A service oriented compiler for operating systems

Ivan Augé
ENSIIE, 18, allée Jean Rostand, 91000 Evry, France
auge@ensiie.fr

Olivier Pons
CNAM Cedric, 292, rue Saint-Martin, 75003 Paris, France
olivier.pons@cnam.fr

October 14, 2008

Abstract

All tools dedicated to system administration of local area network follow the same scheme. For each host they install an initial operating system and then update it using a package manager. The package managers run independently on the hosts, this makes difficult (even impossible) to create and maintain a stable environment in which multiple hosts coexist and cooperate. Furthermore, over time, the update process, that upgrades, adds or suppress software, results in instability of host systems. Finally, a package corresponds most often to a single software with a basic configuration which does not match to the network requirements.
This article presents the YaKa framework which propose an orthogonal approach to this scheme in which the update feature almost disappears and is replaced by a very fast installation of a new clean system. All systems required in the network are generated all together by a compiler which has a complete view of the network. This general view allows to describe systems as a set of abstract services. A service groups one or several softwares and configures them for enabling a given functionality on one or several hosts.

1 Introduction

System administration of local area network is a task which becomes more and more difficult as the number of software, their complexity and interoperability grow. The designer of many operating systems attempts to help for this job by wrapping up standard tasks with shell scripts and a graphical user interface. Theses tools are helpful for a single or few home computers but fail to manage larger networks because they are close to a single system type and their use is time expensive. More recently, tools appeared to manage local area network. For handling a host, they follow the same scheme which consists of the installation of an initial operating system and then of updating it using a package manager. This scheme fails on several points.
Over time, the update process results in instability of host systems by adding, updating, suppressing software. This is mainly due to the weakness of a few packages and to software incompatibilities.
Effective administration of a local area network depends not only on having functional software on every single computer but also on their integration in the network as a whole. A modification of the system on a host may introduce new problems on other hosts and can result in havoc on the complete network. When installing a network software, package managers can not take into account how the software is installed on the other host.
For the system and network tools, there are a lot of ways to use a software and to make

them cooperate together. Most often a package corresponds to a single software with a basic configuration which does not correspond to the network requirements. This implies that the system administrator has to spend a lot of time to understand and configure them.

Our point of view is that these weaknesses are inherent in the incremental update of the standard scheme. We propose an orthogonal approach in which update almost disappears and is replaced by a very fast installation of a new clean system. All systems required in a network are generated all together by a compiler which has a complete view of the network. This general view allows to describe systems as a set of abstract services. A service groups one or several softwares and configures them for enabling a given functionality on one or several hosts.

This article present YaKa, a framework illustrating this approach. It is dedicated to the management computer sites heterogeneous both in software and hardware. It is available[4] freely and distributed under GNU Licence. The rest of this paper is organized as follows. We first give some background and summarise the problems, then we outline our framework and give some experimental results before concluding.

2 Managing computers

2.1 Definitions

Basic glossary

For a better understanding, we define here some general words or expressions that are used in this paper. A **computer site** is a set of computer in an organization, it consists of one or several Local Area network. The **system administrator** is the person responsible for maintaining and supervising the hosts of the computer site. An **end-user** is a person who uses client machines to run applications. Typically, a secretary using MS-Word in an office, or a student using a network sniffer in a practical course are end-users. A computer site consists of **server** and **client machine**. The role of servers is primarily to provide to the client general services such as user's authentications and homes. A **target** operating system is a system built on a machine for an other one. A **vendor** is a proprietary brand of an operating system such as Windows and Redhat, but not Linux.

Operating system

The operating system of a computer is a set of software cooperating to form a coherent whole in order to provide services to the end-users with a reasonable degree of comfort and privacy. In a computer site, these programs also cooperate with softwares of other machines.

An operating system is not a static entity. Brutal halt may result in loss or corruption of files. Over time, softwares are updated, new ones are added, some becomes obsolete. Similarly, computers with new operating systems are added to the computer site, others are suppressed and some are changed (see Figure 6.a). Finally, new users arrive and others leave. A consequence of such changes is that, over time, the operating systems tend to lose consistency, stability and privacy.

[6] mentions the increase of the operating system entropy over time, and proposes a detection and correction mechanism to avoid it.

