

CERN LIBRARIES, GENEVA



CERN-CN-96-001

## Automated Management of an Heterogeneous Distributed Production Environment

Ph. Defert, E. Fernandez Dominguez, M. Goossens, A. Peyrat, I. Reguero

CERN, European Laboratory for Particle Physics. Geneva. Switzerland

Presented at the "Conference on Freely Redistributable Software", Boston, February 2-5 1996

To be published in the proceedings of the "Conference on Freely Redistributable Software"

## Abstract

A production environment is described where users can get access to hundreds of packages from a central repository. Users merely have to introduce a small change in their system setup to be able to run all available applications. On the other hand, people responsible for maintaining these products are provided with tools to automate their generation and to manage multiple versions. A method is also proposed to allow the replication of this environment on a remote site in an efficient way.

## 1 Introduction

The standard software delivered with UNIX systems is not always sufficient. Commercial software is often very specific and expensive. On the other hand, there exist a lot of useful freely available UNIX packages such as editors, window managers, document formatting utilities, drawing tools, etc., which are distributed under source form.

ASIS (the Application Software Installation Server) provides an environment that offers a large number of freely available UNIX applications which are useful for day-to-day work. It frees end-users from having to get, configure, generate, install and manage the software. It also offers tools to product maintainers to generate and install those packages.

ASIS has three logical entities.

- The **repository** is the central part of the project. It contains the applications in an executable format. The included packages cover a large variety of domains: High Energy Physics (HEP) data analysis software, the CERN Program Library, most GNU packages, the latest  $\text{T}_\text{E}\text{X}/\text{L}_\text{A}\text{T}_\text{E}\text{X}$  setup, MIT X-windows contributions, TCL/TK based software and many other tools written by the UNIX community. Presently, the system allows users access to more than 400 products for seven different platforms on SUN, DEC, IBM, SGI and HP computers. Soon nine platforms will be supported. A data base describing all packages with version control information is maintained in the repository.
- The **client** interface is the more visible part. In order to gain access to software on ASIS from a computer, the only change in the system setup is to schedule the execution of an **ASISUpdate** script each time the repository is updated.
- A tools suite helps **product maintainers** keep their packages up-to-date. The provided tools automate the generation and installation process

with quality and reliability receiving maximum attention.

## 2 Organisation of the Repository

The main aim of the ASIS project is to facilitate user access to supported software packages and to provide maintainers with installation tools. A lot of care was devoted to the design of the structure of the repository to make these tools simple and maintainable.

At CERN, the ASIS repository is available on several distributed file systems: AFS, the Andrew File System, NFS, the SUN Networked File System and more recently DFS, the OSF Distributed File System.

Within the context of a heterogeneous environment i.e. a combination of more than one architecture/operating system (O.S.), it is useful to partition software into two parts:

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

Each **specific** part contains the necessary references to the **shared** part. In order to avoid unwanted site dependencies, all symbolic links are relative.

In the repository, multiple versions of a package can coexist, but generally, only one is **InProduction**, i.e. being formally supported.

A version of a product which is not **InProduction** is completely stored in a given directory that contains all **specific** and **shared** data such as in the NIST Depot [?]. The naming scheme is detailed in Figure ??, which shows the directory structure used.

- `<family>/<name>-<version>` product name
- `<platform>` standard name
- directory in which it should be referenced when executed in the user environment, in our example `usr.local` which represents `/usr/local`
- subdirectories needed by the particular package.

Conversely, the **InProduction** products for a platform are all stored in the same directory tree structure (see Figure ??), i.e. directly below the ASIS root directory.

- `<platform>` standard name
- Execution environment directory (`usr.local`)

The correspondence between files and products is kept in the ASIS data base. Only one copy of each product is kept in the repository. Thus, when a product is sent

