Managing and Distributing Application Software.

Ph. Defert, E. Fernandez, M. Goossens,
O. Le Moigne, A. Peyrat, and I. Reguero
CERN, European Laboratory for Particle Physics

Managing and Distributing Application Software.

Ph. Defert, E. Fernandez, M. Goossens, O. Le Moigne, A. Peyrat, I. Reguero
– CERN, European Laboratory for Particle Physics

ABSTRACT

The paper describes a project for distributing application software in the large worldwide High Energy Physics (HEP) community. Hundreds of packages are maintained centrally and users can access them directly through the network. Workstation administrators can optimise access performance and reliability by specifying in their installation scripts the packages to be copied locally or accessed remotely. Product maintainers have a set of tools to generate their packages from the sources, while site administrators can replicate (part of) the central file-base manually or automatically. The generation takes place in different physical network domains. Replication on remote domains implies only propagating changes of the repository.

Introduction

Nowadays, UNIX systems almost always come with the bare minimum of software and in most cases this is insufficient for program development or data analysis. Commercial software tends to be specific and often too expensive. Thanks to the ubiquitous Internet, many users, especially those working in the scientific and educational fields, can access a lot of excellent software, such as editors, window managers, document formatting utilities, drawing tools, etc., most of which are freely distributed in source form.

The ultimate goal of ASIS (the Application Software Installation Server) project is to offer users access to a large number of freely available UNIX applications in order to optimise their productivity and working environment. In our system, end-users do not need to obtain, configure, generate, install and manage the software themselves. Managing the collection of applications is performed centrally by a team of system engineers with dedicated tools to generate and install the packages.

The *repository* is the place where applications are stored in an executable format. The application domains presently covered by ASIS are very diverse: the CERN Program Library, the core of the High Energy Physics (HEP) data analysis software, most GNU packages, the latest TeX/LaTeX setup, MIT X-windows and contributions, TCL/TK based software, PERL based packages and many other tools written by the UNIX community. Presently, the system gives users access to some six hundred products for eight different platforms, i.e., the hardware of SUN, DEC, IBM, SGI and HP with their different UNIX flavours. The number and kind of supported platforms are constantly reviewed according to user demand and available resources. The repository contains a data base describing all packages with version control and support information. It also includes tools for users who want access to ASIS, for product maintainers to perform their management tasks and for remote sites to replicate the environment locally.

A user gets access to the software on an ASIS Local Copy by running the *client* software. With a small addition to the local system setup file, the *ASISwsm* (ASIS WorkStation Manager) script will be executed for each change in the ASIS Local Repository. In this way, new versions will be visible on the local machine and all software package versions will remain in step. This part of the system is specially kept simple since it is used by all UNIX-based workstations at CERN.

A suite of tools helps *product maintainers* keep their packages up-to-date. These tools automate the generation and installation process, with quality and reliability receiving maximum attention. Version control and documentation are also given great care.

The *replication* software allows a remote site or a cluster at the parent site to replicate a master repository on its local infrastructure. The script *ASISlcm* (ASIS Local Copy Manager) is run each time the local administrator decides to update the local copy of ASIS.

The Repository

The structure of the repository was designed with great care and special attention was given to the maintainability and ease of use of the client software, management tools and replication procedures.

ASIS can handle multiple platforms and versions. This means that each product is available for all (if possible) supported platforms, and that several versions of the same product can be present simultaneously in the repository. The "Reference Copy" of ASIS is used to manage the different packages, but should never be used by end-users. "Local Copies" of ASIS are built using the "Reference Copy". Users access a "Local Copy" of the ASIS repository via a distributed file system, independently of the type of file system on which it resides. For instance, at CERN, "Local Copies" of ASIS are available on several distributed file systems: NFS, the SUN Networked File System, AFS, the Andrew File System, and more recently DFS, the OSF/DCE Distributed File System. Each distributed file system has its own "Local Copy" of the "Reference Repository". Users never access the "Reference Copy" directly.

Heterogeneous Platforms

Because we are working in a heterogeneous environment, i.e., a combination of more than one architecture and operating system (O.S.), the software is partitioned into two parts:

- a *specific* part, for each O.S., containing binary files and libraries;
- a *share*able part, usable on all platforms, containing scripts, include files, fonts, startup files, man pages, etc.

Each *specific* part must contain all needed references to the *share*d part to make sure that the repository and the products are self contained. In order to avoid unwanted site dependencies, all symbolic links are relative and independent of the root directory of the project. This makes the software easily exportable.

Platform naming conventions in ASIS are based on the ones adopted by the GNU project and by AFS. The result of the GNU *config.guess* script is the concatenation of three strings, the chip name, the vendor and the operating system with its version number. AFS names are directly determined by the AFS project. For efficiency in AFS, our most-used distributed file system, and for historical reasons, the ASIS platform identifier is the AFS one. GNU names are also

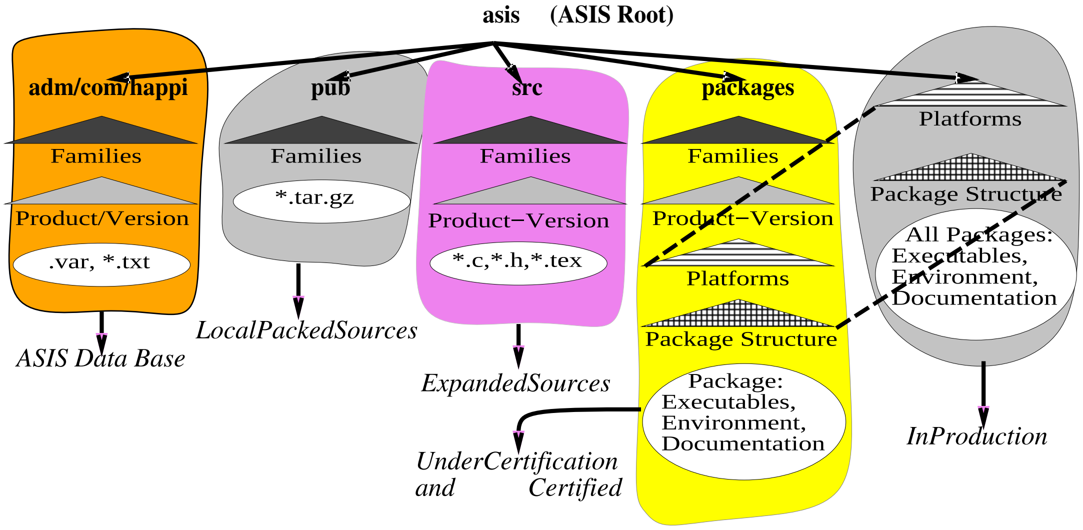

Figure 1: The Structure of the ASIS Reference Copy (or Repository)