Package

A software is basically a set of sources. To make it usable on an operating system on a given machine, it is first necessary to generate its objects from source (generation of package phase), then to install these objects on the operating system (installation of package phase)

and finally to configure it on the machine (configuration of package phase).
The main vendors use packages to install and maintain an operating system on a machine. For handling the packages, there are a few package formats (deb,RPM,Windows ...) which are IP (Industrial Property) and which come within a package manager [5, 15, 13]
For the software developer view, a package is a complex entity. It includes objects (executable and libraries), configuration files, installation and uninstallation scripts, functional dependencies with other packages. The package manager does not help him much in designing the package of its software.
For the system administrator view, the package manager checks the package dependency, installs the missing packages and performs automatically the phases installation of package and configuration of package . The generation of package phase which is done by the software developer, is generally hidden to the system administrator. Nevertheless, especially for software with complex configurations, the configuration of package is most often reduced to a default configuration (see Figure 6.b) considering only the presence or absence of specific software in the operating system. In this case, the system administrator must do himself the configuration of package .

Dependencies

To generate, install and configure software, one must take into account different types of dependencies. Dependencies can be local, if they only depend on software on the computer operating system. These dependencies can be global, if they depend on other machines in the computer site. There are 5 main types of dependencies between two softwares $\mathcal{A}$ and $\mathcal{B}$.

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

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

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

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

2.2 Manual management

Managing operating system is today based on package. Manual management of an operating system is dedicated to home computer and is presented Figure 1.a.
It begins with the boot of an installation system from a CD or a USB key. The installation

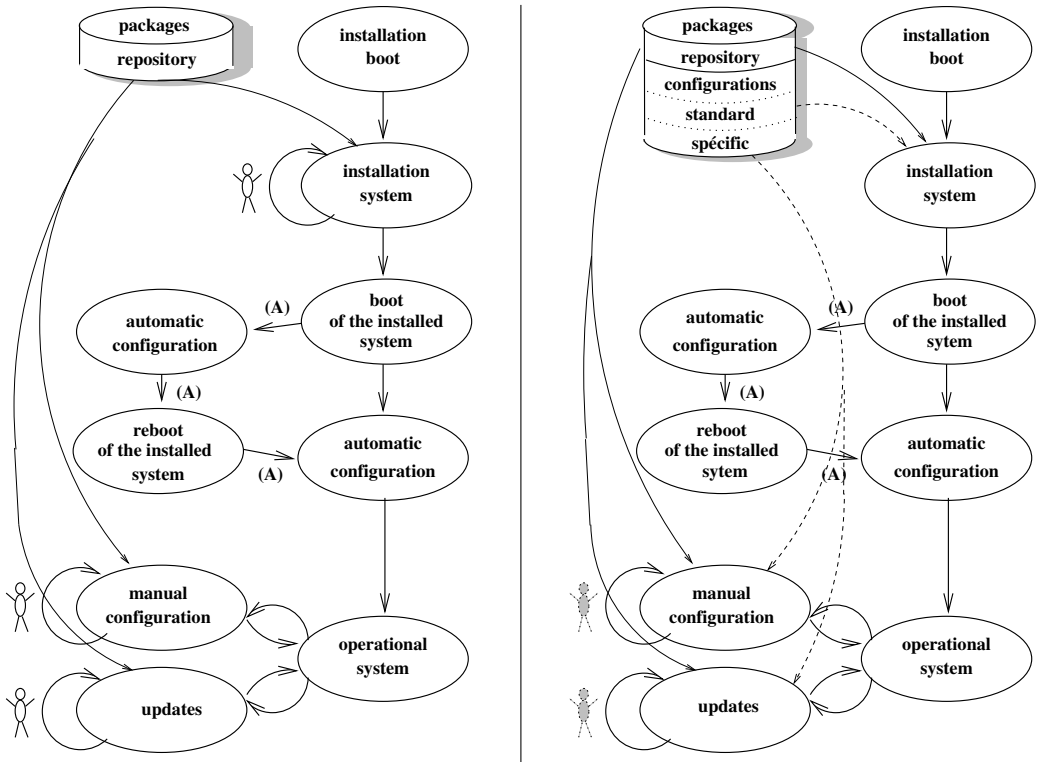


Figure 1: Phases of a computer installation and management.

system built the target system by interacting with the user and picking up the selected packages from a repository. The installed system is rebooted and it performs automatic configuration. This consists of hardware detection, and tools launching according to a set of configuration variables in a database on the computer. This phase may be done in two phases (A arcs).
After the automatic configuration phase, the operating system is operational. The user must log in as the administrator to configure tools that the previous stage has not or poorly configured. Later, when he wants to add or update softwares, he also needs to log in as the administrator.

2.3 Automatic management

The system administrator of a computer site is responsible for dozens or hundreds of machines and can not use the procedure planned for personal computer because the manual part is source of error and it requires too much time. To assist him in this task, existing tools [11, 14, 10, 12, 3, 7] operate on the general outline presented in figure 1.b.