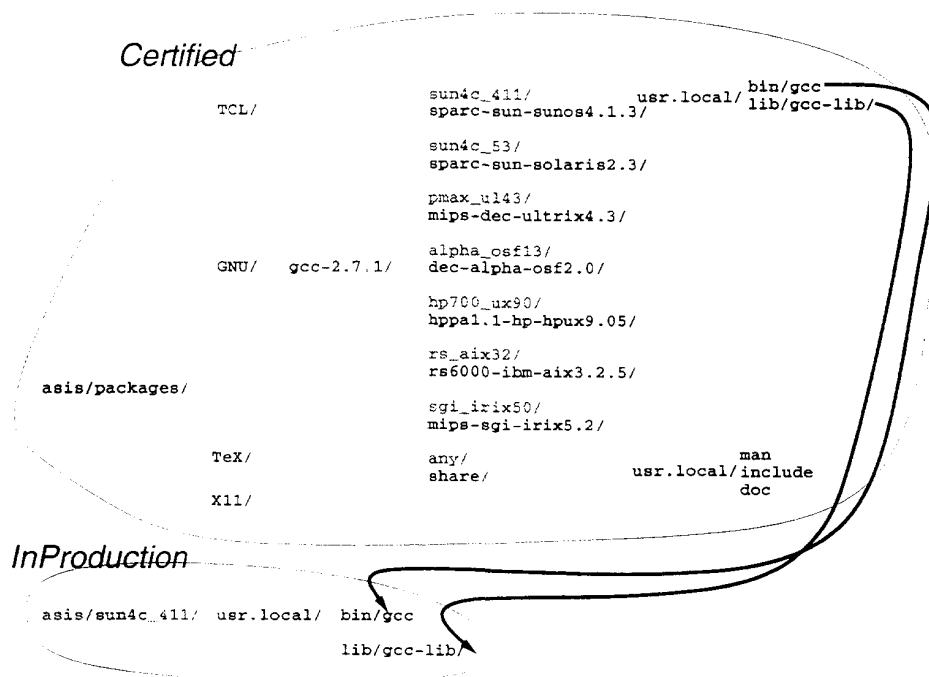


Figure 1: The Directory Tree Structure of the **Certified** and **InProduction** Areas

to **InProduction**, its files are moved from the **Certified** area to the **InProduction** area (please refer to section ?? for the formal definitions of the states).

### 3 Users Access to ASIS

The tool **ASISInfo** and its TK based incarnation **ASISBrowse** give users a short description of each package. They also detail differences between versions and indicate the current state of the products in the repository.

At CERN, as well as in most HEP centres, the UNIX support teams have decided to install all public software in `/usr/local` (see [?]) and ASIS was chosen to maintain that directory.

The **ASISUpdate 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 distributed file systems, for instance at CERN: `/afs/cern.ch/asis`, `/nfs/cern.ch/asis` or `:/asis`. The script does not modify user files but reports possible file name conflicts between ASIS and these files. The person responsible of a package can force a local copy for some files of the product and some other file operations: this is used mainly for the installation of login shells and

some set-uid commands. These constraints are treated by **ASISUpdate** also.

A local system administrator can customise the behaviour of **ASISUpdate** in such a way that

- a product is ignored
- a product is copied locally instead of accessed via links;
- a product is accessed via links only even if the product maintainer decided to do a local copy;
- the version installed on a computer does not have to be the one **InProduction** but can be any version or the latest **Certified** one;
- user files can be declared to be overwrite-able or not.

A TK based configuration editor is part of the tools. Figure ?? shows the product configuration window.

The ASIS repository evolves rapidly as products are continuously updated, bugs corrected and new products installed. Thus, **ASISUpdate** should be run at the same frequency as the repository is updated.

Most users take the default environment i.e. **InProduction**. As all **InProduction** files reside in the same "file system" or "volume" and "directory tree", **ASISUpdate** can optimise the number of links to be created in the execution directory. Similarly, to improve performance, replicas are created for the most

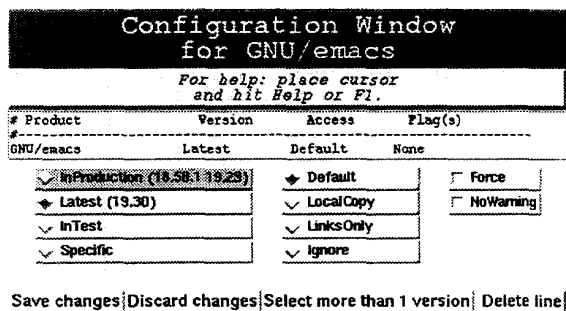


Figure 2: The “product configuration” window

frequently accessed file systems i.e. the more common platforms.

## 4 The Product Maintainer’s Environment

### 4.1 The Software Processing Model

#### 4.1.1 Description

Installing manually for over a year public domain packages, 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. Based on our observations, we decided to introduce a “Software Processing Model” whose state/transition diagram is shown in Figure ??.

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, generally after Makefile(s) and configuration file(s) were generated.
- **Executable**: the executable files were built.
- **Tested**: the tests provided with the sources have been run successfully.
- **Installed**: the executable files, their execution environment and the full available documentation have been transferred into the repository and are ready to be used by the “testers”.
- **Certified**: the software is available to all users after having been validated by the “testers”.

