SEPP – Software Installation and Sharing System

Tobias Oetiker <oetiker@ee.ethz.ch>
Department of Electrical Engineering
Swiss Federal Institute of Technology, Zurich

September 28, 1998

Abstract

SEPP is an application installation, sharing and packaging solution for large, decentrally managed Unix environments. SEPP can be used without making modifications to the organizational structure of the participants' servers. It provides consistent application setup, documentation, wrapper scripts and usage logging as well as version concurrency and clean software removal. This paper first gives an overview of products already available in this field and then goes on describing SEPP.

Motivation

The Swiss Federal Institute of Technology in Zurich (ETHZ) has a fairly large installation of Unix workstations. At the Department of Electrical Engineering alone, there are more than 400 workstations in operation. Most laboratories in this Department have their own system managers and file servers. The advantage of this distribution of responsibilities is that management happens close to the users. The disadvantage is that it leads to a multiplication of efforts in respect to software installation and system configuration.

In an education and research environment, the diverse user population requires a large variety of software packages in their day-to-day work. These range from little utilities to large applications taking up several GB of disk space. The IT Support Group (ISG) of our department, for example, maintains a software base of over 40 GB.

The ISG started the SEPP project with the intention to devise a software installation system which allows system managers to collaborate closely while retaining independence between the different laboratories. For the users it should bring better service by providing structured documentation on all applications, several versions of the same applications available in parallel and immediate accessibility for all applications without the need to alter `.login` or similar files.

Existing Solutions

Application installation and packaging has been an issue for years. Many solutions have been proposed and implemented. Some are mainly concerned with package installation and have no special support for networked environments:

The RedHat Package Manager by RedHat Inc. [1] is the most widely used package manager in the Linux world. It is geared towards setting up software packages on stand-alone workstations, much like the SVR4 package manager. In addition to this, it provides all the means for distributing software in source format, ready for fully automatic compilation and installation on the target system.

The Redhat Package Manager (RPM) does not impose any restrictions on package layout. Files from a package will be placed into the filesystem wherever the package author sees fit. RPM keeps track of the installed software in a database.

GNU Stow by Bob Glickstein [3] is basically a link generator. The idea behind stow is to put every application into its own subdirectory tree and then generate symbolic links into `/usr/local/`. Stow's special ability is to optimize these links. If only one application provides files in `/usr/local/include`, the whole directory will be linked. Once a second package is installed which also provides files for `/usr/local/include`, stow replaces the symbolic links to a directory with a symbolic link for each include file.

In an environment with many packages this link optimization feature will not help much as most directories will be used by several packages anyway.

The Pack Project by Peter Krisensen [2] is also based on the idea of installing each software package into a separate subdirectory and making the binaries available in a central bin directory using symbolic links. A special feature of Pack is that Peter Krisensen maintains a substantial public collection of 'Pack' packages at Sunsite Denmark.

Other solutions have been developed with a networked environment in mind:

Xhier from the Math Faculty Computing Facility, University of Waterloo [4] is a complete software distribution and maintenance system. It provides highly automated means for compiling and distributing software in a campus setup. Xhier requires applications to be organized into packages of related software. These packages can then be distributed to a number of workstations organized in a tree structure. To provide easy access for the user, all relevant files are linked into a common directory tree.

One major obstacle to the success of Xhier outside of University of Waterloo is that Xhier is not publicly available due to license restrictions.

CMU Depot by Wallace Colyer and Walter Wong [5] is the Unix System Configuration Management component of the Workstation Administration/Host Configuration Andrew II project. CMU Depot puts an emphasis on sharing applications across the network. It was developed for an AFS environment, but works with NFS as well. Applications are organized into Collections of related software, each living in its own directory tree. Users are provided with a central directory containing symbolic links to the application binaries.

Depot-Lite by John P. Rouillard and Richard B. Martin [9] is a mechanism for managing software which is designed to be light weight, easy to learn, and to provide support for multiple installed versions of a package.

ASIS by Ph. Defert, E. Fernandez, M. Goossens, O. Le Moigne, A. Peyrat, I. Reguero is the Application Software Installation Server developed a CERN. ASIS is in use at CERN and other High Energy Physics Research Centers around the world. All ASIS sites work together by storing their software packages in one central repository, from where copies are made to second level servers. From there the software can either be copied to local machines or accessed directly via NSF or AFS. Installing ASIS packages is a particularly simple process by means of a GUI. Software is made available to the end-users through symbolic links written into a central binary directory. The system manager of each participating client can decide which versions of which package to install locally.