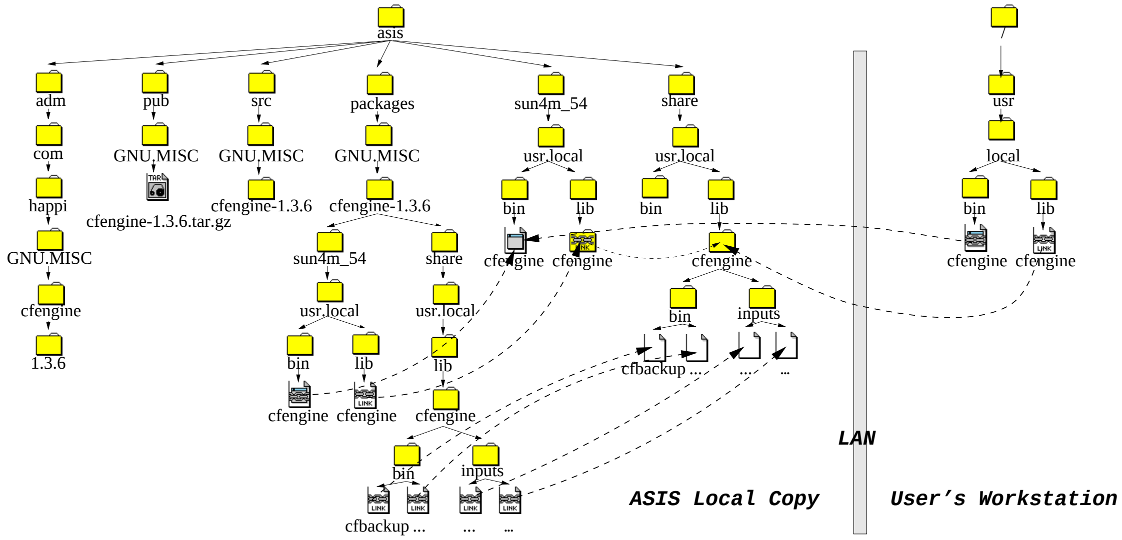

Figure 2: The *GNU.MISC/cfengine-1.3.6* product in an ASIS Local Copy and on a User's Workstation

supported (see Table 1). The common part to all platforms is named *share*.

GNU name	ASIS/AFS name
Currently Supported	
sparc-sun-sunos4.1.3	sun4c_411
sparc-sun-solaris2.4	sun4m_54
mips-dec-ultrix4.3	pmax_ul43
hppa1.1-hp-hpux9.05	hp700_ux90
rs6000-ibm-aix3.2.5	rs_aix32
rs6000-ibm-aix4.1.4	rs_aix41
alpha-dec-osf3.2	alpha_osf32
mips-sgi-irix5.2	sgi_52
Planned Support	
hppa1.1-hp-hpux10.10	hp700_ux100
i386-unknown-linux207	i386_linux2

Table 1: Standard names for various platforms

Versioning

In the repository, multiple versions of a package can coexist. Users have access to versions which can be:

- *InProduction*: the currently supported version which is included by default in the user's environment;
- *Certified*: a version which is present in the repository, but made available to users only on request.

The 'Software Processing Model' section gives formal definitions of these two terms.

Generally, only one version is *InProduction*, i.e., formally supported but, exceptionally, more than one version of a software package can be put *InProduction* (for instance to ease transition to a new major release). For instance, presently, at CERN, several versions of *PERL/perl* and *TCL/tcl* are *InProduction*.

Certified products are stored in their own separate directory structure containing both *specific* and *share*d data, such as in the NIST Depot [9]. The "ASIS package format", shown in Figures 1 and 2, defines the following directory structure:

- <*family*>/<*name*>-<*version*> product name;
- <*platform*> identifier;
- execution directory as seen by the user. In our example, it is *usr.local* representing */usr/local*.[1]
- The subdirectories needed by the particular package.

For reasons of convenience, products have been classified in different families like *GNU.EDIT/emacs*, *MISC/xemacs*, *X11.R5/fvwm* or *INET/pine*. The family can identify an origin and a function (*GNU.EDIT*), an application domain (*X11.R5*, *INET*), or a catch-all category (*MISC*). Search tools are available to find a product by specifying a regular expression for its name or one of its commands.

On the other hand, all products *InProduction* are stored in the same directory tree (see Figure 1). Files are placed by *specific* platform and *share* directly below the ASIS root directory, i.e.,

- <*platform*> identifier;
- execution environment directory (*usr.local*);
- all subdirectories needed, i.e., those required by at least one package *InProduction*.

The correspondence between files and products is registered in the ASIS data base. Only one physical copy of each product is kept in the repository. Thus, when a product is put *InProduction*, all files associated with the given product are moved from the *Certified* to the *InProduction* area.

ASIS Client Software

Users can have access to the contents of ASIS, its description and its documentation.

Access to the Data Base

The ASIS data base contains:

- the list of families with their description;
- the list of products with their description, including version-specific information;
- the state and history of transitions of all versions of a product;
- support information for each product, namely, the author's name and address, the local product maintainer's coordinates and the support level for the package; this support level goes from A to E depending on the availability of the local product maintainer to help the users and track bugs;
- the list of commands and files corresponding to each package;
- the description of how to install multiple versions of a particular product in the user environment, if applicable;
- the list of platforms supported in ASIS, active or frozen, O.S. support included or not, etc.

The script *ASISinfo* offers line mode access to the data base described above. Daily, an HTML copy of the data base is generated using the *ASISinfo* library. In addition, HTML pointers are created to the actual documentation distributed with the packages. Any WWW browser can be used to access these pages.

Access to the Applications

At CERN, as well as in most HEP centres, the UNIX support teams have chosen */usr/local* as the local root prefix for public domain applications (see [14]). One important requirement formulated by many users and some individuals was the possibility for their system administrator to be able install in */usr/local* software which is not contained in ASIS. As a consequence, ASIS has to maintain the directory tree

[1]Another possibility could be *usr.asis* for a system where the supplementary ASIS packages are installed below the */usr/asis* root directory. */cern* is the root for the CERN Program Library.

below */usr/local* on each client workstation in a very careful and coherent manner.

The *ASISwsm perl* script builds the directory(ies) where applications should reside (in this case */usr/local*) in such a way that all packages have the correct execution environment. It creates links from the directory */usr/local* to the distributed file system with the corresponding "Local Copy", for instance, at CERN, */afs/cern.ch/asis*, */nfs/cern.ch/asis* or */:/asis* (the DFS local cell). The script does not modify user files but reports possible file name conflicts between ASIS and the local environment.

A local system administrator can customise the behaviour of *ASISwsm*, specifying a set of options. These include:

- ignore a product;
- copy a product locally, instead of generating links to the distributed file system;
- generate links only for a product, even if the product maintainer forces a local copy;
- install a non-default version (i.e., not *InProduction*); this can be a specific version or the *Latest Certified* one;
- enable or disable overwriting user files.

This customisation is performed by modifying the local configuration file of *ASISwsm*. A line mode interface as well as a TK-based configuration editor are available (Figure 3 shows this TK-based product configuration window).

