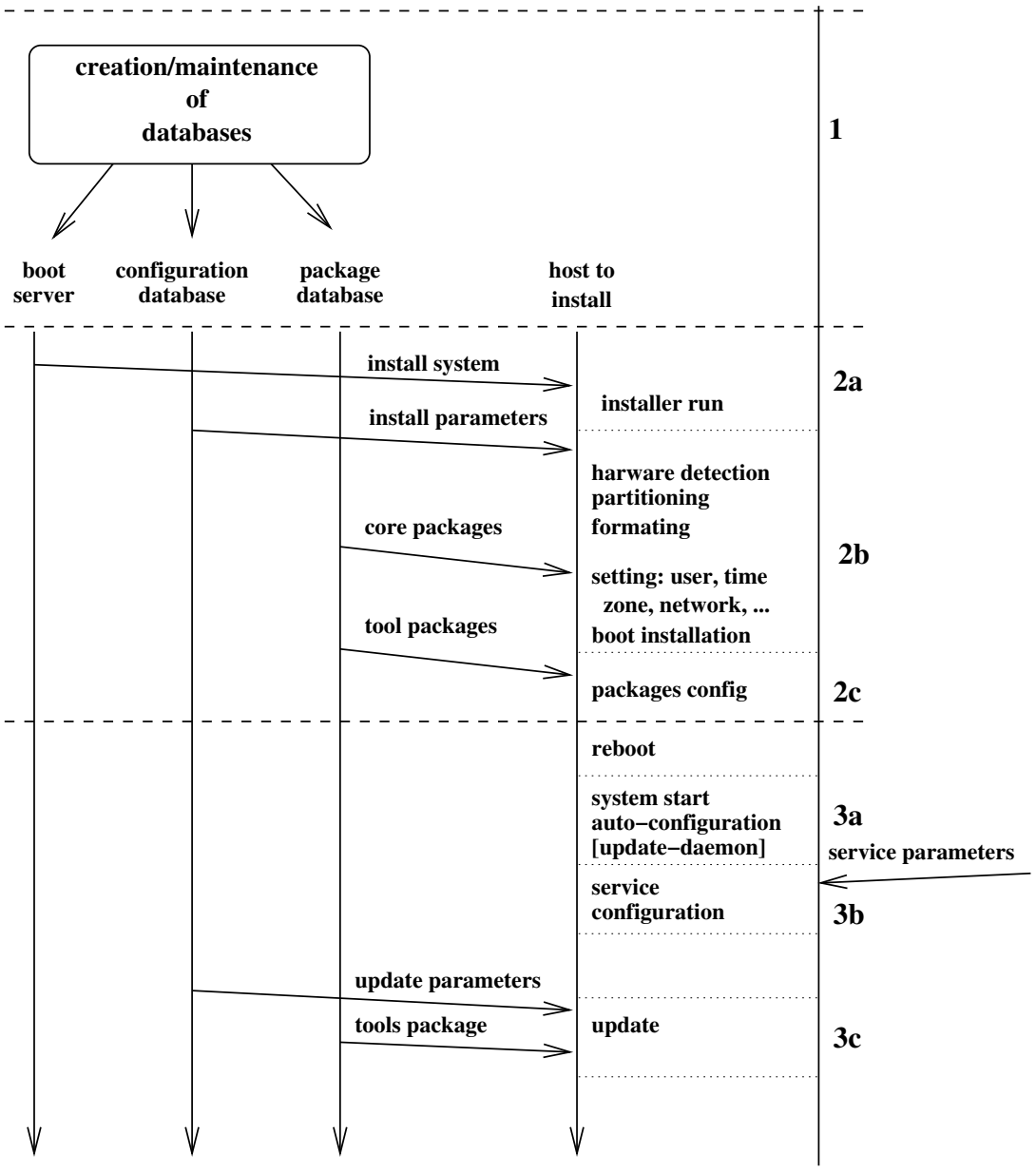


Fig. 1. Functional scheme of package based deployment tools

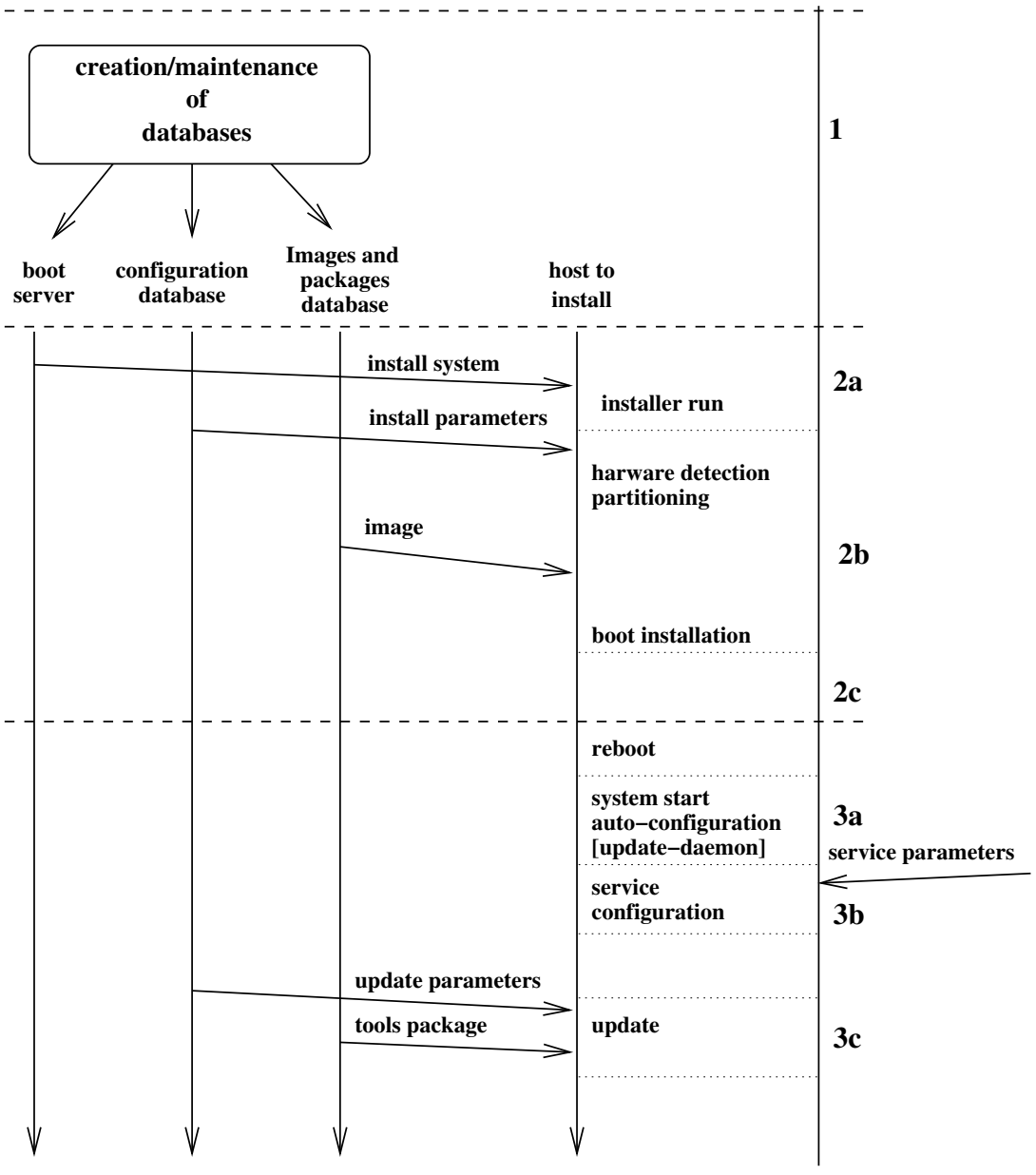


Fig. 2. Functional scheme of image based deployment tools

configuration database. They describe among other things the disk partitions, the root password and the packages to install. Using these parameters (2b Figure 1) the mini system performs several tasks. First, it builds the disk partitions and formats. Then it downloads the *core packages* (the packages needed to create a base system) from the package database and install them. Finally it configures the system (timezone, network, language, mount point, disk boot, ...).