LUDE by David Lebel, Duncan Fraser and Michel Dagenais [8] is the distributed software library developed at the University of Montreal. LUDE allows a distributed setup without central control. The local system manager can choose for each package if it should be run over the network or copied to the local system. Participating systems can be both client and server. Each software package is kept in a separate subdirectory. The users access packages through a binary directory from where links point to the binaries of the individual packages. Packages themselves are highly structured to allow the setup of packages which work on multiple platforms. Management of the system is performed through a single command-line tool.

UPS by William Bliss, Jonathan Streets, Lourdu Udumula, and Margaret Votava is the UNIX Product Support and Distribution toolkit developed at Fermilab for the management and access of software products on local systems by the system administrators and users. UPS supports multiple concurrent versions of the same product available on the same machine. End users have to use a special ***setup*** program to prepare their account for each software package they want to use. Inter-package dependencies are resolved automatically when running the ***setup*** command. Each package is assigned a status like `current, new, test, development` or `old`. The users can use these status labels to choose packages on a maturity level. Information about the available packages is maintained in an external database in the form of a special directory tree.

None of these packages addressed all our local requirements. Most packages were rather large compared to what we had in mind, and none supported wrapper scripts[1] A mix of features from the packages mentioned above plus some local ideas, however, provided a suitable software installation and sharing system for several Departments of the Swiss Federal Institute of Technology. We called this system SEPP.

SEPP Overview

SEPP is a package based software distribution system. Figure 1 shows the major components of a SEPP installation. Two packages are installed in this example.

[1]Wrapper scripts are explained in the next section.

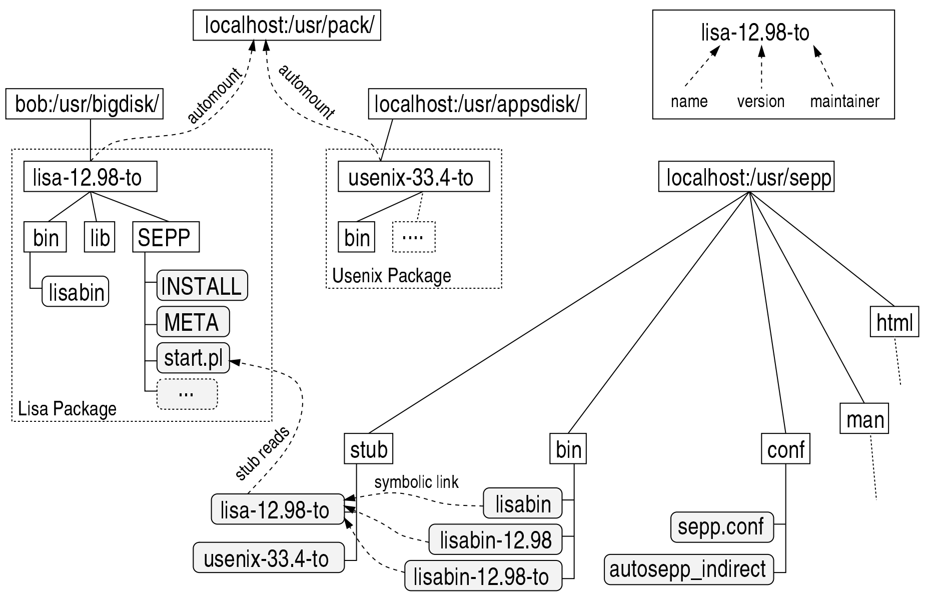


Figure 1: Components of SEPP

- Every software package is installed into a separate subdirectory providing clean encapsulation of all files belonging to the same product.
- Every package contains a special directory called `SEPP`. This directory holds a few files describing the contents of the package, as well as a startup wrapper script (`start.pl`).
- This wrapper script is responsible for preparing the environment for successful execution of the binaries contained in the package. Whenever a program which was installed with SEPP is started, the program does not get executed directly, it is rather the wrapper script of the package which is called and, after preparing the environment, runs the requested program.
- Packages are made available on the local machine using the automounter. The package directories are always mounted below `/usr/pack`. This ensures that software which relies on compiled-in absolute path names finds its files.