The installation of a given machine begins with a boot which downloads an installation system via the network. The installation system fetches, in a standard configuration repository, the target system parameters and automatically built the system. Then it reboots the system it has just installed. This one performs automatic configuration consisting of hardware detection, and launching of tools according to a set of configuration variables in a database on the machine. This phase may be divided in two phases (A arcs Figure 1.b). After the automatic configuration, the operating system is operational for the software already configured by the previous phase. Then a robot fetches in the configuration repository the specific softwares to install or configure. This robot will also start periodically to update softwares.

Most of computer site management tools provides as a robot, a protocol to upload files or run scripts from the repository. Their use requires a lot of work in the design and realization of the configurations repository. De facto, system administrators often prefer to directly log in on the computers to set the configurations, and only use the management tool for the updates.

Some computer site management tools, such as [2], provide a language to describe the computer site and a compiler to generate the repository. But this repository contains **configurations scripts designed to run on the machine being installed** and as a result they do not have any information about the other operating system components nor an overview of the computer site except for a few parameters predefined by the tool. As consequence, specific configurations are not yet easy to achieve.

Finally all these tools are closely associated with a vendor and the package manager it uses.

3 Problems of software managing

We now discuss the main problems of computer sites installation and management tools.

Automatic Installation and Management

Regarding interoperability, most tools are strongly linked to an operating system and can not easily manage multiple systems of different vendors. To avoid this problem, [9, 8] proposes a language for describing operating systems with a directive allowing to select files in function of the vendor.

Regarding installations, they are not really fully automatic in the general case. In fact, they rely on a configuration repository, the creation and maintenance of which require a too big effort from system administrators. Installation is truly automatic only when the target system matches the standard scheme provided by the tool. This excludes the management of specific systems on client machines and the management of server operating systems. Moreover, the tools can not manage multiple systems of the same vendor on the same machine.

Regarding updates, all the tools are based on a package manager. The package manager is a component of the operating system. It runs on every machine and maintains a database of the packages already installed. The package database contains information such as version numbers and how to uninstall. In the automatic version, a robot fetches from the configurations repository the packages to install, uninstall or update and submits the works to the package manager. This mechanism gives rise to security problems (see paragraph Safety below) and over time, to an increase of the entropy of the operating system as mentioned in Section 2.1. To summarize these paragraphs, a complete reliable computer site management tool must allows:

1. to update an operating system without increasing its entropy and the entropy of the computer site,
2. to provide tailored operating systems,
3. to automatically install multiple operating systems of the same vendor on one machine,
4. to install automatically operating systems of different vendors.

Speed of the installation

After the automation of the installation, the second criterion is speed. Indeed, the reinstallation of an operating system is the safest way to restore a stable state. However, if the

installation is too long, system administrator would prefer to manually correct the malfunction; Such manual correction on several machines of the site may increase the system entropy. As shown Figure 1.b, the duration of an automatic package based installation of an operating system is split into: 2 boots, 2 system initializations, the installation of packages (downloading and writing on disk), the configuration of packages . For an image based automatic installation, this duration is divided into: 2 boots, 2 system initializations and the image installation (download and writing on disk). Among these steps only the duration of the boots and the image installation cannot be reduced.
The use of tailored systems instead of general ones, reduces the number of packages to download and decreases the duration of the installation of package .
The initialization of operating systems is long because it is done by general scripts which spend more time detecting the initializations to do than to perform them. Using initialization scripts tailored for the operating system can reduces this time.
The configuration of packages can be avoided if the downloaded packages are already configured for the operating system and the target machine.

Package granularity

In order to build tailored operating systems, the package hard and soft dependencies are a critical issue. This raises the problem of package granularity. In figure 2.a, packages $\mathcal{B}$ $\mathcal{C}$ and $\mathcal{D}$ depend on $\mathcal{A}$ but in different ways. For being able to tailor an operating system, the

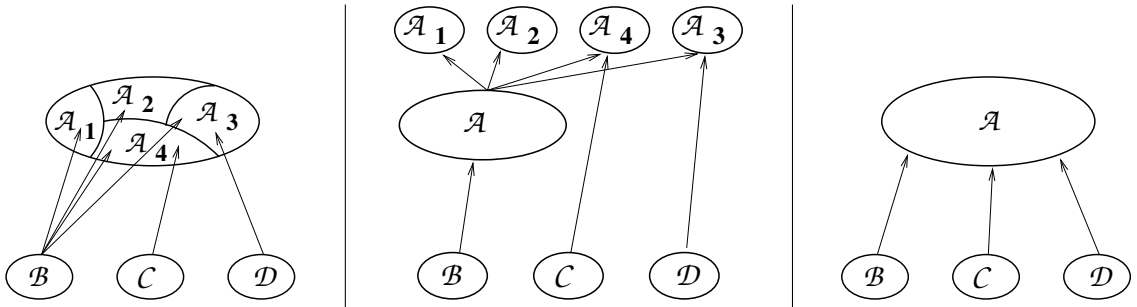


Figure 2: Package granularity problems.