In addition, *ASISwsm* also allows the product maintainers to assign a different access type or to define post-installation commands. For instance, one can force a local copy for login shells like *GNU.SYS/bash* or *MISC/tcsh* or, it might be necessary on some platforms to give the root set-uid bit for the command *xterm* of the *X11.R5/mit* product.

The ASIS repository evolves rapidly as products are continuously updated, bugs are corrected or new packages are introduced. Thus, *ASISwsm* should be run on a user's workstation at the same frequency as the corresponding "Local Copy" is updated. Anyhow, it can be run more frequently without harm since it only uses minimal system resources. If *ASISwsm* detects that the local copy of ASIS has not changed since its last run, it exits smoothly without doing any work. Moreover, *ASISwsm* is an idempotent command, i.e., its result is identical if it is fully executed once or multiple times. In other words, interruptions can be recovered by re-running the command.

Most users install the default environment *InProduction*. As all files of *InProduction* packages reside in the same "file system" or "volume" and "directory tree", *ASISwsm* can optimise the number of links to be created in the execution directory. For instance, if all links to be created in a directory point to the same target directory, then a link is created at the directory level rather than creating links for each file.

Similarly, to improve performance and reliability, replicas are created for the most frequently accessed file systems, i.e., the more common platforms. Replication is possible with sophisticated automounters in the NFS world and is part of the distributed file system itself, like in AFS and DFS.

As a side-effect, the way that the *InProduction* area was designed allows users to get access to ASIS by creating simple links pointing from the directory */usr/local* to the *InProduction* area. Note that in this

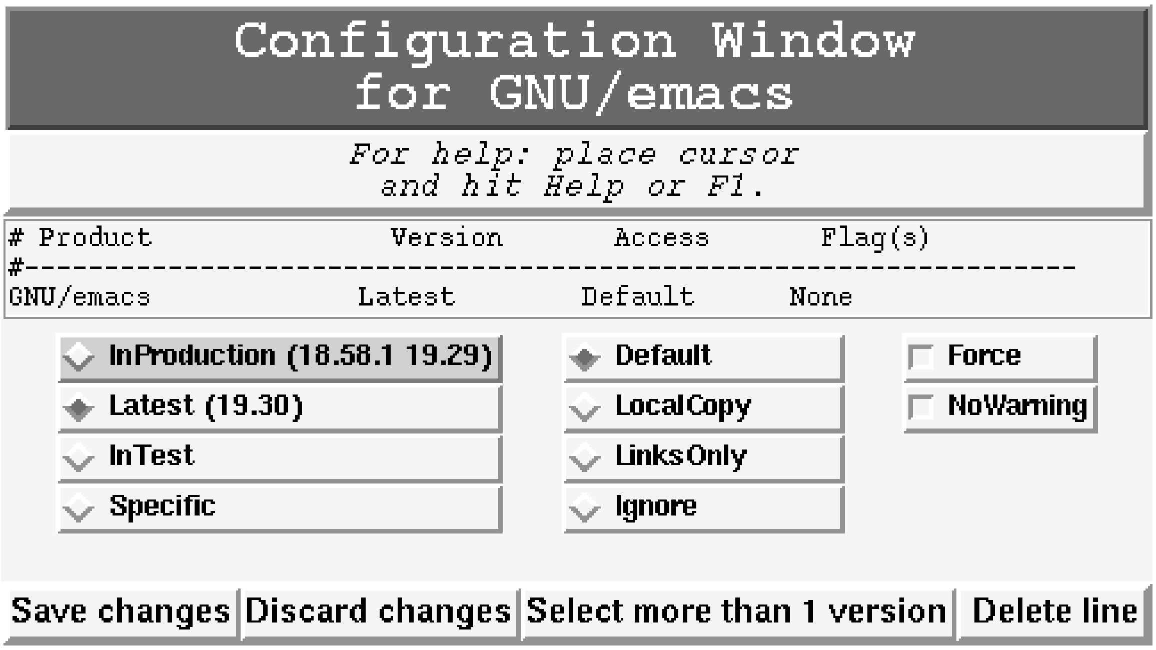

Figure 3: The "product configuration" window

case it is impossible to install anything else in */usr/local* or to use other versions than those *InProduction*.

Per User Environment Customisation

On a user-based system, different users may have different requirements as to which version of a package they need or want. On the other hand, the script *ASISwsm* customises software access at the level of the workstation. Therefore, it was necessary to introduce an ASIS tool to customise a user's environment. This tool is quite similar to the *setup* command in the UPS/UPD project at Fermi-Lab (see [2]).

The command *ASISsetup* is called by a user to change the environment to get access to a version of a product different from the default on the local system. Unfortunately, this procedure is not always possible. Some commands cannot be customised using shell environment variables. Moreover, some product maintainers cannot devote enough time to enter this setup definition in the ASIS data base.

The Repository Management Model and Software

The Software Processing Model

For over a year, we have installed public domain packages manually. During that period, we were able to observe the various steps that packages go through, from the moment that the sources are released until when the programs are delivered to the users. However, since the number of packages to be installed or updated constantly grew, as did the number of supported platforms, it soon became clear that the procedure had to be automated and that from the very start scalability and version control had to be taken into account.

Thus, based on our observations during the manual installation phase, we adopted for ASIS a "Software Processing Model" represented as a state/transition diagram in Figure 4. The model defines the following states:

- *Unknown*: the package is not present in the data base.
- *RemotePackedSources*: the system knows from which remote site it can get the source and where to store it.
- *LocalPackedSources*: the sources as distributed by the author(s) are stored in the repository.
- *ExpandedSources*: ASCII sources are available in the repository.
- *ConfiguredSources*: sources are ready to be compiled on each platform, generally after generating Makefile(s) and configuration file(s).
- *Executable*: the executable files were built.
- *Tested*: the tests provided with the package distribution kit were run successfully.
- *Installed*: the executable files, their execution environment and all available documentation were copied into a single scratch directory and checked for consistency.
- *UnderCertification*: the ready-to-use files and the full documentation were copied into the ASIS reference repository, where they are available for "testers".