After these steps, host specific packages, called *tools packages* (2c Figure 1) are installed. Unlike the core packages, they are not integrated into the installer. These packages have a default configuration which, in most case, does not match the requirements, so the actual configuration is delayed. For instance, the *rwall* tools package may just provide binaries, which is not really useful, or may be configured by default to allow every user to send messages to any machines, which is fine in a home LAN but not in a computer site. Actually, in a computer site, only servers should be able to send message to client boxes.

Next boots At every reboot, the system auto-configures itself (3a Figure 1).

Once the system is started, the system administrator must configure **once** (usually at the second reboot) the services that the previous step has left unconfigured or incorrectly configured (3b Figure 1). Configuration is performed by editing files, downloading and running scripts or downloading and unpacking archives. To do that, the system administrator logs on the machine either manually or by running remote scripts.

At this point the machine is operational. Periodically existing tools must be upgraded or new tools must be installed (3c Figure 1). In both cases the configuration and package database must be updated, then the system administrator or a daemon start the package manager. In the case of the installation of a new package, the corresponding services may be configured as previously (3b). In the case of an upgrade, the existing configuration is usually kept. Note that this operation may break the software if the new release is not compatible with some options of the previous configuration.

2.2 Image based approach

Image based tools presented Figure 2 follow more or less the same scheme as the package based tools. Before starting the installation, system administrators have to build the image and to store it in the package/image database (1 Figure 2). A mini-system is downloaded. It starts an installer that initializes the disks, downloads the image and then installs the boot (2 Figure 2). An image depends on the hardware and network configurations amongst other things. In the worst case, a different image is needed for each host. To avoid this, a standard image could be built and adapted after its installation on the host (3b Figure 2) This operation can be performed either manually or by using tools such as [13,7,2]. The machine is now configured; its update (3c Figure 2) is similar to the package based installation.

The main drawbacks of this method are that system administrators spend a lot of time handling images, and that downloading images is much slower than installing distribution packages. This method is generally used for operating systems which does not support package installation or for bypassing restrictions of installer tools.

2.3 Existing solutions

Existing softwares differ in database formats, access protocol and packaging tool but these differences are not really relevant for our purpose and are not discussed in this paper. The significant differences are in the support of different steps, in the way to build the configuration databases process and in the operating systems that can be installed. We shortly discuss some popular tools without pretending to be exhaustive.
Fai [10] supports steps 1, 2, and impolitely 3c. The package database is a mirror of the Debian package database and the configuration database must be built manually. The step 3b may be suppressed by writing scripts called "hooks", but they should be written manually by system administrator. This system is dedicated to install Debian operating system, but by using hooks, it is possible to install images of other operating systems. Kickstart [15,16] and jumpstart [18] are similar tools but respectively for Redhat Linux and Solaris.
Lcfg [2] supports the steps 1, 2, 3c. The package database is a mirror of a rpm base. The configuration database is generated from description files allowing system administrators to define components and host profiles. A component is a set of variables with default values and scripts. It corresponds more or less to a package or to a simple feature like a file as `/etc/export`. A host profile is a set of components where default values can be overwritten. For specific configuration, the step 3b may be suppressed by rewriting the component scripts. The generation of the configuration database makes it safer than the previously mentioned softwares. Nevertheless there are significant weaknesses in this framework:

- A configuration script has neither knowledge of the other components installed on the operating system, nor a global view of the computer site except for a few built-in parameters like for instance MAC addresses of the others hosts.
- Running configuration scripts on the host slows down the installation.
- Bad configurations, such as misspelling a configuration variable name or a command path in a configuration script, or a missing dynamic library are only detected upon installation or upgrade.

Cfengine [7] proposes another approach. It only supports step 3b and have to be used with an installer tool [12]. The configuration database is generated from description files. A configuration may be view as a set of couples (condition, action). On the host a daemon periodically reads the configuration and executes the actions corresponding to the unmatched conditions. This allows to update existing tools, to install new tools and even to monitor the host. The description language allows to define abstract notions such as a NFS client and to map them on different operating systems. Nevertheless configuration descriptions can not easily refer to other components installed in the operating system and even less to the other hosts of the computer site.
M23 [8] offers a web interface to manage the database installation of Debian packages[17]. It supports the steps 1, 2, and 3c. But it does not offer abstract services and as a result step 3b is left to the charge of system administrators. If an administrator expects user's accounts to be handled by NFS/SAMBA/NIS, he will have to read the documentations of these tools and to configure them manually. If he is not an expert of these tools, he will spend several weeks to find a configuration corresponding to his needs. If he wants to switch from NIS to LDAP, all the configurations needs to be redone.

We identified the main weaknesses in existing tools for managing computer sites:

– Except for [7] they are tool-oriented and do not provide any abstract services such as Network users, mail hub, system message broadcasting.
– They intensively use system updates. On the one hand this raises reliability and security issues. On the other hand updating is performed by package manager which installs much more packages that what is really needed and fails to automatically configure complex softwares. Reliability problems are mainly due to misconfiguration and to the lack or incompatibility of dynamic libraries so the stability of operating system decreases as the time grow. Security update are mandatory but they must be done before operating system attacks. Package manager problems are due to package granularity. The finer the grain, the more dependencies decrease but also more difficult is the package maintenance.
– In order to support various hardwares and softwares, they use a lot of large and complicated scripts parametrized by a lot of variables. This may be adapted for home systems as this allows the system to provide a graphical user interface for system configuration, but this notably slows down the system initialisation process.
– They often require system administrator intervention.

3 The framework

3.1 Overview

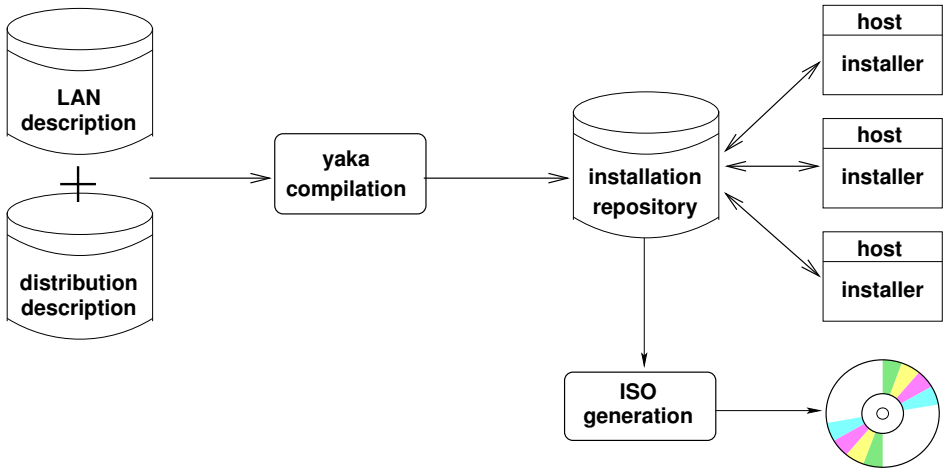


Fig. 3. General flow graph of YaKa framework

The YaKa framework provides a language, a compiler for this language and an installer. The language allows one to fully describe a computer site, as well as a full Linux distribution (see Section 5) or a small system suitable for an installation ramdisk. The general flow graph of the framework is presented Figure 3. To start with YaKa, one must describe the computer site using the language. Then the compiler generates automatically the *installation repository*. The later contains the boot server (DHCP/TFTP) and the download-unit database. A download-unit is an archive file. There are standard and configuration download-units. The standard download-units contain software files that do not depend on the system and host in contrary to the configuration download-units files. To install a host, one simply asks it to boot from the network, the installer

is downloaded and installs the system fetching only the needed download-units into the *installation repository*. For hosts unable to perform a network boot, the YaKa framework provides a command to generate ISO images from the *installation repository*. This approach results in the functional scheme presented Figure 4.