$\mathcal{A}$ package must be split into a set of sub-packages $\mathcal{A}_i$ as shown Figure 2.b. This approach increases the number of packages and so the complexity of their maintenance. Furthermore the division of $\mathcal{A}$ is not an intrinsic property of $\mathcal{A}$, but derives from current and future packages based on $\mathcal{A}$. For these reasons, packagers do not split a software into sub-packages as shown Figure 2.c and thereof the $\mathcal{A}$ package is installed each time that $\mathcal{B}$, $\mathcal{C}$ or $\mathcal{D}$ are installed on an operating system. For the $\mathcal{C}$ and $\mathcal{D}$ packages which use only pieces of $\mathcal{A}$, this affects the installation speed if $\mathcal{A}$ is big and it imports all potential vulnerabilities of $\mathcal{A}$ (see Figure 6.c).

Safety

In this section, we focus on general security of computer sites. It consists of reliability, confidentiality and efficiency. They depend on the choices made in the 3 main phases of the computer site installation and maintenance.

⇒ **General infrastructure:** There is 2 levels for setting the general infrastructure of a computer site.
At the logical level, the system administrator must choose among other thing, the services, the softwares providing them, how to balance load, how to set up redundancy for

services supporting it, the backup system.
At the physical level, it consists of splitting the computer site into sub-networks and of choosing the machines with the hardware that is more adapted to the required services. This requires great skill and experience, to be aware of the many existing software alternatives for each service and to be aware of the compatibilities and of the incompatibilities between these softwares.
Finally, the main benefit in reliability and efficiency results directly from the general infrastructure.

⇒ **Configuration:** All software chosen in the previous phases must be configured. In the simplest case this configuration consists of setting some configuration variables (see Figure 6.d), in more complex cases it consists of creating files and setting dozens or hundreds of variables (see Figure 6.e). Sometimes, it requires to write some scripts or programs in order to allow some softwares to cooperate (see Figure 6.f).
Finally, the main benefit in confidentiality results directly from the configuration. This phases is the second source of benefit in reliability and efficiency.

⇒ **Software weakness correction:** Many software have potential security vulnerabilities. To reduce this risk, the first prevention is to install and run only the necessary software. For these software, the system administrator must apply the security patches provided by software distributor as soon as a vulnerability is discovered. But this should be done **before that the system has been attacked**. This condition is not easy to determine so in doubt, a complete reinstallation of the operating system is needed. This reinstallation is not usually done by lack of automatic installation procedure.
Finally, even if it is given a lot of media coverage to the software weakness, it is quite rare and much less important than the former phases.

Assisting the system administrators

The preceding paragraphs show the complexity of organizing and managing a computer site as a whole. We should admit that the help supplied to the system administrators by both the vendors and the installation tools is insufficient. Indeed, most of packages provide binaries, sometimes a default configuration, which generally does not correspond to the computer site requirement (see Figure 6.b). Similarly, for using installation tools the system administrator must install the configurations of software in the configuration repository (Figure 1.b). The impossibility of packages to configure the software comes from the fact that on the one hand some of them are complex frameworks and on the other hand that the configuration also depends on other machines of the computer sites.
Each system administrator has to read the documentation of these software and to experiment them in order to clearly understand them. This learning may take several weeks to finally get a configuration equivalent to the one of his colleagues on other computer sites or to an imperfect configuration.
A computer site management tool should enable the configuration of such software or groups of such software based on the computer site infrastructure.

4 Description of the YaKa framework

The YaKa framework[4] provides a language, a compiler for this language and an installer. To start with YaKa (see Figure 3), 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* .

4.1 Functional view

The functional view of the software is presented Figure 3. There are three stages. The first is the description of computer site, the second is the compilation of the description to generate the *installation repositories* and finally an installation on each machine of the computer site.