  If a package is available from several places, the automounter map is constructed to use alternate sources for the package if the primary server is not responding.
- The packages' binaries are made available to the end-users trough symbolic links in the `/usr/sepp/bin` directory.
- These links do not point directly into the package directories, but to *stub* scripts stored in `/usr/sepp/stub`. SEPP generates one stub script for each package it installs. Stub scripts are written in Perl and are responsible for running the package's `/usr/pack/`*`package`*`/SEPP/start.pl` file. The *stub* and the `start.pl` file together make up the wrapper script of the package mentioned above.

- Package names are built from 3 components:
  1. The name of the package
  2. The version number of the package
  3. A shorthand for the name of the package maintainer

  This ensures that package names are unique and everything can be mounted under `/usr/pack`.

Starting from this setup, SEPP adds many convenient features both for the users as well as for the administrators of packages.

User Features

While it is good for system managers to have a clean and well organized software setup on their servers, the user's comfort must be the prime objective. The main user-visible feature of SEPP is therefore *ease of use*:

- To use an application installed under SEPP, no changes to `.login`, `.cshrc` or `.profile` are required, apart from adding `/usr/sepp/bin` to the `PATH` variable. The `/usr/sepp/bin` directory contains symbolic links representing all installed applications. Each program is started through a wrapper script which prepares the environment according to the requirements of the program. This includes choosing the appropriate binary in a multi-architecture environment, setting special environment variables or creating configuration files before the program is run for the first time.

- Documentation about all the locally available packages is provided on a web site and, where possible, also as manual pages. By design, SEPP forces the system manager to provide at least a minimal amount of structured documentation to be present in a package which is then used to automatically generate a documentation web site.

- SEPP supports the installation of several versions of the same package concurrently. The user can start the default version of a program by using the plain program name, while other versions are available through *`program-version`*. This means, for example, that Emacs 20.2 is started by using the command *`emacs`*. But version 19.23 is also available to users who start it with *`emacs-19.23`*. If two administrators are maintaining *`emacs-20.2`* packages and set them up differently, both packages can be installed concurrently on the same system. The versions are then distinguished by a second suffix to the executables, based on the names of the two persons maintaining the packages. The system manager of each system can decide which package and version is the default and is started by typing *`emacs`*.[2]

[2]The wrapper script mentioned above takes care of removing the version number from `ARGV[0]` so that applications which depend on being called a certain name work as expected. prior to calling the binary. The wrapper also adjusts the PATH variable so that the program finds the correct version of any companion programs it might call during operation. For emacs this means that it would always use the version of `movemail` which was installed together with the particular version of emacs.

Management Features

The users can only benefit from SEPP's features if the system managers actually provide applications through SEPP. Therefore much effort was spent on making SEPP easy to use from the system managers point of view:

- SEPP is primarily an organizational measure. It does neither require any special daemon processes nor root privileges to work. Installing SEPP does not require altering the whole system setup. It takes only about 15 minutes to set up SEPP on a server, plus some additional time to update the clients' automounter maps and syslog configurations.
- A Perl script called *seppadm* is provided, which simplifies the maintenance of SEPP packages. The *seppadm* tool sets up skeleton installation trees for new packages and installs and removes SEPP packages from a server. Furthermore the *seppadm* tool ensures that no name clashes occur when installing a SEPP package. This is done both for stock OS binaries as well as other SEPP binaries. If clashes occur with other SEPP packages, the administrator can define whether or not the new package overrides old binaries and manual pages. Because of the elaborate naming scheme for binaries, this mechanism provides an ideal test bed setup for new versions of a package. While the previous version of the binary remains available under the normal name, the new version can be accessed by appending the version number to the binary's name.
- The automounter is used to mount all package directories below `/usr/pack`. This makes the physical location of a package directory irrelevant. A package can be stored on any partition of the local machine or on a remote server. The application binaries still appear to be installed under `/usr/pack/`*package*. This even fools setup programs of commercial applications which use *pwd* to determine their installation directory.
- Every SEPP installation maintains a catalog file listing all packages stored locally, together with their NFS pathname and a short description. SEPP can be configured to use catalog files from other servers to gain access to all their locally installed packages. This allows several SEPP servers to be tied together without requiring central management.
- A package can specify a list of other packages which are required before it can be installed. In general, however, it is preferable when a package contains all the tools and libraries it needs to run. This takes some additional disk space but is much simpler to maintained than multiple packages all cross-linked together. In our experience this policy usually does not lead to a significant growth of package size.
- The application wrapper scripts mentioned above allow the package maintainer to take any action required to make the application work, just prior to launching the program binary, without making the end-users edit their `.login` file. This cuts down support time because programs "Just Work".
- The wrapper scripts automatically log application usage through *syslog*. This allows to track application usage by configuring the syslog daemons on all clients to forward their messages to a central logging server.