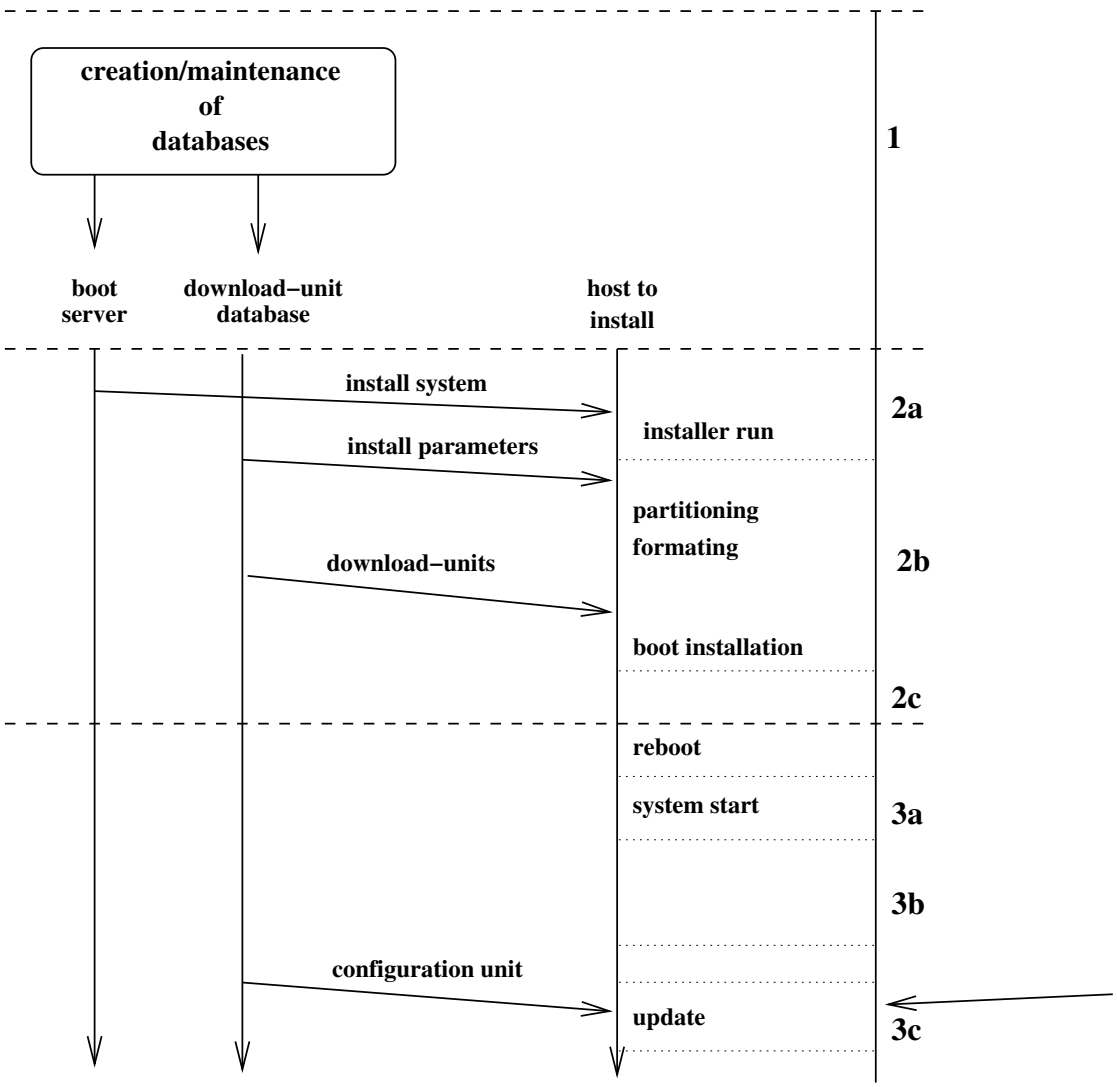


Fig. 4. Functional scheme of YaKa tools

Database installation In the YaKa framework the configuration and package databases are replaced by a download-unit database (1 Figure 4). It is included in the *installation repository* and so it is fully automatically generated. In contrary to others installation tools, the configuration files are **ready to use**, there is no parameter but directly the files the system requires. For instance the "rc.sysinit" file, downloaded within a download-unit, contains the line "`/sbin/loadkeys fr-latin1`" if the host has a french keyboard or "`/sbin/loadkeys us`" if the host has a US keyboard. This is to compare with the "rc.sysinit" file of the Mandriva Linux distributionwhich uses several tens of lines for the same result.

First boot The installer of the mini system does the disk partitions, it formats them if required and then it downloads the download-units. There are neither additional hardware detection, nor package configuration. This is due to the **ready to use** nature of the configuration files.

Next boots At the second boot the system is fully operational. There is no need for the system administrator to operate as all services are already configured (3b Figure 4). The only exceptions are for the few services using data that the system administrator does not want to leave on the *installation repository* for security reasons, such as the LDAP administrator password. Notice that the global description allows to knows all services running on all machine without needing service discovering tools such as [19].

The update step (3c Figure 1) does not exist, it is replaced by a full reinstall. Of course this leads to a system shutdown but the unavailability time is low enough to be supported. For instance on a 100 Mbits network, a reinstallation, from shutdown to the login prompt, takes less than 2 minutes for a server and about 5 minutes for a 3.3 GB Linux system.

For servers that should not be halted, YaKa provides configuration units that can be downloaded and installed by the system administrator on running systems. For instance, this permits to add/suppress users or to add/suppress entries in a firewall easily.

YaKa also supports image installation and multiple operating systems on the same host. The installer can be configured to install/reinstall one or several systems on a host. For a system using several partitions it can be configured to install/reinstall one or several partitions. This last feature is especially useful when reinstalling a system to preserve the already existing user's home, databases, ...

Unlike other tools, YaKa does not perform updates on a running machine, and does not execute configuration script when a machine is installed. All this is replaced by a reinstallation. This is possible because both the reinstallation and the reboot are very fast. The reinstallation is fast because it only initializes the disks, fetches the download-units and installs them without any transformation. The reboot is fast because the system start scripts only contain the necessary code and they only start the required services.

Regarding reliability and security, a full reinstallation guarantees that the system restarts in a clean state, that malicious software installations have been cleaned up and that unknown processes have been killed. Finally, installation/reinstallation being fully automated, system administrators can leave this task to the end-users on the client desktops.

3.2 Compilation phases

The compilation flow graph is presented Figure 5. The *description database* is a set of *Y-Packages* described in YaKa language. A *Y-Package* is the basic element for building operating systems, it corresponds to one or several softwares. They are compiled with `yaka-comp` to generate the *Y-Package binary database*.