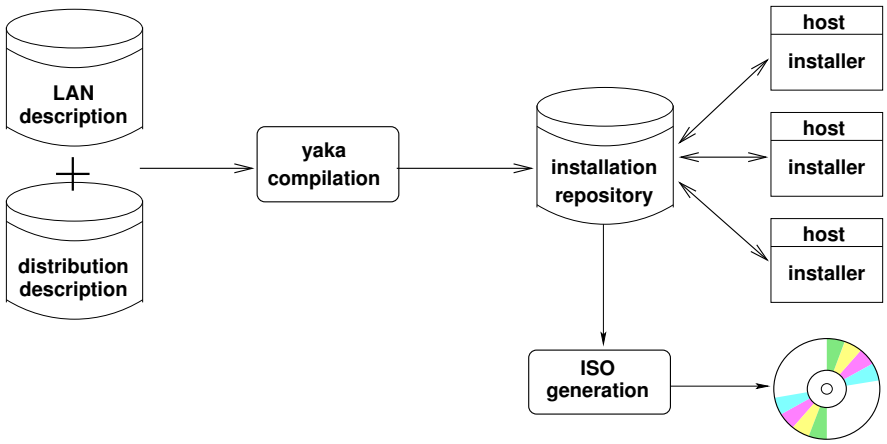


Figure 3: functional view of YaKa.

Description

Our description language permits to describe precisely and in a **global way** a full computer site. For that, it supports several abstraction levels described below:

Computer level: A machine is a set of physical parameters (MAC address, disks, graphics card, screen resolution, ...) and logical parameters (name, network address, gateways, disk partitions, users, ...).

System level: At this level, the operating systems are described and bound to the machines of the computer site. A system is defined as an image which is a set of couple $(disk\ partition, file)$ or as a set of component called services.

Service Level: A service is an entity that extends the usual notion of package. It includes the elements listed below. All these elements are optional.

- others services on which it has a hard dependency.
- a script to generate all files (binaries, libraries, ...). In concrete terms, these files are obtained either by unpacking one or more tools distributed as binary code, or by compiling one or more tools distributed as source code.
- a section listing all $fixed$ files of the service to be installed on the target system. We call $fixed$ file, a file which is copied such as to all the target system. It is a subset of the files generated by the previous script but other files picked everywhere can be added.

- a set of *s*oft files. They correspond to the service configuration files to be installed on the system. They are called soft because they are generators. The file to be installed is the result of this generator.

Our language also supports object oriented mechanisms such as inheritance and dynamic linking. Furthermore, as shown Figure 3, this description is divided in two parts. The distribution groups the standard services, it can be shared by several computer sites. The LAN description defines the specific services of the computer site and includes the description of the computer site at the computer and system levels.

compilation

According to the description, the compiler generates the *installation repositories* . A *installation repository* contains the download-units and the servers DHCP, TFTP, FTP with their complete configurations and a script to start them.

The compiler run the generators of the *s*oft files in the context of the target system and machine (the generator of a soft file presents in S systems to be installed on M machines will be run $M.N$ times). These generated files correspond to the configuration files.

Every system on each machine is generated **ready to use**. For example in the general configuration script of a machine, we may find for the keyboard configuration the line "`/sbin/loadkeys fr-latin1`" or "`/sbin/loadkeys us`" depending on whether the keyboard is French or American. This can be compared to classic Linux distributions where the same function is achieved by several dozen lines of script possibly reading auxiliary data files. A consequence is that ready to use systems start faster (a few tens of seconds) and moreover they only start what is really necessary avoiding potential security holes.

The compiled approach permits to the *s*oft file generators to know a priori (a) the system in which the service fits, and (b) the systems of the other machines of the computer site. This is far more efficient than the a posteriori approach commonly used by the package managers. Indeed, for (a) they must find in the file tree if a given service exists and how it is configured. For (b), the only solution is to get the information by network mechanisms such as broadcast discovery, interrogation of centralized servers. These mechanisms (see for example [1, 16]),complicate the structure of computer site, decrease its stability and reliability and are potential security problems.

Within a compiled approach, all the files of a target system with a lot of information are present in the compiler data structure. Such information may be the file type, the dynamic libraries it requires and so on. This makes it possible to handle precisely all the dependencies mentioned in 2.1 (page: 3). Especially the YaKa compiler implements the hard cross dependency and the soft dependency. This later allows to drastically reduces the packages dependency graph without modifying the package granularity.

Finally, the compiled approach also allows to do many **static checks** leading to more reliable systems and to saves system administrators's time.

Installation

The *installation repository* being created and the servers being started the installation mechanism of a computer is as follows:

1. The user requests a network boot on the machine.

2. The DHCP-BOOTP server returns a PXELINUX boot which displays a menu describing all the systems that can be installed on this machine. One of the entries in this menu allows to go back to the disk boot and start a system already installed.