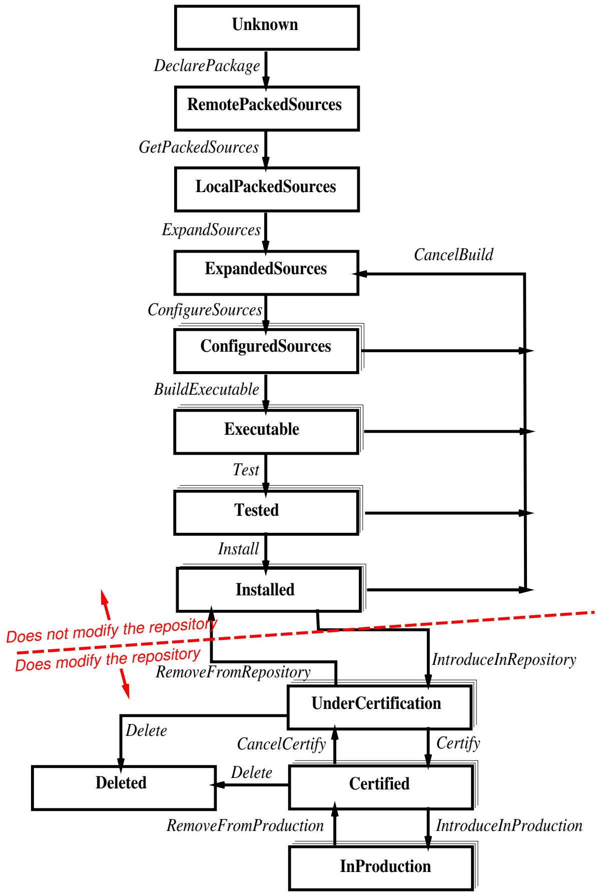

Figure 4: The "Software Processing Model"

- *Certified*: the software is available to all users after validation by the "testers", the manager of the local copy has updated the local repository.
- *InProduction*: the software is included in the default user environment and is officially supported.
- *Deleted*: the product is removed from the repository.

All states from *ConfiguredSources* to *InProduction*, are specific to each supported platform.

These states together with all possible state transitions (shown in italic) are seen in Figure 4.

The state transitions can be grouped according to their effect on the repository and can be

- *passive*, when they leave the reference repository unchanged. This is the case for all transitions above the dashed line in Figure 4, i.e., between the states *Unknown* and *Installed*.
- *active*, when they modify the reference copy of ASIS. These are transitions which cross or are situated below the dashed line of Figure 4, i.e., all transitions between the states *Installed* and *InProduction*/*Deleted*.

It is seen that passive transitions can be executed completely independently on different products while active transitions act on the unique reference repository. Therefore the execution of active transitions must be strictly controlled.

The result of the full sequence of passive transitions for a particular product is an object containing the full package, its environment and its documentation for all supported platforms in ''ASIS package format'' (see Figures 1 and 2). The transition *IntroduceInRepository* moves the object into the repository. All other active transitions manipulate the object.

It is also possible for a product which is distributed in executable form to be brought directly into the repository without going through any of the passive transitions. This is acceptable provided that the ASIS object to be introduced into the repository has the right structure. For instance, such procedures are applied for products like *MISC/xemacs* or *X11.R5/netscape*. Often, but not always, the process to build the right structure for such binary distributed packages can be automated. The *Install* transition creates the right directories and copies the given binaries into the right position; all other passive transitions are no-operation procedures.

The Automated Generation Environment

A product maintainer can perform all *passive* transitions (see above) in an automated way using a tool called *happi*, the ''Heterogeneous Automated Product Processor and Installer''. *happi* can execute a single transition, change the state of a product, or display the current state together with associated internal information. *tkhappi* is a TK-based user-friendly interface of *happi*, which improves also the readability of its output (see Figure 5).

General tasks, like getting the original distribution of a package or expanding a tar file, are performed on a server computer. For each supported platform, there exists a dedicated computer, where executables are built. The computers service these generation tasks and are called in ASIS jargon the ''reference machines''. *happi* is responsible for distributing the operational tasks for the software processing to these reference machines. *happi* performs the authorisation and the control of the remote processes. It is written in such a way that the tasks executed on the remote computers just execute *happi*. When *happi* is run, it first detects which tasks have to be run and where. Then, it launches as many concurrent tasks as possible. General tasks are mainly sequential, but O.S.-specific generations are executed simultaneously on all the reference machines. *happi* verifies the status returned by each task and determines the next actions to undertake. It also updates the data base describing the new state reached on each architecture.

happi collects logging information and builds log files and a history for each product. Logs and history can be most conveniently consulted via *tkhappi*, but any other text viewer can also be used.

happi only deals directly with passive transitions, hence several *happi* tasks acting on different products can run at the same time.

Task communications are handled mainly using *expect* [8], while all other functionalities are programmed in *perl* [15].

The ''product maintainer'' must provide *happi* with a ''configuration file'', written in *perl*, where all passive transitions are described. Each transition is represented by a *perl* function whose name is identical to the name of the transition (Figure 4). In addition,

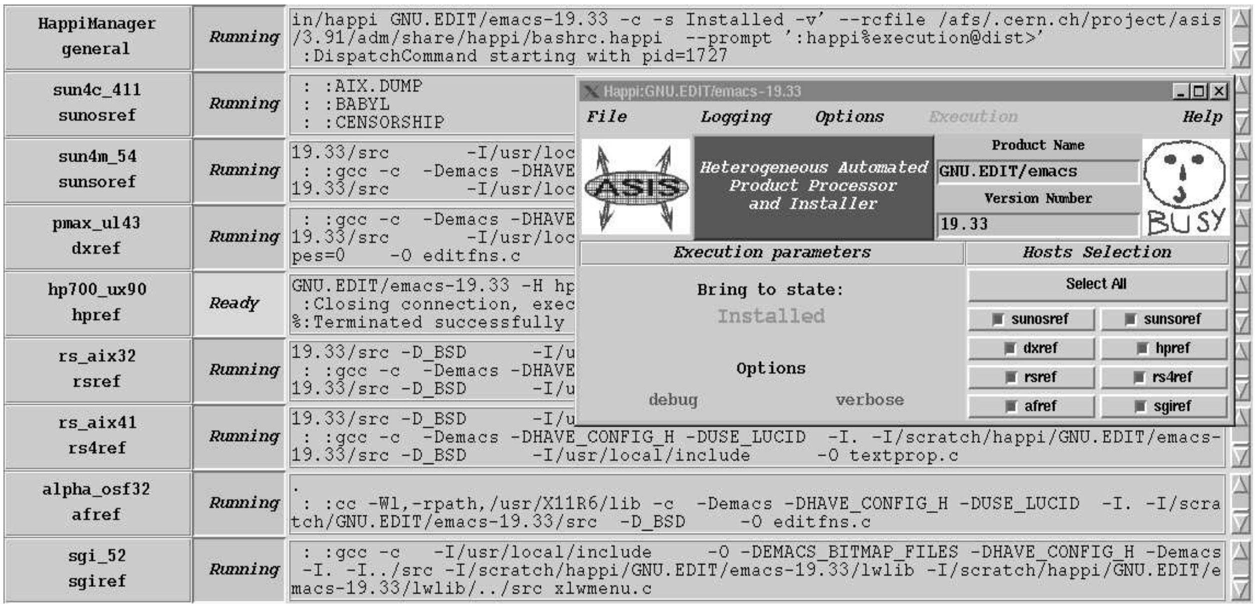

Figure 5: *tkhappi*: the tk based interface to *happi*

other parameters are represented by *perl* variables and can be assigned a value, overwriting the defaults if needed. Each function contains the sequence of operations to be executed in order to perform a successful state transition. In most cases, this file is very simple, since many packages are well designed and easy to build. Moreover, *happi* comes with an extensive library of functions for the most-frequently performed operations. Figure 6 shows a typical configuration file. Figure 7 shows a more complex case. In the latter, some work had to be devoted to correctly separate the installation of the *specific* and *share*d files.

happi is delivered with libraries, called packages in the *perl* jargon, that cover most of the repetitive operations:

- the standard *happi* library, containing dedicated *happi* specific operations;
- the *GNU library*, specific to the GNU products;
- the *documentation generation library*, to build Postscript and HTML files from the package documentation written in LaTeX or *texinfo*;
- the *file manipulation library*, operates on files,
- the *synchronisation library*, to synchronise the tasks on the different reference machines;
- the *O.S. knowledge base library*, to find optional libraries, include files, optimal compilation/link flags, etc. for the different supported platforms;
- the *ASIS data base query library*, to get information on other products in ASIS.