Computer site description (*LAN description* in Figure 5) also uses the YaKa language, to describe the hosts of the computer site. For every host, the hardware, network parameters and desired systems are specified. These systems are either disk images or are built from the *Y-Package*. Furthermore at the end of the installation process, user archives

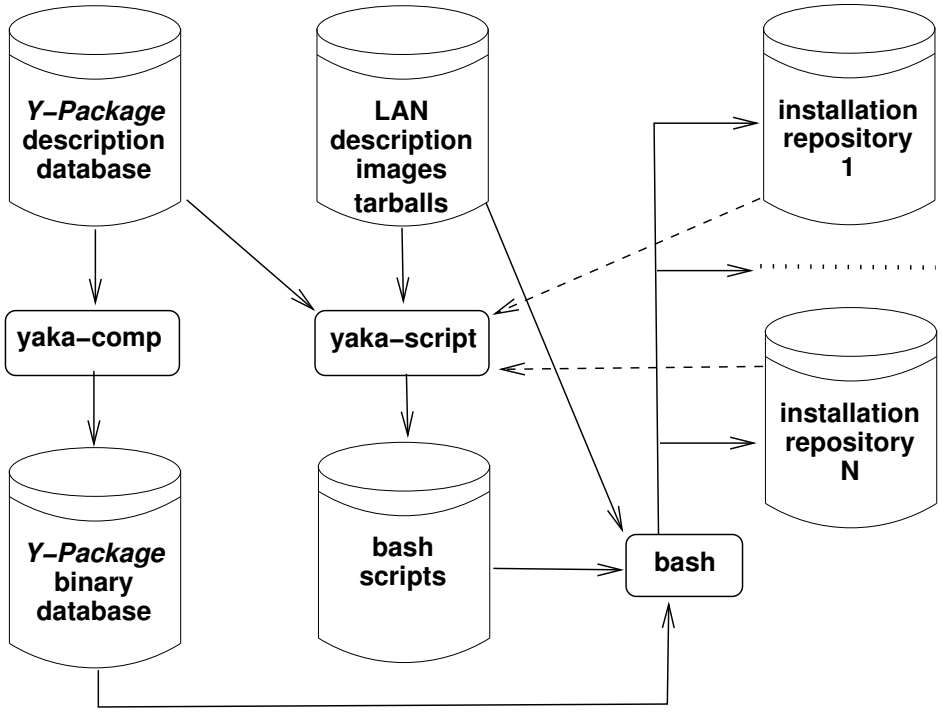


Fig. 5. Flow graph of YaKa compilers

can be added. This is useful to restore data such as user's home or DBMS data. The `yaka-scripts` command reads the computer site description, the *Y-Packages* which are referenced and then creates *bash scripts*. Running these scripts generates the *installation repositories*. A *installation repository* is a directory containing the DHCP, TFTP, FTP servers with their configuration files and a script to start them. It also contains a fully configured PXELINUX and all the download-units. For efficiency reasons, `yaka-scripts` first looks in the *installation repositories* (doted arrows Figure 5) and creates only the bash scripts for the missing download-units.
Finally, LAN description supports several *installation repositories*. This allows to switch from a standard installation environment to a specific one when needed. For instance, a standard installation environment installs Microsoft Windows and Linux operating systems with a multi-boot and network accounts, a specific installation environment installs a operating system for handling the hosts as a computing grid.

3.3 Installation scheme

Running an *installation repository* starts the DHCP, TFTP and FTP servers. Figure 6 illustrates exchanges between these servers and a machine being installed.

Network boot At network boot, the bios broadcasts a DHCP/BOOTP message, the DHCP server answers, giving an IP address and a boot program. We currently use PXELINUX[3].

PXELINUX boot This program requests a configuration file from TFTP servers. This file describes installation options and presents a boot menu to the user. The user selects what he wants to install. Then the program downloads via TFTP a Linux kernel and a minimal Linux system (8 Mb) containing the installer within a ramdisk. There is a single ramdisk in an *installation repository*. PXELINUX ends

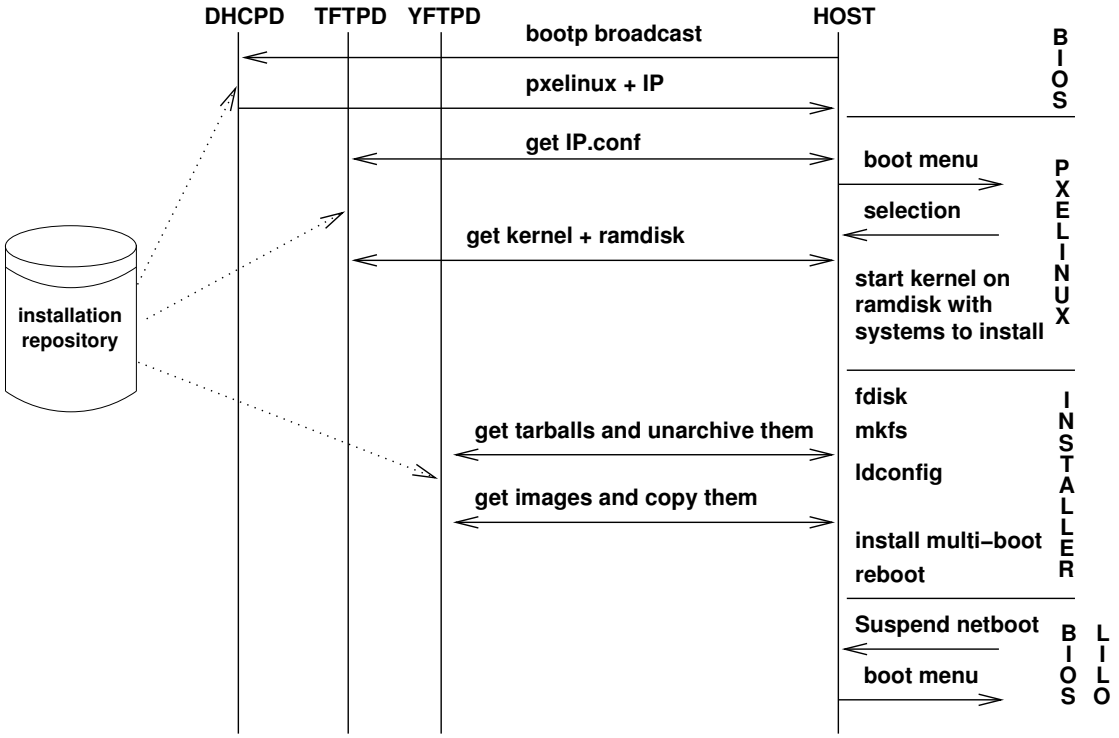


Fig. 6. Installation scheme

by starting the Linux kernel, giving it the user selection via the kernel parameters. After its initialisation, the Linux kernel launches directly the installer instead of the standard `/etc/init` program.

Installer Using the kernel parameters and the MAC address of the network interface, the installer finds in a data file present on the ramdisk the sequence of tasks to complete the system installation. The only tasks it can perform are making disk partitions, formatting partitions, downloading and unarchiving tarballs, downloading gziped files and copying to a disk partition, creating the dynamic library cache and installing a multi-boot loader. Then the system reboots and is fully installed and configured.

Multi-boot The user chooses among the systems that are already installed.

The important point is that once the description of the computer site in the YaKa language is created, the process requires no human intervention except for the network boot and multi-boot menus. These two steps have a timeout that starts the system tagged as default in the YaKa computer site description.

4 The language

The YaKa language embeds a preprocessor which transforms the program before compilation. It adds support to suppress comments, for stringification, to include file and to select whether to include a chunk of code or not.

4.1 Language summary

The main syntax elements of the language are:

Y-Variable They are string variables or sets of string variables, some are predefined such as `ip4` which defines the IP address of the first network interface of a given host. They can be set and read. Others read primitives are provided, allowing to delay variables evaluation until they reach the context of a system on a specific host.

Y-Host It is defined as a set of *Y-Variables*, the variables defining the IP and MAC addresses are mandatory. Hosts may be grouped to be handled more easily.

Y-Package It contains: 1) A set of *Y-Variables*: theses can be the configuration parameters of the package, such as the address of the proxy for an http browser or generation parameters such as the version number. The configuration variables can have default values which can be overloaded by redefining them in the host or in packages including this package. 2) A set of files, making up the package. They can be simple files (binary or data) or bash scripts generating host specific files. 3) A set of sub-packages, those indicate the package dependencies. 4) One generation/compilation script. Its function is to produce the files required in the set of files. The only restriction in the script is the bash syntax. For instance, for getting the binaries of a given software, one can choose to compile the official source archive or to run the official binary installer or even to extract them from the package of an other distribution such as Debian or Redhat

Y-Disk It is a set of partitions, a set of mounts and exports of file systems.