3. The user selects the system(s) to be installed.
4. Using TFTP the PXELINUX boot downloads from the installation repository, a Linux kernel and a minimal Linux system in a ramdisk. Then it starts the minimal system giving as kernel parameter the names of the systems to be installed.
5. The minimal system starts the YaKa installer instead of the standard *init* program.
6. The YaKa installer performs the following operations:
   - It gets in the kernel parameters the names of systems to be installed.
   - Using these names and the MAC address, it searches for the configuration of the machine and for the tasks to be done to install each system. This is done by reading in the ramdisk, a file describing all systems of all machines of the computer site,
   - It initializes the machine (network, disks).
   - For the systems based on services, the installer downloads the download-units from the repository via FTP and unpacks them on the disk. For the image based systems, it downloads the images and writes them unchanged on the disk.
   - it installs a multi-boot.
   - it reboots.
7. The user selects the wanted system in the multi-boot menu.

The YaKa installation is very fast because it requires only downloading and writing on the disks the download-units. Unlike the installation tools based on package, no treatment such as software configuration is required. Moreover, being really totally automatic, installation and reinstallation can be done by an end-user.

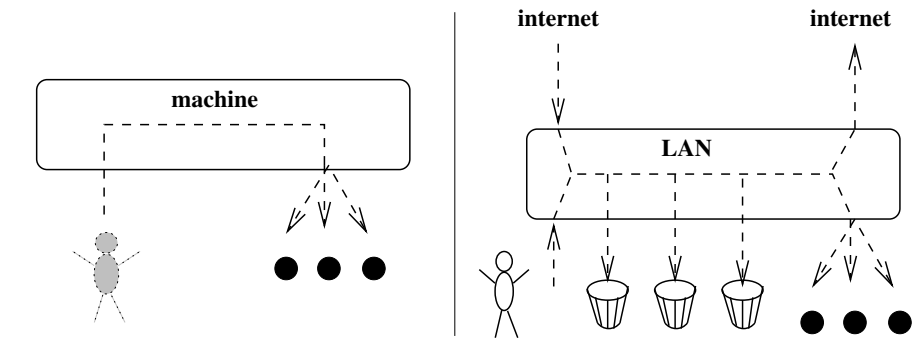
4.2 Service

The service is the innovative concept of YaKa. The idea is to define a function independently of the software it uses and of the machines where they run. It is illustrated on the example of the email service.

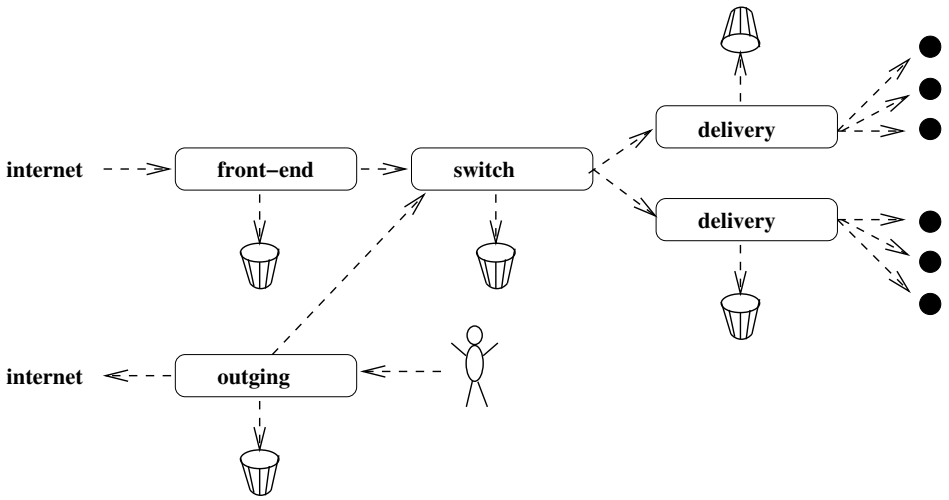
The basic function of email is to enable a human or software agent to send messages in mailboxes. As illustrated in figure 4.a this function can be more or less complex. The simplest case (on the left hand in the figure) is a single machine which contains only users and mailboxes. It is mainly used for the administration of the machine. The complete function (on the right hand in the figure) is more for human agent. It provides to the users a mailbox. It allows them to send messages to the other users of computer site and to external mailboxes, and to receives in their mailboxes messages from the local users and from the external world. Today, because of spam and viruses, incoming messages must be filtered, marked and undesirable messages must be suppressed.

Figure 4.b presents the general scheme of the email management. From outside to inside, the `front-end` receives the messages, applies filters and transmits them to the `switch`. This one applies more complex filters and forwards the messages to the `deliveries`. Theses last daemons can still apply filters before writing messages in the user's mailboxes.
From inside to outside, `outgoing(s)` retrieves all messages from the inside and sends them to the external world or to the `switch` depending on the message recipient.