Examples of use of *happi* packages are shown in Figures 6 and 7. Some of the subroutines are described below.

- *Split* runs *GNU tar* to expand the sources. It uses standard locations both for the input, the compressed tar file and for the output, the ASCII sources.
- *Run* executes a shell command. It performs logging and error detection, analysing both the return value of the command and its output.
- *Link* runs the X11 *lndir* command on the local reference machine in order to create links to the centrally stored sources. It is the standard way to get access to the sources on the different reference machines.
- *Make* runs *GNU make*.
- *ModifyFile* edits a file.
- *DefaultExecDir* builds the user execution directory name for the given keyword.
- *DefaultInstallDir* returns the installation directory name according to the “ASIS package format” for the given keyword.
- *Sharelink* creates a link pointing from the *specific* part of a package to the corresponding *share*d area.
- *gnu::MakePSandHTML* builds Postscript files from existing *dvi* files and HTML files from the corresponding *texinfo* files. This is targeted to *GNU* software: it assumes that documentation is written in the *texinfo* format. In fact, the *gnu::MakePSandHTML* function is just another name for *doc::MakePSandHTMLFromTeXInfo* which belongs to the documentation package.
- *file::copy* copies a file.
- *file::wipeout* deletes recursively directories.

In Figures 6 and 7, also some *happi*-specific variables were used:

- *$final_automatic_state*: the default state to which *happi* brings a product if not specified otherwise;
- *$packedfile*: the relative position of the product file in the *LocalPackedSources* area; this variable is automatically assigned to a standard value at *happi* startup; it may be overwritten by the product maintainer if necessary;
- *$sys*: the standard platform name (read-only);
- *$share*: a boolean which is true on the reference machine dedicated to do the work for *share*, for instance to generate Postscript and HTML documentation.

Other variables are available to deal with complex generation and installation procedures.

As soon as the product maintainer has provided the system with the *happi* configuration file, the system can run completely automatically. The command *mirror* [2] is run periodically to interrogate all distribution sites about their contents and to retrieve new sources for the packages supported in ASIS. *mirror* stores the retrieved files in the ASIS repository. Similarly, local authors can drop new distribution kits in the ASIS *LocalPackedSources* area.

Independently from *mirror*, *ASISdnv* (ASIS Detect New Versions) is run regularly (or executed manually, if needed). It scans *LocalPackedSources* and detects new versions of packages supported in ASIS. *ASISdnv* automatically launches the execution of *happi* which brings the product in the state specified by the ASIS parameter *$final_automatic_state* set in the *perl* configuration file. Thereafter, *ASISdnv* sends a report to the product maintainer describing the result of the execution of *happi*. The product maintainer can read the *happi* execution logs with *tkhappi*. Even when the execution was diagnosed to be successful, the generation logs should nevertheless be verified to ensure that the process was actually correct since the available expert system for log analysis is still rudimentary. When the generation appears to have failed, *tkhappi* is used to help fix the problems.

The ASIS Transaction Model

Even though the full Software Processing Model describes the actual operations acting on a package satisfactorily, active transitions cannot be performed

[2] *mirror* was written by Lee McLoughlin (lmjm@doc.ic.ac.uk) and is a package written in Perl that uses the ftp protocol to duplicate a directory hierarchy between the machine it is run on and a remote host.

```
## -*- mode:perl -*-
#       File for X11.R5/tgif
$final_automatic_state = 'Installed';
# The tar.gz file is not in the standard pub/X11.
# It is in a subdirectory of pub/X11, given just below.
# Mirroring  the whole ftp.x.org:/contrib directory
# gives us no control on the contrib directory tree
# structure.
$packedfile=
  "contrib/applications/$name/$name-$version.tar.gz";
sub ExpandSources {
# Run tar to extract the sources
 &Split;
}
sub ConfigureSources {
# Link to centrally stored sources
  &Link;
# Generate makefiles
  &Run('xmkmf');
  &Make('Makefiles');
}
sub BuildExecutable {
# The author defines TGIFDIR in a non-standard
# way. Below is the ASIS choice.
 &Make('TGIFDIR='.&DefaultExecDir('USRLIBDIR').'/tgif');
}
sub Install {
# /usr/local/lib/X11/tgif only contains shared
# data, thus we make a link from the specific
# /usr/local/lib/X11/tgif to the shared one.
 $tgifdir=&DefaultInstallDir('USRLIBDIR').'/tgif';
 &Sharelink($tgifdir);
# Install in 'Installed' area not in /usr/local/...
 &Make
  ('BINDIR='.&DefaultInstallDir('BINDIR').\
    "TGIFDIR=$tgifdir MKDIRHIER=mkdirhier install");
# Install man pages only on the machine
# dedicated to share.
 if ($share) {
  &Make
   ('MANDIR='.&DefaultInstallDir('MANDIR')\
     MKDIRHIER=mkdirhier install.man");
  }
}
```