Y-Download It is a couple (*Y-Disk*,*Y-Package*) or (*Y-Disk*,(partition,image)). In the first format, the files of the *Y-Package* are installed in the partitions defined in the *Y-Disk* taking into account the file system mounts defined in *Y-Disk*. In the second format, the data of the image are written into the partition which must be defined in *Y-Disk*.

Y-Netboot It is a set of couples (*Y-Host*, set of *Y-Download*). It corresponds to an *installation repository*. The DHCP/PXELINUX server will propose a menu to all the *Y-Hosts*, the items of the menu being the *Y-Downloads*. Then the installer installs the *Y-Download* corresponding to the item the user selected.

A computer site is fully characterized by a *Y-Netboot*, a *Y-Netboot* defines the *Y-Hosts* with their associated *Y-Downloads*, the *Y-Downloads* defines the *Y-Package* (the files of system) and the *Y-Disk* which maps the *Y-Packages* files to the hardware disks. The *Y-Variables* are the generic parameters of the *Y-Packages*, they allow to manage host specificity.

4.2 Language specificity

The files defined in a *Y-Package* can be simply copied from an existing file or can be generated in different ways using the system and host context. An example of system context dependency is the PAM configuration. The PAM/login configuration file depends on the identification tool used (standard unix, NIS or LDAP). An example of host context dependency is the configuration of an Internet client, because it depends on whether the computer is directly connected to the Internet or not.

Figure 7.a illustrates how the configuration file of a ftp client named `/etc/ftp.conf` can be generated. The `hs-file` token indicates that the data enclosed by the `RAWDATA` markers are the content of the file and macros like `YKV` are expanded in the system and host context. Here the `YKV` macro expands the HTTP-PROXY variable to its value or to `none` if the variable is undefined.

```
F /etc/ftp.conf hs-file <<RAWDATA
  ...
  proxy=YKV(HTTP-PROXY none)
  ...
RAWDATA
```

a) using basic filtering

```
F /etc/issue hs-bash <<RAWDATA
  name=standard
  if yakaget -pkg funny ;
     then name=funny ; fi
  echo "system $name "
  echo "generated the $(date)"
  echo "by $(hostname)"
RAWDATA
```

b) using bash script

```
F /bin/dummy hs-bash <<RAWDATA
  cc -DDUMMY=YKV(DUMMY-VAR) dummy.c
  cat a.out
  rm a.out
RAWDATA
```

c) using bash script

Fig. 7. Configuration file generation

Figure 7.b illustrates how the `/etc/issue` file can be generated using a bash script. The `hs-bash` token indicates that the data enclosed by the `RAWDATA` markers is a bash script. This script is run in the system and host context. The data the script writes on the standard output stream become the content of the `/etc/issue` file. In such a script, the `yakaget` command allows to fetch information from the system and host context. In the current example, the "`yakaget -pkg funny`" command checks if the system being built contains the `funny` *Y-Package*, `date` and `hostname` are the usual Unix commands.
In the last example (Figure 7.c), we create in the target system, the file `/bin/dummy` which contains the executable generated by compiling the `dummy.c` file. In this compilation the `DUMMY` cpp macro can be set to different values in every host by setting the `DUMMY-VAR` variable. So every host can have a different binary.
These examples show the power of the YaKa script features: 1) It's an easy way to adapt files to a given system of a given host in function of the information available in the system and host context. The main information kinds are: checking if a variable is defined, getting the value of a variable, checking if the system contains a given file, checking if the system being built contains a given *Y-Package*, getting the path of all directories of a given base name, getting all the hosts of a given network address. 2) Since we do not run script on installed host, we have not to install the tools used by the script on the host system. In the opposite to other tools, in the last example a C compiler and the file `dummy.c` have not to be present on the target system.

In a package oriented tools there are several dependency types, both in package generation and in package installation.

Generation dependency Package A has a generation dependency on the package B when to generate A, one must first generate package B. This does not imply the need of B to use A.

Hard dependency Package A has a hard dependency on package B when package A can not run without package B. The usual reasons are that A requires executables or dynamic libraries, possibly with configuration files, from B. The amount of the required B is very variable.

Soft dependency Package A has a soft dependency on the package B when package A requires for running one or several B dynamic libraries linked within the A package binaries (not loaded explicitly through the `dlopen` function).

Soft cross dependency Package A has a soft cross dependency on package B when the A package modifies the B package by adding one or several files. For instance, installing caml adds language syntax files to emacs. We call this soft because installing emacs with the caml language syntax files on a system which does not contain caml is not a problem and may even be useful.

Hard cross dependency Package A has a hard cross dependency on the package B when the A package adds files to the B package or modifies already existing files of B. For instance installing a http-broswer adds an entry in a window manager menu. We call this hard because installing the window manager with the menu entry to the broswer on a system which does not contain the browser will result in a dead link.

For generation dependencies, the language provides the "`<read> package-name`" primitive. For the hard dependency, the language provides the "`use package-name`" primitive which adds all the file of the *name Y-Package*. Furthermore YaKa offers the possibility to add only some files of a *Y-Package* to another *Y-Package*. For instance, a client system using Samba for handling user accounts, does not need the full Samba software but only the `smb.conf` file and the `smbpasswd` command. These files can be picked from the Samba package. Such a precise selection is not possible with most of the existing installers because the inside of package is not available whereas with YaKa all the information of all *Y-Package* resides in the compiler memory.

For the soft dependency, when generating a system, the compiler searches for the missing dynamic libraries. It searches them in the existing *Y-Packages* and adds automatically the missing libraries to the generated system and notifies the user about those it has not found. This allows to suppress a lot of hard dependency that other tools have. It is also especially useful when taking a piece of a package. In the former example (user's account handling with Samba), one does not have to worry about the dynamic libraries needed by the `smbpasswd` command.

For the soft cross dependency YaKa works as the others tools, the files of a package dedicated to an other package are installed even if the other package is not present on the system.

For the hard cross dependency, the built-in `yakaget` command allows a *Y-Package* to take in account the existence of another *Y-Package* or of a file in the generated system. Furthermore the set of variables allows to distribute among the *Y-Packages* a given information. For instance in the *Y-Package* `firefox` we have the primitive "`VLV MainMenu="firefox@firefox.png@/usr/bin/firefox"`" which define a window manager menu entry, labeled firefox, with icon `firefox.png`, with binary `/usr/bin/firefox`. In the *Y-Package* `acroread` we have similarly "`VLV MainMenu="accroread@accroread.xpm@/bin/accroread"`". In a system containing the `firefox` and the `accroread` *Y-Packages* the `MainMenu` variable will contains the two values, in a system where `accroread` is not present the corresponding entry will not be present in the `MainMenu` variable. So, all window managers can use

the `MainMenu` set of variables to create a main menu adapted to the generated system.

When describing a package or when configuring a computer site a lot of errors are possible, the YaKa language and the compiler provide several features to avoid them.

mistakes in variable handling The package designer defines variables for configuring the packages that the system administrator must set. Classic mistakes are 1) the system administrator uses a configuration variable for his own need and so overwrites its default value, 2) he wants to set it but misspells its name, 3) he forget to initialize a mandatory variable.
The language defines and types the configuration variables of a package as in "`define VLV MainMenu`" which types MainMenu as a set. The other types are `CFV` for configuration variables and `ENV` for environment variables. When such a variable is defined, the variable name becomes case insensitive and variables must be assigned using the type token (eg: `VLV MainMenu = ... ;`) avoiding mistake (1) and assigning a variable which has not been defined as a configuration variable with a type token (eg: `VLV MainMenu = ... ;`) generates an error avoiding mistake (2). To avoid the mistake (3), the "`yakaget -vov varname`" command allows to retrieve the value of the `varname` variable in the system and host context. By default it generates a fatal error if the variable is undefined in the given context.