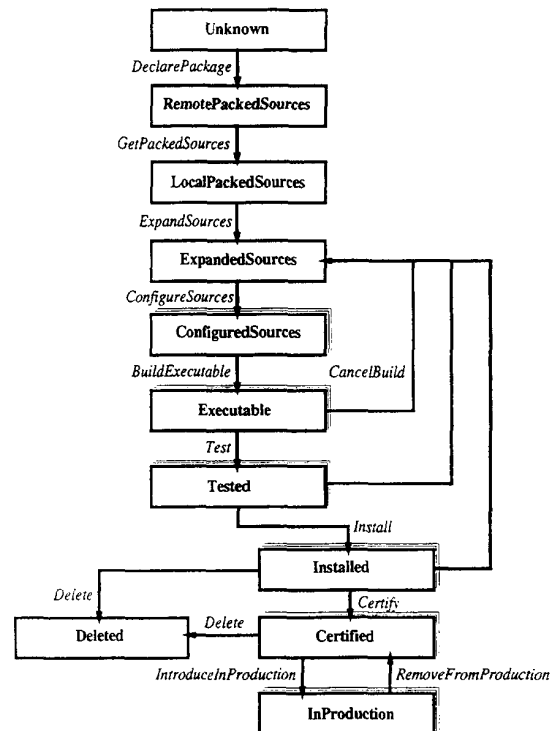


Figure 3: The “Software Processing Model”

- **InProduction**: the software is included in the default environment of the users and officially supported.
- **Deleted**: product is removed from repository.

The states **ConfiguredSources** to **InProduction**, are split in as many sub-states as there are supported platforms.

These states together with all possible state transitions are shown in Figure ??.

#### 4.1.2 The Automated Environment

A product maintainer can perform all operations in the ASIS “Software Processing Model” in an automated way using a tool called **happi** which stands for “Heterogeneous Automated Product Processor and Installer”. **happi** can execute a single transition, bring a product from its current state into another or display the current state of a product and its associated internal information. **tkhappi** is an X-windows TK based interface that increases the usability of **happi** and the readability of its output (see Figure ??). Each supported platform is associated with a dedicated computer, where executables are built, and **happi** can distribute the building tasks to these reference machines. **happi** was written using **perl** ([?]) and **expect** ([?]).