- Some applications have configuration files which must be adjusted to the local environment. SEPP can handle this problem by copying part of the package's directory tree to `/usr/sepp/var/`*`package`* which is a local directory on every SEPP server. The application itself has to be configured to pick up its configuration file from `/usr/sepp/var/`*`package`*.

Using SEPP

The following sections gives a brief example of how to create and install a SEPP package called `lisa-12.98-to` using the *`seppadm`* tool.

First a word on the terminology used in this section:

Package Preparation is the first step to make an application available within a SEPP setup. It involves using the *`seppadm`* tool to create a skeleton package directory, downloading and compiling the software, installing the software into the skeleton directory and finally updating the files in the package's SEPP directory to fit the application.

Package Installation makes programs contained in a SEPP package visible in the `/usr/sepp/bin` directory and therefore available to all the users who have this directory in their PATH variable. When installing a package, it does not matter if the package is stored on the local system or on a remote server as all file access is governed by a single automounter map. A new package only becomes visible to remote sites after it has been installed successfully on the site where it has been prepared.

Package Mirroring allows the system manager to make a local copy of a package which has been installed from a remote server.

Creating a SEPP package

1. *`seppadm prepare lisa-12.98-to`* creates a skeleton application installation directory and updates the automounter map to make the directory available as `/usr/pack/lisa-12.98-to`. The physical location of the install directory is chosen automatically from a list of possible locations by selecting the location with the maximum available disk space. The list of storage locations has to be configured when installing the SEPP base package. It is also possible to give an absolute location when creating a package directory.

2. After downloading and unpacking the source, it can be compiled. Assuming the example package uses autoconf, compilation is very simple:

   ```
   ./configure --prefix=/usr/pack/lisa-12.98-to
   make; make install
   ```

   This configures, compiles and installs the program into the new package directory. The only change necessary to the standard compilation procedure is the use of the *`--prefix`* argument to guide the program into the right directory and prevent it from being installed into *`/usr/local/bin`* where it would usually go.

3. The *seppadm* command in the first step copied several template files into `/usr/pack/lisa-12.98-to/SEPP/`. These files must now be edited to fit the application:

   INSTALL contains a detailed description of the steps necessary to compile and install the package.

   META is a structured text file with information about the package. It includes a one-line description of the package, the addresses of the local package maintainer and support staff, and pointers to the package's binaries and documentation. The *seppadm* tool reads this file when installing a package or when regenerating the SEPP documentation web site.

   README is a brief description of the package. It may include information about local changes, solutions to frequent problems, ...

   CHANGES lists all changes which were done to the package after initial installation within SEPP.

   patches is a subdirectory where patches are stored which were necessary to get the package to work.

   start.pl is the wrapper script for the application. In the simplest case it will just contain the line `AppRun "bin/"`. More problematic software products may require the setting of environment variables or the creation of per user configuration files. When a user starts an application installed under SEPP, this script will always be run before the actual application binary is executed.

Installing a SEPP package

seppadm install lisa-12.98-to makes the package available on the local server. Every user who has `/usr/sepp/bin` in the PATH can now access the *lisabin*, *lisabin-12.98* and *lisabin-12.98-to* programs.