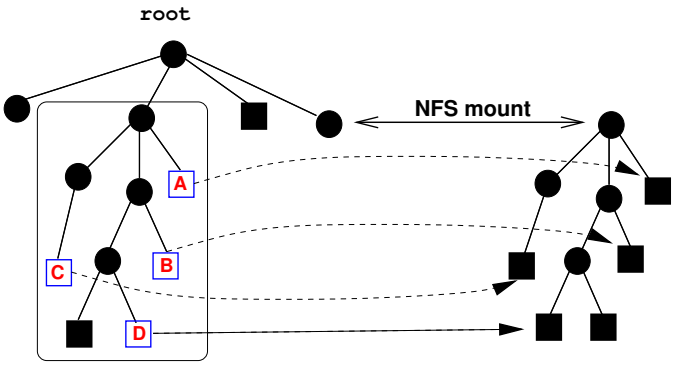
misspelling file name The package designer often has to write scripts. Those reference files, the most common errors being misspelling the file or referencing an non-existent file or for binaries a bad executable search path. The `YKG(str)` macro detects these errors at compilation time. This macro searches for file paths ended by `str` and expands to the absolute path of the file. The expansion is done in system and host context. If the built system does not contain such a file, a fatal error is raised, if several files match the shortest path is selected and an error message is printed. So the line starting the ftp server may be written in the system startup script "`YKG(n/in.proftpd) -c YKG(proftpd.conf)`" which will generally expand to "`/usr/sbin/in.proftpd -c /etc/proftpd.conf`"
This macro can also be used in variable setting, the previous setting of the `MainMenu` variable may be written to enforce reliability:
"`VLV MainMenu="firefox@YKG(cons/firefox.png)@YKG(in/firefox)"`"

forgetting dynamic library When building a system, the compiler indicates the missing dynamic libraries. Options allow to print the executables or the dynamic libraries requiring them.

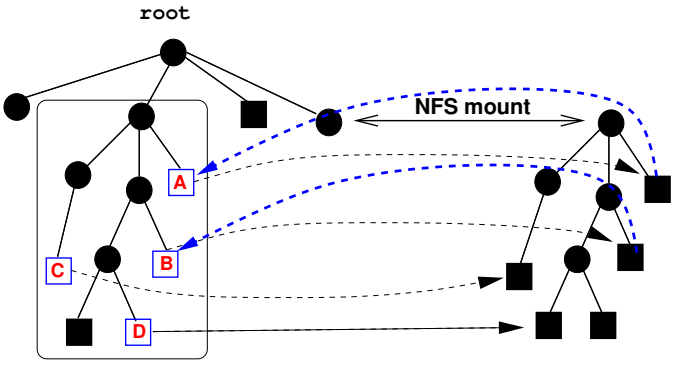
multiple definition of file Sometimes different packages create different files under the same path. The installation order of download-units will result in different system. The compiler automatically detects such inconsistencies and generates an error.

To speed up the installation, it is possible to delay the download of tools at the first use using a feature we call the YaKa file system [3]. It consists in mounting via the network an image of the real file system and to replace some local regular files by symbolic links to the files of the network file system. Figure 8.a shows the two file systems, circles being directories, black square being ordinary files and white square being symbolic links to the network file system.
From the user point of view, YaKa file system is completely similar to a local file system

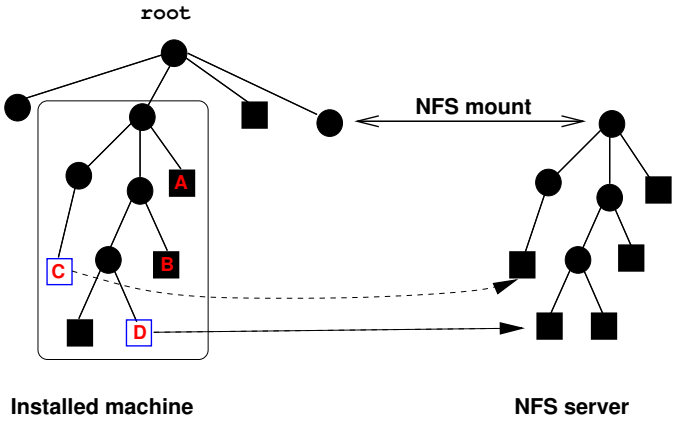
[3] We call it file system even if it is not one but we have not found a right nomenclature.



(a) Links between file systems before using A



(b) Requiring A and B



(c) Links between file systems after using A and B

Fig. 8. Principle of YaKa file system

except a time overhead due to the network. But it differs from a classic network file system: the first time an user runs an application (concretely he accesses a set of files, *A* and *B* Figure 8.b), the system gets them via the network, then a daemon scanning for accessed symbolic links replaces them by the real file (Figure 8.c), so the subsequent runs of the application are done locally. Notice that the YaKa file system does not require any kernel patch.
The language integrates primitives for handling the YaKa file system. Firstly in the *Y-Package* syntax element, the package developer has to specify files that can support the YaKa file system. Secondly the system administrator must enable the YaKa file system in a *Y-Download* element (presented above) with the next primitives:

```
yfs-dir: <dir0> <dir1> ... ;
yfs-serv:   <Download-S> <Host-S> <path-S> ;
```

Let's assume that `Download-C` is the *Y-Download* having these primitives. All the hosts loading `Download-C` will have YaKa file system support for the directories $\texttt{dir}_i$, this means that the files of these directories will be replaced by symbolic links if the package developer has allowed it. The symbolic link destination, the NFS mount of `Host-S:path-S` directory and the YaKa file system daemon start are automatically set. The second primitive indicates that the system `Download-S` downloaded by the host `Host-S` will have the file system image of the `Download-C` in the `path-S` directory. The NFS server is started automatically. This automatic generation grants the coherence of links, mount and export of the network file system.

5 Results

The language is powerful enough to describe a full source based distribution. The distribution contains several hundreds of *Y-Packages*. The leaf *Y-Package* contains: base system utilities (PAM, LDAP, NIS, SAMBA, SSH, CRON, APPACHE, MYSQL, CVS, FTP, LPR, FOOMATIC, FIREWALL, VIM, EMACS, ...), 3 X11 supports (XORG standard, XORG ATI, XORG NVIDIA), 8 window-managers and KDE, languages (C, C++, Java, Caml, Python, Perl, ...), classical web and multimedia tools (THUNDERBIRD, FIREFOX, OPERA, MPLAYER, XMMS, XCHAT, OPENOFFICE, TEX, ...), various other tools (AUTOMAKE/CONF, MAKE, FLEX/BISON, GNUPLOT, SCILAB, ...), games.
The distribution also integrates higher level *Y-Packages*. They offer services as basic monitoring, disk rescue, disk synchronisation, auto remount of network disks, and multi-user screens (independent keyboard/mouse/screen on the same machine).
The distribution defines also top *Y-Packages* which are base to describe target systems. The most significant ones are the *pkg-std Y-Package* which regroups most of the *Y-Packages* in their development version, the *pkg-gpt* which is dedicated to general purpose teaching, the *pkg-serv* which is very small and dedicated to servers, the *pkg-office* which is dedicated to office work and is configured to look like Microsoft Windows.
The distribution has been designed to be "service oriented". For instance, to handle mails we have the following *Y-Packages*: a *Y-Package* mailhub-frontend which receives and filters the mails from the external world, a *Y-Package* mailhub-delivery which delivers the mails into the user's mailbox and a mailhub-out which receives the incoming mail from the computer site to the external world. In a system including these 3 *Y-Packages*, a single smtp server will perform the 3 services. If a system includes only

the first service and 3 other systems include only the delivery service, you get a single smtp server which filters quickly the incoming mails and dispatch them to the others delivery servers.
All services configure themselves taking into account the variables and the host and system context. For instance, to switch from a computer site based on NIS/NFS for Unix and SAMBA for Microsoft Windows (with synchronized NIS and SAMBA databases) to one based on LDAP, there are just to change 5 variables.