Figure 6: The *X11.R5/tgif* configuration for *happi*. Comments were added for each step to aid comprehension.

```
## -*- mode:perl -*-
#          File for GNU.EDIT/emacs
$final_automatic_state = 'Installed';
sub ExpandSources {
  &Split;
}
sub ConfigureSources {
# On alpha's, use the OSF X11 libraries as no MIT X11 release 5. Better use
# cc then.
  &Link;
  $cfg_cmd='./configure --prefix='.&DefaultExecDir('prefix').
                                                  '--with-x-toolkit=yes';
  $cfg_cmd .= ' --with-gcc=no' if ($sys =~/^alpha_osf/);
  &Run("$cfg_cmd");
}
sub BuildExecutable {
# Overwrite CFLAGS for performance reasons and change some bitmaps directories
  &Make('CFLAGS=-O bitmapdir='.&DefaultExecDir('bitmaps').':'.
                                                &DefaultExecDir('pixmaps'));
}
sub Test {
  &Make ('check');
}
# Load GNU software specific library
require 'happi/GNUUtils.pl';
sub Install {
# Modify Makefile once for installing only the files specific to each platform;
# thereafter do the install in the "Installed" area
  &ModifyFile('Makefile');
  &Make("prefix=&DefaultInstallDir('prefix') install");
# Delete unnecessary created specific directories
  &file::wipeout(&DefaultInstallDir('datadir',$sys));
  &file::wipeout(&DefaultInstallDir('sharedstatedir'));
  &file::wipeout(&DefaultInstallDir('mandir'));
# create a link from specific to shared area
  &Sharelink("$dir{Installed}/$sys/usr.local/info",'F');
  if ($share) {
    *changed = *share_changed; &ModifyFile('Makefile');
    &Make("prefix=$dir{Installed}/share/usr.local install-arch-indep");
# Delete unnecessary created shared directories
    &file::wipeout(&DefaultInstallDir('libexecdir','share'));
    &file::wipeout(&DefaultInstallDir('bindir','share'));
    &file::unlink(&DefaultInstallDir('infodir').'/dir');
    &Makedir(&DefaultInstallDir('sharedstatedir').'/$name');
    &file::rmdir(&DefaultInstallDir('sharedstatedir').'/$name/lock');
    &file::symlink('/tmp',&DefaultInstallDir('sharedstatedir').'/$name/lock','O');
  }
# Install documentation (on share of course !)
#  - Create the dvi files (nice we are with GNU)
#  - Generate ps, HTML and Install in the 'doc' standard area
  if ($share) {
    &Make('dvi');
    &gnu::MakePSandHTML();
  }
}
#  Subs to modify makefiles to separate share and the rest
sub changed {
```

```
   s#install-arch-indep## if (/^install:/);
 }
 sub share_changed {
   s#${srcdir}#$dir{ExpandedSources}#g if (/^COPYDIR\s*=/);
 }
```

Figure 7: The *GNU.EDIT/emacs* configuration file for *happi*

without introducing some further control. Concurrency cannot be permitted if the system is to be kept simple and transitions are better performed in a coordinated manner.

For instance, when a product maintainer has to change the version of a product that is *InProduction*, two state transitions must take place: (1) a version is first *Remove*d*FromProduction*, and (2) a new one is *Introduce*d*InProduction*. Both these operations should be executed in the right order and both must be successful in order to ensure that the user environment is not modified drastically by leaving a product in an intermediate state, which prevents its further use after the next update. Moreover, if a product was only removed from production and no other version was introduced, users will no longer be able to access it.

It can happen that different products claim to be the owner of the same file. This can occur by bad coordination or wrong reorganisation of packages. If the repository has to remain coherent, only one modification can happen at any given time in order to guarantee that the introduction of a package does not destroy the execution environment of another. Thus, sequencing the different repository modifications becomes a necessity.

Therefore, it was necessary to introduce a transaction system for ASIS to ensure sequentiality and to introduce atomicity. Each transaction consists of a sequence of operations (transitions and/or specific actions). Each transaction is either performed completely and successfully or not at all. Transactions are ordered and performed one at a time.

A product maintainer is thus guaranteed that the package being manipulated does not disappear from the *InProduction* area when actually trying to update the version *InProduction*, even if a problem occurs during the execution of one of the state transitions. Transactions are themselves executed sequentially which simplifies the control of the file name space and increases the file system integrity.

The ASIS Transaction System

The system is based on the cooperation between the ASIS Central Transaction Manager (*ASISctm*) and the Local Copy Managers (*ASISlcm*).

The Central Transaction Manager is responsible for checking, committing and distributing the transitions to be executed.

The Local Copy Managers are responsible for maintaining their own repository and, possibly, checking the feasibility of a transaction in the local environment.

A local copy is said to be *compulsory*, if any modification must be applicable to that particular copy as soon as this modification is valid on the reference copy. A compulsory copy can be updated from the reference copy at any time. Other copies are called *independent*.

A transaction is submitted by the product maintainer to the ASIS Central Transaction Manager which accepts or refuses it. The *ASISctm* program verifies that

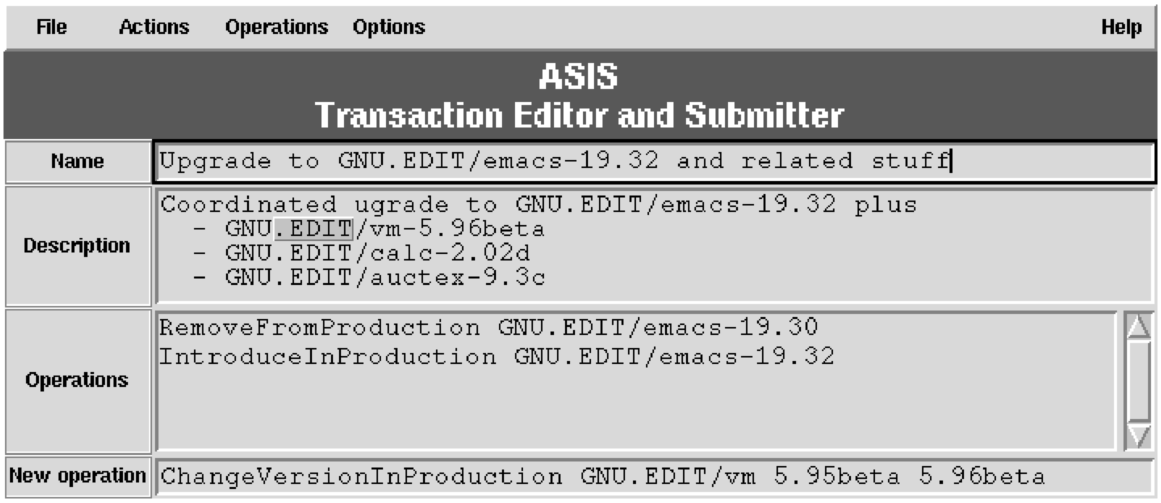

Figure 8: The Transaction Editor and Submitter (Prototype).

- the submitter is duly authorised for each operation and product of the transaction;
- the product does not conflict with other products in the file name space;
- the needed resources are available (disk space, access, etc.) for the reference and each *compulsory* copy.

When all checks are successful, the transaction is committed. The product maintainer knows that the modification will be performed successfully on the reference repository and that the next update of the compulsory copies will also be successful and will include the submitted changes. Updates of *independent* copies are not guaranteed.

The update of the reference copy will happen automatically without further intervention by the product maintainer. It is performed asynchronously when the *ASISlcm* (the ASIS Local Copy Manager) of the reference copy is run. The frequency at which *ASISlcm* is run is determined by the ASIS Manager.

The Central Transaction Manager maintains:
- the ordered list of committed transactions;
- the list of dependencies between transactions;
- the lists of compulsory copies and their attributes;
- the authorisation data base;
- the *virtual state* of the repository, i.e., the image of the repository with all committed transactions performed.