Mirroring a SEPP package

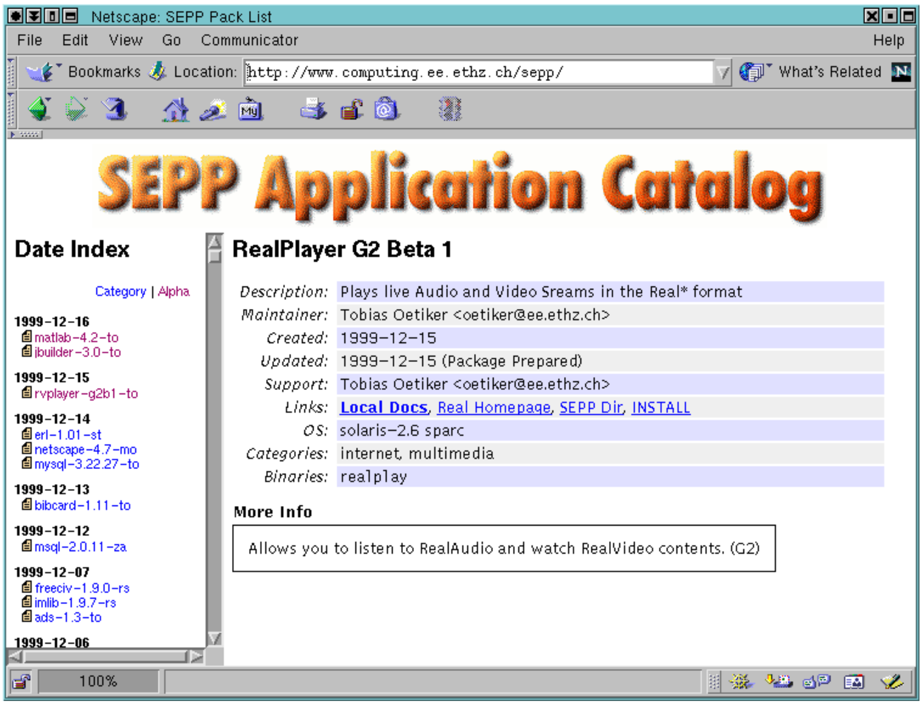
Remote system managers can not only install the application, they can also make a mirror copy of it, using *seppadm mirror lisa-12.98-to* to ensure maximum performance and availability.

Other functionality of *seppadm*

Apart from the basic functions shown above, *seppadm* can also build a web site which lists all the applications installed locally (*webbuild*), as shown in Figure 2. Further there are functions to retrieve a listing of all applications (*report*) available from remote sites, for updating the local mirrors (*mirrorupdate*) and for removing old package (*remove*).

Real World

As explained in the "Motivation" section, SEPP had to be an easy-to-use, distributed solution in order to gain acceptance within the labs. During the devel-



SEPP Application Catalog

Date Index

Category | Alpha

1999-12-16
matlab-4.2-to
jbuilder-3.0-to

1999-12-15
rvplayer-g2b1-to

1999-12-14
erl-1.01-st
netscape-4.7-mo
mysql-3.22.27-to

1999-12-13
bibcard-1.11-to

1999-12-12
msql-2.0.11-za

1999-12-07
freeciv-1.9.0-rs
imlib-1.9.7-rs
ads-1.3-to

1999-12-06

RealPlayer G2 Beta 1

Description:	Plays live Audio and Video Sreams in the Real* format
Maintainer:	Tobias Oetiker <oetiker@ee.ethz.ch>
Created:	1999-12-15
Updated:	1999-12-15 (Package Prepared)
Support:	Tobias Oetiker <oetiker@ee.ethz.ch>
Links:	**Local Docs**, Real Homepage, SEPP Dir, INSTALL
OS:	solaris-2.6 sparc
Categories:	internet, multimedia
Binaries:	realplay

More Info

Allows you to listen to RealAudio and watch RealVideo contents. (G2)

Figure 2: SEPP generated Documentation Web Site

opment phase of SEPP, the ISG kept close contact with managers from various labs of the department. An early design document was distributed to get feedback on the proposed design and features. This had the double benefit of getting people interested in the project and tidying up the design before it was even implemented. Once the first release of SEPP was available, the ISG started to install all new software under SEPP. This soon led to a substantial amount of packages being available. There was a lot of interest when talking about SEPP with the lab system managers. In most cases though it was the need to get access to some new software package from the ISG server which led to the installation of the SEPP base package on a lab server.

SEPP offers the possibility to transparently mirror a package to the local server, to enhance availability of the package as well as to reduce load on the network. Comparing the number of packages which are mirrored to the number of packages which are just cross-mounted between servers, shows that most system manages prefer to keep local copies of smaller packages while software in the > 1GB class is mostly cross-mounted. This might change in the future once the "switched 100 Mbit to the desktop" plan of the ETHZ is put into practice.