YaKa is a concise language, it allow to describe in about 150 lines a computer site containing: 1 machine with a master LDAP server and a SAMBA account server, 1 machine with a slave LDAP server and a NFS and SAMBA home server and 2 clients having a multiboot (unix-yaka and a windows image).
The language can also describe large computer site as the teaching LAN of the ENSIIE. The class rooms have more than 100 desktops with heterogeneous hardwares and supporting several operating systems:

1. A YaKa/Linux based on the *pkg-gpt* top *Y-Package* for everyday use.
2. A YaKa/Linux similar to the former but with network restriction to avoid plagiarism during laboratory classes.
3. Several Microsoft Windows.
4. Automatic examination systems, one for programming test, one for middleware test and one for lex and yacc test.
5. Practice versions of the former systems to allows student to prepare for the exams.
6. Several systems dedicated to system, network, LAN administration courses. They allow students to be root, they may be uninitialized for LAN administration practicals, they may have special Linux kernels and specific tools like iproute2 for network teaching.

The standard configuration of theses machines is a multi-boot with 4 entries, one entry is shared by operating systems that are periodically used. Everybody can install or reinstall one system among 8. For handling the clients and other purposes there are two YaKa servers, three NFS and SAMBA home servers with a firewall at MAC address level, two DNS, synchronized NIS and SAMBA account servers, an http proxy server, two rescue server, an administration mail server, a project submission server, a http, cvs, anonymous ftp, bug tracker servers. All these servers run also a YaKa distribution and are handled by YaKa.

The ENSIIE computer site description takes about 3000 lines including a small on-line user documentation for 144 hosts, 1449 UNIX systems, 78 WINDOWS systems and 8 *installation repositories*.
On a **P4-3GHz**computer, the full generation of the distribution (*yaka-comp* Figure 5) requires 1 day, the standard generation using some pre-compiled *Y-Packages* (bootstrap, OpenOffice, KDE, ...) needs 6 hours. This generation is only required once.
Once the distribution is built, the first generation of the *installation repositories* takes 43 minutes decomposed in 8 minutes for the compilation (*yaka-script* Figure 5) and 35 minutes for generating the download units (*bash* Figure 5). Subsequent light updates such as adding or suppressing an host, changing a host configuration spend 6.25 minutes (3.25 minutes for compilation and 3 minutes for the *installation repositories*). Subsequent updates such as adding a new software requires 11.75 minutes (3.25 minutes for compilation and 8.5 minutes for the *installation repositories*) plus the time to generate the software (*yaka-comp* Figure 5).

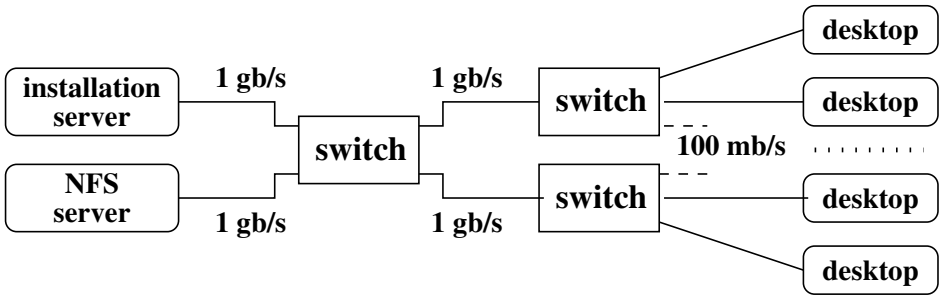
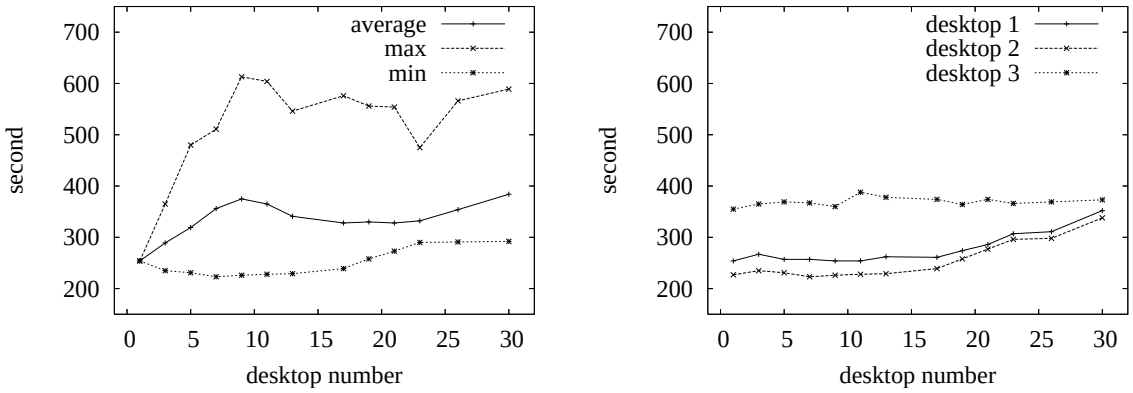
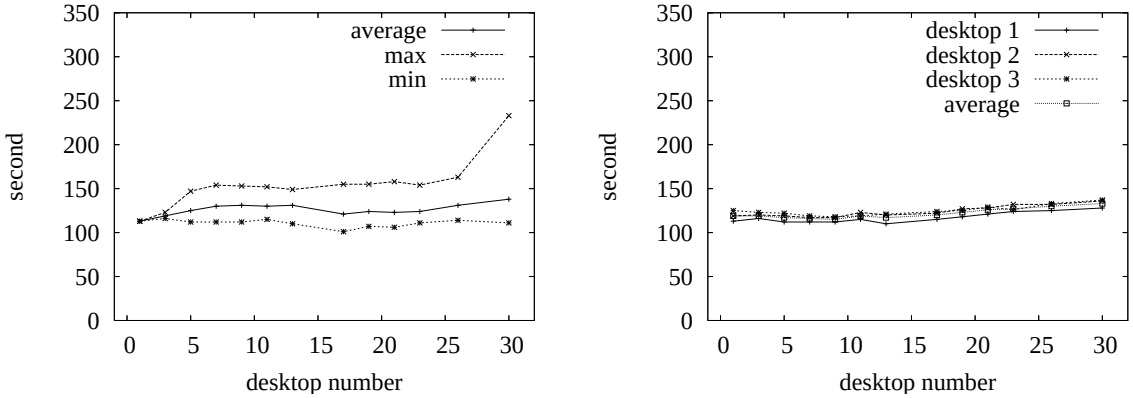


Fig. 9. Network used for experiments



(a) for N host (b) for a given host among N

Fig. 10. Duration of an installation



(a) for N host (b) for a given host among N

Fig. 11. Duration of an installation with YaKa file system

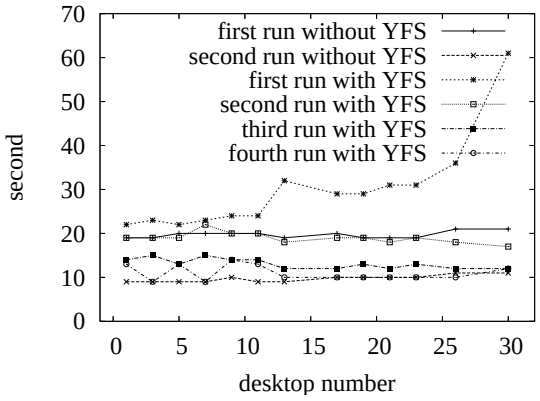


Fig. 12. Time overhead of the YaKa file system

We tested the download speed on the LAN presented Figure 9. In all these tests we have used a Linux system of 1.5 gb containing among other thing X11, ICEWM, FIREFOX and OPENOFFICE, and each test was started simultaneously on all the desktops.