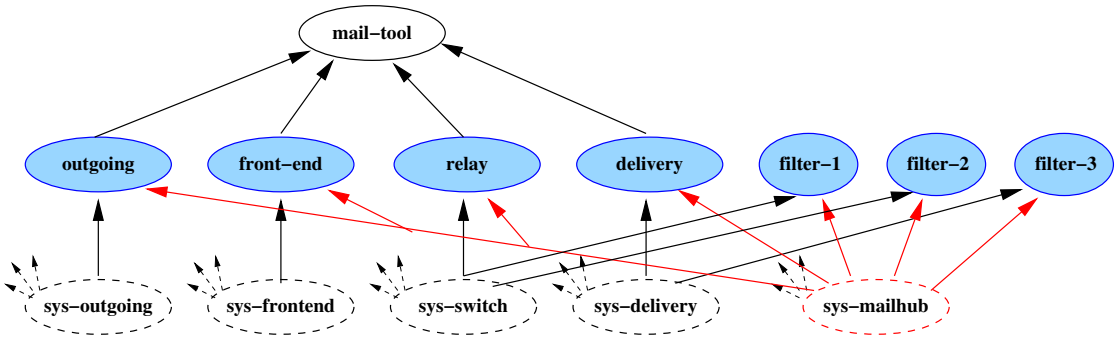
In Figure 4.c, filled ovals represent the basic services. They services, Except filters, they inherit of the same internal service because it is the same software but configured differently. By inheriting from these services, The components *sys-front-end*, *sys-switch*, *sys-delivery* and



a) general function



b) processing chain



c) services and systems

Figure 4: Service and package.

sys-outgoing are built by inheriting these basic services and they are the email part of systems running on the servers presented Figure 4.b.
We can combine the email services to get all possible pattern such as *sys-mailhub* which brings together all features on the same machine.

This example illustrates the concept of service that separates the notion of a software tool from that of package. Indeed, the services `delivery` and `outgoing` are the same mail software but configured differently, the service `front-end` groups filters and a mail software. This software is configured to use filters and only propagate messages from the outside to the inside.

The compiled approach greatly facilitates the implementation of the concept of service because it allows us to have all the needed information to generate the configuration files. Finally, the global approach gives a knowledge of the services of all machines, and it permits to increase the coherence of the computer site. For example, email browser services can be automatically configured by looking for the machines providing the `delivery` and *outgoing* service.

5 Experimental results

To prove the validity of the approach, we developed a consistent Linux distribution, we described the computer site of the ENSIIE teaching network (a postgraduate school in computer science), and we deployed it in 2005. It is still used in production today.

The ENSIIE teaching network consists of 400 users, 150 machines (a dozen servers, the others being self-Service computers for students), dozen of operating systems (Windows, QNX, YaKa-Linux).
In this example, the YaKa compiler running on a PC P4-3GHz, takes about 6 minutes to generate the installation repositories that contains 1449 YaKa-Linux systems, 78 Windows and QNX systems.

Let us resume this experience in respect of the problems presented section 3 (page 5).

Automatic Installation and Management

The installation of operating systems on a machine is 100% automatic since *installation repositories* and the minimal installation system are created by the YaKa compiler. At ENSIIE, installing specific operating systems or reinstalling standard ones are left to end-user.

1. The tool provides a sustainable solution for software update breaking with the online software update. It prefers a full reinstallation of a stable system previously updated.

2. The tool can easily produce tailored YaKa-Linux operating systems. The smallest is the minimal installation system which is also described in the YaKa language. At the ENSIIE, we have described specific systems such as one dedicated to server, one dedicated to development, one dedicated to office work, one dedicated to system and network teaching (expurgated of standard software such as word processors, enriched by specific tools such as network sniffer and with an unprotected administrator account).

3 & 4 The tool allows to install any number of systems. The YaKa-Linux systems being downloaded via a package approach and the systems of others vendors via images.

Speed of the installation

The speed of installation based on package is due to several factors: 1) the possibility to install tailored operating system reduces the amount of data to download, 2) the system start-up script being also tailored, no time is spent for detecting what should be initialized, 3) finally, the packages being configured by the compiler, there is no longer any configuration of packages during the installation.
For the 10baseT network presented Figure 5 and for about twenty machines, the duration from the time one switches on the computer until the "login" prompt appears, is less than 2

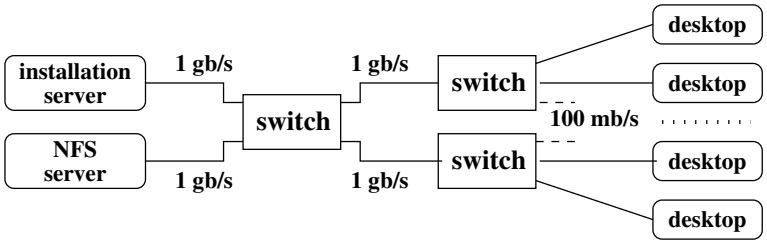


Figure 5: 10baseT 1Gb/100Mb Network.

minutes for the system dedicated to network teaching and, 5 minutes for the system dedicated to office work. and to office work.