Product maintainers can use *ASIStes* (the ASIS Transaction Editor and Submitter) to ease manipulating and submitting transactions, i.e., the changes to the repository needed to maintain the packages. *tktes* is the corresponding TK-based Transaction Editor and Submitter (Figure 9).

tktes enables product maintainers to:
- build the list of operations to perform in a single transaction, submit this transaction to the Central Transaction Manager and, finally, commit it if correct;
- list the committed transactions;
- view the *virtual state* of the repository, i.e., the state of the repository (file name space, disk space, etc.) after execution of all committed transactions.

The Replication Model and Software

Presently, the ASIS repository is being mirrored by many HEP sites. Even though mirroring is optimised by only transferring modified files (as described in [1]), a complete traversal of the whole ASIS directory tree is necessary. The operational cost is thus proportional to the size of the repository, i.e., several thousands of files.

If all transactions, i.e., repository modifications performed on the master copy since the last local update, are known, it is unnecessary to scan all ASIS files. Some transactions like *IntroduceInProduction* can be performed entirely locally as they just move files from the local *Certified* to the local *InProduction* areas. Only modified and newly created files are to be transferred through the network. No check is done on unchanged data. The time to update the local copy then becomes proportional to the size of the changes made to the ASIS repository, which is more acceptable.

The ASIS Local Copy Manager (*ASISlcm*) updates the content of the local repository from a designated *master* copy. It obtains the "performed transactions list" from the *master* and determines, since the time when it was last run, the sequence of transactions

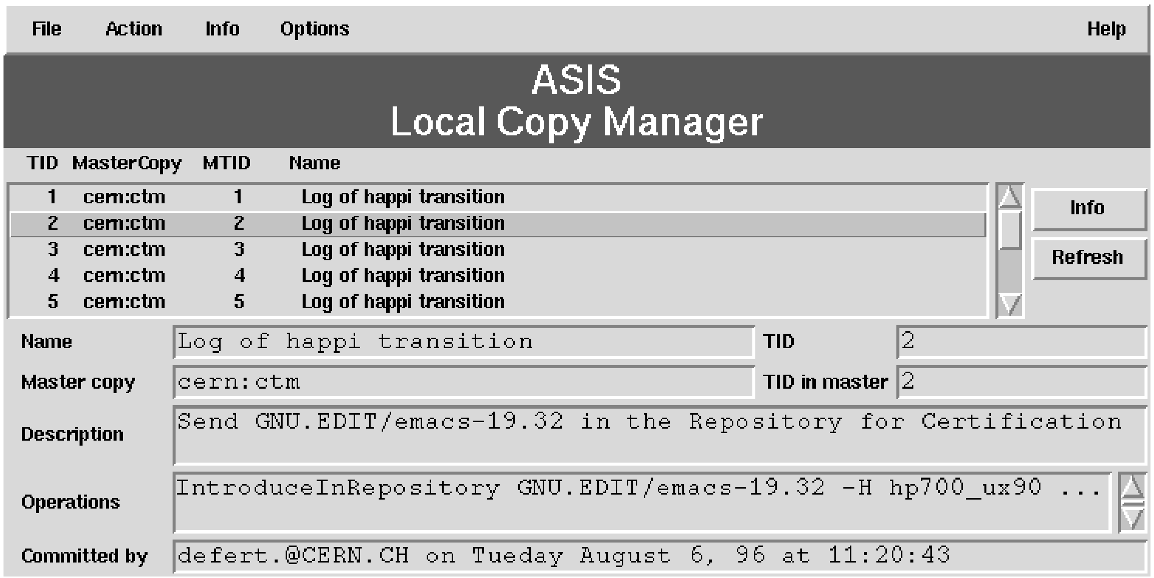

Figure 9: The Local Copy Manager (Prototype).

to be executed to update the local repository. It verifies the feasibility of the transactions, performs the necessary operations and maintains the list of already executed transactions.

ASISlcm can be customised to control the following:

- the frequency and the scheduling of the updates;
- the location of the *master* copy, i.e., the replica from where to take the transactions and data;
- a runtime filter to apply before performing transactions, e.g., rejecting, delaying, or modifying some transactions according to local rules, like ignoring some families, products or platforms.

For an *independent* copy, the choice of the master and the update frequency is completely open. However, when *ASISlcm* is run on a *compulsory* copy, the master must be the "Reference Copy". In this case, the transactions list is read from the "Reference Copy" and new files are copied from there also. *ASISlcm* can be used to update the "Reference Copy". Then, no master is used and new files are taken from the *Installed* scratch area with transactions read from the list of "Committed Transactions".

The TK based *tklcm* allows a local copy administrator to view and manipulate the transactions list (see Figure 9).

The Status of the Project

Functionality

ASISwsm is now part of the standard CERN installation procedure of all UNIX systems and is used on several hundreds of workstations.

There are three ASIS local copies at CERN, to serve NFS, AFS and DFS users.

happi and *tkhappi* are used to maintain most public domain and CERN packages. At present, when a new source distribution kit is released at a distribution site, the full generation is not started automatically. The *ASISdnv* program does not launch *happi*: it just scans the *LocalPackedSources* area of the ASIS repository to find new versions of products and builds a "to-do" list for the product maintainers. Additionally, the product maintainer are informed by an electronic mail so that they can run *tkhappi*. All transitions in the "Software Processing Model" were implemented using idempotent functions, except *IntroduceInProduction* and *RemoveFromProduction* which could only be built as atomic operations secured by a check-point mechanism.

Beta versions of *ASISctm*, *ASIStes* and *tktes* run in the ASIS developers environment, but are not yet delivered to other product maintainers. The latter use a version of *tkhappi* which inserts an entry in the transaction list for each active transition performed, see Figure 9.

ASISlcm and *tklcm*, also in beta test, are used to manage pilot replicas for small clusters at CERN. By the end of 1996, the University of Barcelona should use *ASISlcm* to replicate the ASIS environment. Presently, ASIS is mirrored in many HEP sites like the Rutherford Appleton Laboratory (U.K.), CASPUR (INFN, Italy), etc. These sites, however, still use full mirroring. *ASISlcm* will be gradually introduced as soon as it has been thoroughly tested and validated.

Usage

At CERN, there are presently around seven hundred Sun's and HP's workstations each, two hundred IBM's, three hundred and fifty DEC's and fifty SGI's, giving a total of about two thousand UNIX workstations and servers. Some seventeen hundred X-terminals are also registered. More than 5000 users have a UNIX login account. Most of the workstations access the Application Software Installation Server and most UNIX users use software contained in ASIS for their main development tasks.

Around six hundred public domain packages are available for eight different platforms. Ten per cent of these products have more than one version *Certified* or *InProduction* in ASIS.

The size of the CERN ASIS local copy serving the NFS clients is about eleven gigabytes, forty per cent of which is devoted to the CERN Program Library. Similar copies are available for AFS and DFS.

The "Ecole Polytechnique Fédérale de Lausanne" (EPFL) now uses the ASIS tools suite to install public domain software and to manage the versions at their own site. They participated a lot to the debugging phase, mainly to resolve site-dependency problems. Recently, they have taken on the responsibility for the development and maintenance of the "User Interface" part, mainly based on TK. The EPFL group is getting more and more involved in the project and there is no doubt that future versions of ASIS will include features specified and designed by them.