Figure 10.a gives the time required to install N hosts. The measured duration is composed of shutdown time plus the network boot plus the installation time plus its shutdown plus the second network boot and plus the installed system startup until the login prompts on the Linux consoles. The curves show that the installation duration varies in ratio from 1 to 2.5. This is essentially due to bad network connectivity and disk weaknesses. The variations of the left part of the graphs are not significant. The average and minimal curves show that the network starts to slow the installation at about 16 hosts, this is confirmed by Figure 10.b which gives the duration for installing the same host depending of the number of hosts that are installed simultaneously. It is more than the expected 10 given by the bitrate ratio of the switch line bitrates ($1000/100\ mb/s$). In fact the deskstops alternate downloading and writing on the disk and they do not use all the 100 mb/s available. So 20 desktop are installed in about 5 minutes. To keep this performance with more hosts, it is possible to run several *installation repositories* to distribute the load or use the YaKa file system.

Figure 11.a gives the duration using YaKa file system for installing N hosts and Figure 11.b gives the duration for installing the same host. The average curve starts near 25 hosts to grow up slowly. A single *installation repository* installs 30 desktops in about 2 minutes. It is three times more than the expected 10. That is essentially due to about 50% of time is now spent in bios and system start.

The last experiment is to start simultaneously on all the hosts X11, ICEWM, FIREFOX (getting a page) and OPENOFFICE (opening a document) and to time when FIREFOX and OPENOFFICE are fully started (page and document are on the screen). The results are presented Figure 12, they show that YaKa file system becomes quickly similar to the local file system. The overhead of 40 seconds for 30 hosts that will probably grows exponentially is not critical. Indeed it assumes that 30 humans starts a YaKa file system application in less than 15 seconds, this is not a common situation.

6 Conclusion

YaKa is mature and it is intensively used at ENSIIE, it provides flexibility, saves system administrator time and has really made our computer site software homogeneous.

The complexity of system generation is given by the formula $\sum_{i=1}^{N} S_i$ where N is the number of machines, S_i is number of systems of ith machine. This is round to $N.\mathcal{S}$ where $\mathcal{S}$ the average number of systems by machine. At ENSIIE the complexity is $100.10 = 1000$ ($\mathcal{M}$ =100 and $\mathcal{S}$ =10). By extrapolating the ENSIIE experiment, it is possible to manage in a centralized way similarly computer sites up to ten times larger (to a complexity of 10000) before the generation time becomes prohibitive. Indeed it is possible to handle 500 machines with 20 systems on each machine or a super computer farm of 10000 hosts. Another limitation is the number of hosts that must be loaded simultaneously. It can be solved by using the YaKa file system and/or by using several synchronized *installation repositories*. For instance at ENSIIE, we use 2 *installation repositories* which allow to load 60 machines simultaneously in good conditions.

The YaKa innovation is that all systems are generated fully ready to use. Unlike others installation system tools, no work is performed once the system is downloaded on the machines. The YaKa experiments prove that this approach is valid even for a large number of systems. The main advantage of this approach is a very fast installation which allows to easily switch from a system to another and which increases security by allowing to restore an uncorrupted system.

YaKa comes with an innovating language allowing to describe and manage a computer site as a whole at all the levels: host hardware, systems of hosts, system description, links between hosts, software generation and installation. It allows also to define services in an abstract manner. Furthermore, the compiled approach offers to the system administrator a lot of static controls. So the system administrator can correct errors at compile time before any installation.
The language is powerful enough to describe a complete distribution. A distribution can be built using an existing binary distribution such as RedHat RPM[5] instead of source archives as we did. For that, only files are taken from the binary distribution, and configuration is done with the YaKa mechanisms. The YaKa compiler by adding automatically the dynamic libraries decreases the package dependencies of the initial binary distribution.

For people intending to experiment YaKa, it is interoperable with existing systems and other installation tools. Indeed for an existing computer site, it is possible to restrict the use of YaKa to a host sub-set, the others being simply declared in order to introduce it incrementally. When in 2004, we migrated the ENSIIE teaching network to YaKa, we started with one practical room, then the second one, then all the other rooms, then the user's account servers, and finally we split the single user's home server on 3 servers. The complete migration took about one and half year. This migration was quite long for security reasons, YaKa was under development.

Most computer sites we know of are either missconfigured or use obsolete system tools. For instance, half of them have the basic NFS tool wrongly configured. The main reason is that system administrators can not be expert in all basic system tools and standard distributions and/or installation tools provide only binaries and a default configuration. In fact often system administrators stop to configure a system tool as soon as it works. YaKa can help to change this situation. As YaKa offers the notion of services, a promising perspective is to build a generic layer describing computer site

over the YaKa language. We think that a few templates designed by specialists are sufficient enough to describe most of the existing computer sites. A standard system administrator will just instantiate the template matching his site to have all system tools well configured.

References

1. Paul Anderson, Patrick Goldsack, and Jim Paterson. Smartfrog meets lcfg: Autonomous reconfiguration with central policy control. In *LISA '03: Proceedings of the 17th USENIX conference on System administration*, pages 213–222, Berkeley, CA, USA, 2003. USENIX Association.
2. Paul Anderson and Alastair Scobie. LCFG: The Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002.
3. H. Peter Anvin. PXELinux. `http://syslinux.zytor.com/pxe.php`.
4. Ivan Augé. Yaka: An environment for describing and installing site wide systems. `http://yaka.ensiie.fr/`.
5. Ed Bailey. *Maximum RPM*. Sams, Indianapolis, IN, USA, 1997.
6. Mark Burgess. A control theory perspective on configuration management and cfengine. *SIGBED Rev.*, 3(2):12–16, 2006.
7. Mark Burgess and Ricky Ralston. Distributed resource administration using cfengine. *Softw. Pract. Exper.*, 27(9):1083–1101, 1997.
8. Hauke Goos-Habermann. *m23 manual*, 2008. `http://m23.sourceforge.net/docs/manual/html/en/manual.html`.
9. Paul Anthony Kasper and Alan L. McClellan. *Automating Solaris installations: a custom JumpStart guide*. SunSoft Press, Mountain View, CA, USA, 1995.
10. Thomas Lange. Fai - fully automatic installation. `http://www.informatik.uni-koeln.de/fai/`.
11. Mandriva. Mandriva pulse web page. http://www.mandriva.com/enterprise/en/products/pulse-2-0-the-industry-leading-and-open-source-technology-for-large-windows-and-linux-de.
12. Jeremy Mates. Kickstart & cfengine. `http://sial.org/talks/kickstart-cfengine/ks-cf-mod.pdf`, April 2005.
13. Microsoft. sysprep microsift report. `http://support.microsoft.com/default.aspx?scid=kb;en-us;302577`.
14. Microsoft. Windows Installer. `http://msdn.microsoft.com/en-us/library/aa372866.aspx`.
15. Redhat. kickstart installation. `http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/ch-kickstart2.html`.
16. Brett Schwarz. Hacking red hat kickstart. *Linux J.*, 2003(108):8, 2003.
17. G. Noronha Silva. Apt howto. Technical report, Debian, 2004. http://www.debian.org/doc/manuals/apt-howto/index.en.html.
18. Peter van der Weerd. Building a Jumpstart server for Solaris. *Sys Admin: The Journal for UNIX Systems Administrators*, 9(5):8, 10, 12, 14, May 2000.
19. Weibin Zhao and Henning Schulzrinne. Enhancing service location protocol for efficiency, scalabiliy and advanced discovery. *J. Syst. Softw.*, 75(1-2):193–204, 2005.