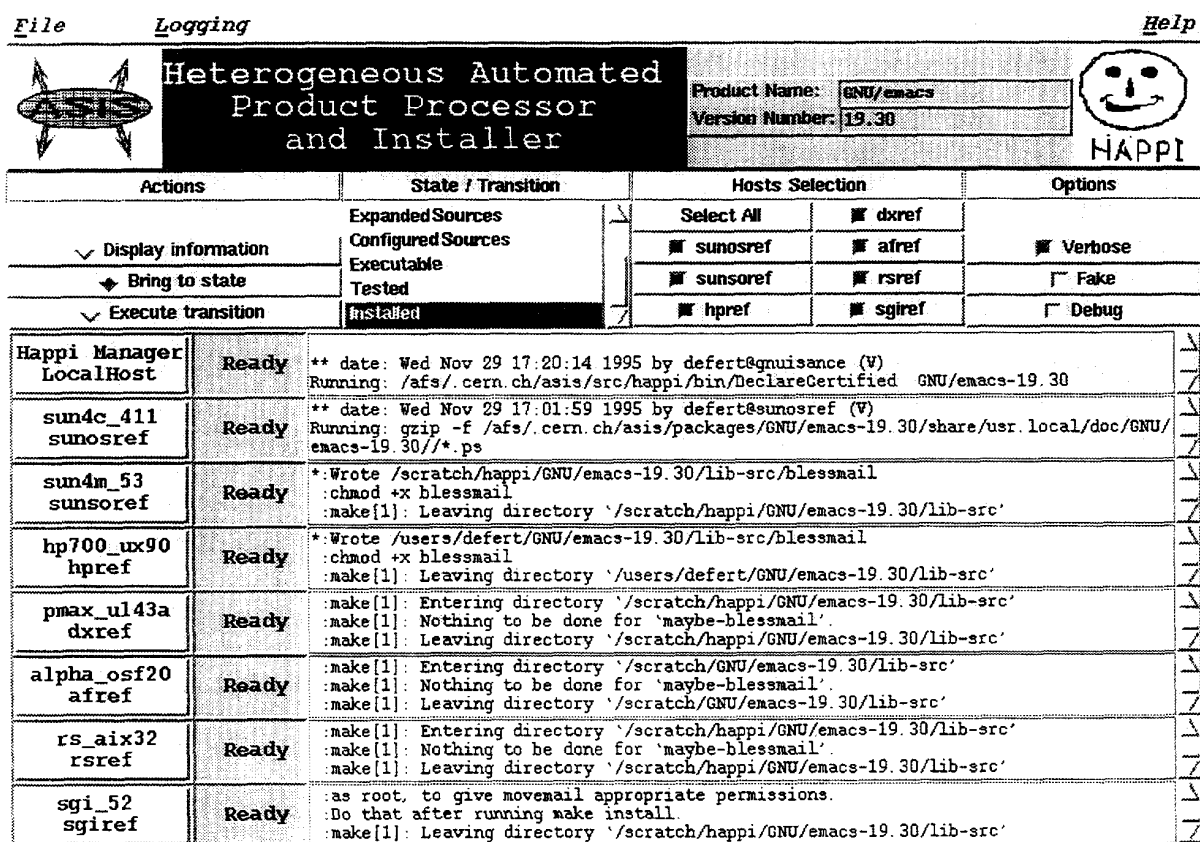


Figure 4: tkhappi: the tk based interface to happi

The actual work of the "product maintainer" is to provide happi with a "configuration file", written in perl, where all transitions are described.

Figure ?? shows the one for X11/tgif which is nicely commented.

In most cases, this file can be very simple, since many packages are well designed and easy to build, and also because happi offers a library of functions corresponding to the most frequent operations.

A more complex case is given in Figure ??, the configuration file for GNU/emacs where some work had to be devoted to correctly separate the installation of the specific and shared files.

Examples of the use of the standard happi libraries can be found in both Figures ?? and ?. Some of these subroutines are described below:

- Split to run GNU tar to expand the sources using the standard location to find the compressed tar file and to store the sources;
- Run to execute a shell command;
- Link to run X11 lndir on the local reference machine to create links to the central sources;

- Make to run GNU make;
- ModifyFile to edit a file;
- file'xxxx for file manipulations;
- Sharelink to create a link from the specific part of the package towards the corresponding shared area;
- GNUMakePSandHTML to generate for each dvi file found in the directory tree its corresponding Postscript and HTML files (this applies only to GNU software as it assumes that documentation is in TeXinfo format).

When the system detects a new version, it automatically brings the product in the state specified, using the definitions of all transitions present in the configuration file.

## 4.2 The ASIS Transaction System

### 4.2.1 Repository Changes as Transactions

Even though the Software Processing Model describes the actual operations performed on a package satisfactorily, the modelling of repository changes needs the

```

## -*- mode:perl -*-
#       File for X11/tgif
#
$final_automatic_state = 'Installed';
#
# The tar.gz file is not in the standard pub/X11.
# 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. This is the ASIS choice.
#
  &Make('TGIFDIR=/usr/local/lib/X11/tgif');
}

sub Install {
# /usr/local/lib/X11/tgif only contains shared
# data, thus make a link from specific
# /usr/local/lib/X11/tgif to the shared one.
#
  $tgifdir=
    "$dir{Installed}/$sys/usr.local/lib/X11/tgif";
  &Sharelink($tgifdir);
#
# Install in 'Installed' area not in /usr/local/...
#
  &Make
    ("BINDIR=$dir{Installed}/$sys/usr.local/bin/X11 \
     TGIFDIR=$tgifdir MKDIRHIER=mkdirhier install");
#
# Install man pages only in the machine
# dedicated to share.
#
  if ($share) {
    &Make
      ("MANDIR=$dir{Installed}/share/usr.local/man/mann \
       MKDIRHIER=mkdirhier install.man");
  }
}

```

Figure 5: The X11/tgif configuration for **happi**

introduction of a new concept. When a product maintainer has to change the version of a product that is **InProduction**, two state transitions must take place: (1) a first version is *RemovedFromProduction*, (2) a second one is *IntroducedInProduction*. Moreover, both operations should be executed in the right order and in an atomic way. Therefore, a transaction system was introduced in ASIS. A transaction is a sequence of operations (transitions and/or specific actions) that are all performed completely or not at all.

A product maintainer is then sure that his package will not disappear from the **InProduction** area when he changes the version **InProduction** even if a problem occurs during the execution of the state transitions. The same concept also solves the problem of packages that depend on each other. If it is necessary to update simultaneously more than one package because of version incompatibilities, the transaction will contain the necessary transitions for all interdependent products.

#### 4.2.2 Replication using the Transactions List

Presently, the ASIS repository is mirrored in many HEP centres. Even though the mirroring is done using clever programs which only transfer the modified files (like described in [?]), a complete traversal of the whole ASIS directory tree is necessary. The operational cost is then proportional to the size of the repository i.e. some thousands of files. If a local replica center happens to know all transactions that were performed since its last update, it can deduce from the ASIS data base what are the files that were modified or moved. The move operations can then take place locally. Modified and new files are transferred through the net. No check is done on unchanged data. The time to update the local replica then becomes proportional to the size of the changes done to ASIS, which makes much more sense.