Availability

The tools suite is in beta test, and during this period the sources are available from CERN. The CERN management has still to determine the policy for distributing production versions. All HEP centres are entitled to receive the software free of charge. Other academic institutes should hopefully get it also. In principle, others should contact CERN for more detail.

Other Information

The full documentation of ASIS is available at the URL: *http://wwwcn.cern.ch/dci/asis*. This gives access to the "ASIS Users and Reference Guide" [5], the "ASIS Product Maintainer's Guide" [4], and various related publications, plus the full set of documentation distributed with the packages included in the ASIS repository.

The full tools suite can be obtained from the URL *ftp://asisftp.cern.ch/dist/ASIS/ASIS.tar.gz*. The *tar* file contains a *README* file which serves as a very first prototype of the "ASIS Installation Guide". Currently, the distribution contains the version which is in production at CERN and does not contain the transaction system. The distribution will be regularly updated with bug corrections and improvements. The ASIS tools suite version 4 will be distributed as soon as it has been certified and the distribution policy defined

Future Developments

Software interdependencies are at present not handled. We are studying possible solutions as proposed in Fermi-Lab's UPS/UPD [2], in Depot-Lite [11], or in the Debian Lignux Distribution Project [7] which could be used as a basis for ASIS product dependency management.

Author Information

E. Fernandez Dominguez has completed his End of Degree Project at CERN in order to get a Telecommunications Engineer Degree at the Technical University of Madrid (UPM). He now works for "Telefonica Sistemas" in Madrid; contact him at <*eusebio@ete.tid.es*>.

O. Le Moigne has been working at CERN since he received his degree of Telecommunication Engineer of the ENSTB School (Brest, France) in 1995. He can be contacted at <*olivier.le_moigne@cern.ch*>.

A. Peyrat received his degree of Computer Science Engineer of the ENSIMAG Engineering School of Grenoble (France) in 1993. He worked at CERN between 1993 and 1995. He can presently be contacted at <*apeyrat@email.teaser.fr*>.

I. Reguero received the "Ingeniero Superior de Telecomunicacion" degree from the Polytechnic University of Madrid in 1985. He studied in parallel two curriculum areas: "Data Communication / Data Transmission" and "Computer Science/Data Transmission". His graduation thesis was "Specification and Implementation of Interpersonal Messaging Protocol, Following the X.420 Standard". He worked in 1986 and 1987 as system engineer for the IBM IN network. He worked in 1988 for "Banco De Santander" as communications systems support of a network of more than 2000 offices. He is currently a systems engineer at CERN working on large systems performance and communications, network backup systems, Unix system software installation and consultancy for Unix system administrators. He can be reached at <*ignacio.reguero@cern.ch*>.

M. Goossens. After obtaining a MSc in physics in 1972 and PhD in physics in 1978 from the Free University of Brussels (Belgium) he joined CERN at the beginning of 1979. After working two years as a Fellow on a muon experiment, he moved to software support working on memory management problems. Soon he came to realize the importance of good and up-to-date documentation, and thus became gradually more involved in the field of text processing and documentation. Over the years he worked with several typesetting systems, more in particular LaTeX and Framemaker, Postscript, SGML and HTML. Recently his main interest is the development of tools to facilitate using sources of documents for generating online hypertext documentation, like WWW and Acrobat. He is the co-author of a book *The LaTeX Companion* (Addison-Wesley 1994). In 1994, he was elected President of the French-speaking TeX Users' Group GUTenberg, and since 1995 also of the International TeX Users' Group TUG. He can be mailed at <*michel.goossens@cern.ch*>.

Ph. Defert received an MSc in Computer Science from the Free University of Brussels in 1975 and a PhD in Applied Mathematics in 1983 from The University of Namur (Belgium). He worked at the European Southern Observatory between 1984 and 1987 in Space Telescope European Coordinating Facilities in Garching bei Muenchen (Germany) as ST Data Analysis Scientist. After having moved to CERN to participate to the design and implementation of the control system of the Large Electron Positron Collider, he is now a member of the Computing and Networks Division of CERN specifically in the Distributed Computing Infrastructure Group. In this environment, he manages the ASIS Project, subject of this paper. His present research interests include Software Engineering. Contact him at <*philippe.defert@cern.ch*>.

References

[1] Paul Anderson. Managing program binaries in a heterogeneous unix network. In *LISA V Proceedings*, pages 1-9. Usenix, 1991.

[2] William Bliss, Jonathan Streets, Lourdu Udumula, and Margaret Votava. *UNIX Product Distribution, User's Guide*. Fermilab, FNAL, 1992.

[3] Michel Dagenais, Stephane Boucher, Robert Gerin-Lajoie, Pierre Laplante, and Pierre Mailhot. Lude, a distributed software library. In *LISA VII Proceedings*, pages 25-32. Usenix, 1993.

[4] Philippe Defert, Alain Peyrat, and Eusebio Fernandez Dominguez. *ASIS: Product Maintainer's Guide*. CERN, the European Laboratory for Particle Physics, 0.91 edition, 1995.

[5] Philippe Defert, Alain Peyrat, and Ignacio Reguero. *ASIS: Users and Reference Guide*. CERN, the European Laboratory for Particle Physics, 3.00 edition, 1994.

[6] Ph. Defert et al. Automated management of an heterogeneous distributed production environment. In *First Conference on Freely Redistributable Software*, pages 1-8, 1996.

[7] Ray Dassen et al. *Debian GNU/Linux*. Debian, http://www.debian.org, 1996.

[8] Don Libes. *Exploring Expect*. O'Reilly and Associates, Inc., 1995.

[9] Kenneth Manheimer, Barry Warsaw, Stephen Clark, and Walter Rowe. The depot: A framework for sharing software installation across organizational and unix platform boundaries. In *LISA IV Proceedings.*, pages 37-46. Usenix, 1990.

[10] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, 1994.

[11] John P. Rouillard and Richsrd B. Martin. Depot-lite: A mechanism for managing software. In *LISA VIII Proceedings*, pages 83-91, 1994.

[12] John Sellens. Software maintenance in a campus environment: The xhier approach. In *LISA V Proceedings*, pages 21-44. Usenix,. 1991.

[13] Stephen Shafer and Mary Thompson. The sup software upgrade protocol. Technical report, Carnegie Mellon University, School of Computer Science, 1988.

[14] Rainer Toebbicke and Philippe Defert. Recommended standard for unix workstation environment setup. Technical report, CERN, the European Laboratory for Particle Physics, 1990.

[15] Larry Wall and Randal Schwartz. *Programming perl*. O'Reilly and Associates, Inc., 1991.

[16] Walter C Wong. Local Disk Depot – Customizing the Software Environment. In *LISA VII Proceedings*, pages 51-56. Usenix, 1993.