Package granularity

The compiler implementing the *soft dependency* allows to suppress a lot of *hard dependency*. With the global approach a package designer can add only some piece of a package to an other, this allows also to suppress a lot of *hard dependency*. These to features reduces drastically the complexity of the package dependency graph.

Safety & Assisting the system administrators

The service concept breaks the traditional association between package and software. The compiler allows to configure services taking into account other services running in the same system or in other system of the computer site.
The functional abstraction of the services, allows to use complex software or a group of software without having to worry about the details of their configurations. This solves a lot of security problems. This abstraction provides service library that are really helpful to system administrators.

6 Conclusion

The approach proposed by YaKa and validated by its use in production at ENSIIE is a good solution to the problems of computer site installation and management. This approach is based on two innovations:

- ⇒ A language which allows to describe a computer site as a whole and at all levels: the machine hardware, the operating systems to install on the machines, the system description as image or as a set of services, the service generation.
- ⇒ A compiler generating ready to use operating systems. This is the opposite of the installation and management tools based on a package manager running on each machine.

Compared with other tools, these innovations provide especially the following advantages: The compiled approach allows to perform a lot of static verifications on the generated operating systems. The global view of the computer site allows to define a service as an inter-

a) Adding a slave DNS server requires to change the DNS configuration (eg: the `/etc/resolv.conf` file for Unix) of all machines of the computer site.

b) The package of *rwall* tools 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.

c) For example installing the small converter `a2ps` in Mandriva requires among other things a full installation of the graph visualization suite (`graphviz`) and of the powerful image management suite (`ImageMagick`).

d) The configuration of the common `less` GNU command requires only to have the LESSCHARSET environment variable set to the right character set.

e) The configuration of the SAMBA software requires several configuration files the main of which beeing the `smb.conf` file having several hundreds of variables. Furthermore, when authentication and home servers are not placed on same host, SAMBA must be configured on every server in a different way.

f) To ensure a reliable synchronization of a SAMBA and NIS password database, a small program is to be written. This is invoked by the SAMBA framework when a user changes its password and its role is to change the NIS database too.

Figure 6: List of concrete examples.

software and inter-machine entity. Finally, the generated systems that are ready to use, start very quickly.

At the ENSIIE, the YaKa experience is now extended in three directions. The first is to improve the computer site description language and to improve the YaKa-Linux distribution with new services. The second is to define services over a major Linux binaries distributions (Redhat, Debian, ...). Finally, the last one is to achieve a general public installation by the Web.

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. Large scale linux configuration with lcfg. In *ALS'00: Proceedings of the 4th conference on 4th Annual Linux Showcase & Conference, Atlanta*, pages 42–42, Berkeley, CA, USA, 2000. USENIX Association.

[3] Paul Anderson and Alastair Scobie. LCFG: The Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002.

[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] M. Burgess. Computer immunology. *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, page 283, 1998. `http://www.usenix.org/publications/library/proceedings/lisa98/full_papers/burgess/burgess.pdf`.

[7] Mark Burgess. A control theory perspective on configuration management and cfengine. *SIGBED Rev.*, 3(2):12–16, 2006.

[8] Luke Kanies. Puppet: Next-generation configuration management. *;login: the USENIX Association newsletter*, 31(1), February 2006.

[9] Luke Kanies. puppet. Technical report, reductive labs, 2007. http://reductivelabs.com/.

[10] Paul Anthony Kasper and Alan L. McClellan. *Automating Solaris installations: a custom JumpStart guide*. SunSoft Press, Mountain View, CA, USA, 1995.

[11] Thomas Lange. Fai - fully automatic installation. `http://www.informatik.uni-koeln.de/fai/`.

[12] Microsoft. sysprep microsift report. `http://support.microsoft.com/default.aspx?scid=kb;en-us;302577`.

[13] Microsoft. Windows Installer. `http://msdn.microsoft.com/en-us/library/aa372866.aspx`.

[14] Redhat. kickstart installation. `http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/ch-kickstart2.html`.

[15] G. Noronha Silva. Apt howto. Technical report, Debian, 2004. http://www.debian.org/doc/manuals/apt-howto/index.en.html.

[16] Weibin Zhao and Henning Schulzrinne. Enhancing service location protocol for efficiency, scalabiliy and advanced discovery. *J. Syst. Softw.*, 75(1-2):193–204, 2005.