## 5 The Status of the Project

**ASISUpdate** is now part of the standard CERN installation procedure of all UNIX systems and runs on several hundreds of workstations.

**happi** and **tkhappi** are used for maintaining the majority of the public domain packages. Presently, when a new source distribution kit is released at the distribution site, the **mirror** program does not yet trigger the full generation. It sends an electronic mail to the "Product Maintainer" who then can launch **tkhappi**. All transitions were implemented using idempotent functions except *IntroduceInProduction* and *RemoveFromProduction* that could only be built as atomic operations secured by a checkpointing mechanism.

```

## -*- mode:perl -*-
#       File for GNU/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 --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=/usr/local/include/X11/bitmaps:/usr/local/include/X11/pixmaps');
}

sub Test {
    &Make('check');
}

# Load GNU software specific library
#
require 'happi/GNUMore.pl';

sub Install {
    # Modify Makefile once for installing only the files specific to each platform;
    # thereafter re-modify Makefile to install the shared part
    #
    &ModifyFile('Makefile');
    &Make("prefix=$dir{Installed}/$sys/usr.local install");
    if ($share) {
        *changed = *share_changed; &ModifyFile('Makefile');
        &Make("prefix=$dir{Installed}/share/usr.local install-arch-indep");
    }
    # Some directories were created in specific and should be removed and info is shared data
    #
    &file'wipeout("$dir{Installed}/$sys/usr.local/share");
    &file'wipeout("$dir{Installed}/$sys/usr.local/com");
    &file'wipeout("$dir{Installed}/$sys/usr.local/man");
    &Sharelink("$dir{Installed}/$sys/usr.local/info", 'F');
    #
    # 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");
        &GNUMakePSandHTML();
    }
}

# 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 6: The GNU/emacs configuration file for happi

## 6 Future Developments

Synchronisation of all management processes is done using “locks”, which sometimes results in a dead-lock. When our transaction system will be in place, such problems should no longer occur.

At the same time, the transaction system should allow an optimal management of repositories on local and remote sites, since only files changed during the last transactions will need to be checked. The manager of a replica might also filter the transactions, refusing some, delaying others or modifying strategic ones. For instance, all CERN customisations could be rejected like for instance the `CERN/news-config` package which contains the location of the news server, the news domain name, etc.

Software interdependencies are at present not handled. We are studying possible solutions as proposed in Fermi-lab's UPS/UPD [?] or in Depot-Lite [?], which could be used as a basis for ASIS product dependency management.

## 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 Udu-mula, 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, 1.00 version, 1995.
  - [5] Philippe Defert, Alain Peyrat, and Ignacio Reguero. *ASIS: Users and Reference Guide*. CERN, the European Laboratory for Particle Physics, 3.00 version, 1994.
  - [6] Don Libes. *Exploring Expect*. O'Reilly and Associates, Inc., 1995.
  - [7] 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.
  - [8] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, 1994.
  - [9] John P. Rouillard and Richsrd B. Martin. Depot-lite: A mechanism for managing software. In *LISA VIII Proceedings*, pages 83–91, 1994.
  - [10] John Sellens. Software maintenance in a campus environment: The xhier approach. In *LISA V Proceedings*, pages 21–44. Usenix, . 1991.
  - [11] Stephen Shafer and Mary Thompson. The sup software upgrade protocol. Technical report, Carnegie Mellon University, School of Computer Science, 1988.
  - [12] Rainer Toebecke and Philippe Defert. Recommended standard for unix workstation environment setup. Technical report, CERN, the European Laboratory for Particle Physics, 1990.
  - [13] Larry Wall and Randal Schwartz. *Programming perl*. O'Reilly and Associates, Inc., 1991.
  - [14] Walter C Wong. Local disk depot - customizing the software environment. In *LISA VII Proceedings*, pages 51–56. Usenix, 1993.
- E. Fernandez Dominguez** is completing his End of Degree Project at CERN in order to get a Telecommunications Engineer Title at the Technical University of Madrid (UPM). He may be contacted at `<eusebio.fernandez@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  $\text{\LaTeX}$  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  $\text{\LaTeX}$  Companion* (Addison-Wesley 1994) and since 1994 he is President of the French-speaking  $\text{\TeX}$  Users' Group GUTenberg, and since 1995 also of the International  $\text{\TeX}$  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 design and implement 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. His electronic mail address is `<philippe.defert@cern.ch>`.