Future Directions

SEPP was designed to work without external databases apart from a text file listing the packages physically available on the local server and the automounter map. All other information is taken directly from the `SEPP` directory inside each package. With an increasing number of packages installed, the processing of this information can take a considerable amount of time. A future version of SEPP might use a cache file with pre-processed information, which has only to be updated when the `CHANGES` file of a package has been altered. Performance is not yet a problem in our setup. With 90 packages installed it takes 15 seconds to regenerate the whole documentation web site on the ISG's Ultra Enterprise 2 + SSA server. This includes analyzing all packages and writing out a web page for each one.

The current implementation of SEPP works best for user applications which are not required to successfully boot a machine. Daemon processes can be provided as SEPP packages as well, but because of the transparent automounting feature, this could lead to unintended dependencies between different servers. In the worst case, this could make it impossible to boot when two machines crash at the same time while depending on packages from each other. To prevent this problem from occurring, a feature will be added to SEPP which enforces that crucial packages are always mirrored to the local server.

With a number of sites using SEPP, it has become difficult to add major new features to the system as packages are cross mounted between different servers. One idea to alleviate this problem would be to have version numbers for the package format and make the administration script check these before installing a package. If the version number of the package format was higher than the one handled by the administration script, the system manager would be offered to retrieve a new version of the administration script.

At the Swiss Federal Institute of Technology, SPARC/Solaris is by far the most widely used Unix platform. Therefore it has been the SEPP system's primary target. The overall design of the SEPP system takes multi-platform capability into account, and it is successfully being used in a mixed Solaris/Irix environment, but running it in a really mixed environment with other than SVR4 based Unix variants would be an interesting test for the systems design.

Conclusion

The potential productivity and quality of service provided by the system managers of the department was increased both because more applications are available to the users in a consistent setup and because the individual system managers can devote more time to direct user support and conceptual work. This in turn also leads to a better quality of life for the system administrators and thus generates a positive feedback loop.

Acknowledgments

I would like to thank my fellow system managers Elmar Heeb, Andi Karrer, Christoph Wicki, and Fritz Zaucker at the Swiss Federal Institute of Technology for the feedback during the design process and beta testing of the final product. And last but not least I would also like to extend my gratitude to *Larry Wall* for creating Perl.

Availability

The SEPP base package as well as details about the SEPP mailing-list are available from
`http://www.ee.ethz.ch/sepp/`. SEPP is distributed under the terms of the GNU General Public License.

Author Biography

Tobias Oetiker got a Master's degree in Electrical Engineering from the Swiss Federal Institute of Technology, Zurich (ETHZ) in 1995. After working for one year at De Montfort University in Leicester, UK doing Unix system management, he returned to Switzerland and has since been employed by the Department of Electrical Engineering of the Swiss Federal Institute of Technology as a toolsmith and system manager.

What is in a Name

In case you have been wondering what SEPP stands for, I must disappoint you: It is not an acronym. Sepp is a Swiss and Austrian short form for Joseph, and one might have the image of an old mountain farmer in mind when hearing the name. Maybe a future successor to SEPP will be called HEIDI.

References

[1] RedHat Inc. RedHat Package Manager
http://www.rpm.org

[2] Peter Krisensen. Pack Distribution Project
http://sunsite.auc.dk/pack/

[3] Bob Glickstein. GNU Stow application installer.
http://www.gnu.ai.mit.edu/software/stow/stow.html

[4] John Selens. Software Maintenance in a Campus Environment: The Xhier Approach. LISA V - Sept. 30-Oct. 3, 1991.

[5] Wallace Colyer and Walter Wong. The CMU Depot Project
http://andrew2.andrew.cmu.edu/depot/

[6] Anne Heavey. UPS and UPD v4 Reference Manual
http://www.fnal.gov/docs/products/ups/

[7] Ph. Defert, E. Fernandez, M. Goossens, O. Le Moigne, A. Peyrat, I. Reguero. ASIS Application Software Installation Server
http://wwwcn.cern.ch/dci/asis/

[8] David Lebel, Duncan Fraser, Michel Dagenais. A Distributed Software Library
http://www.iro.umontreal.ca/lude2/

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