All For One Port, One Port For All

Bram Moolenaar
Stichting NLnet Labs
<Bram@A-A-P.org>

The ports system provides a convenient way to install an application from source code. With just a few commands the files for the latest version are downloaded, build and installed. A port specifies patches that need to be applied, allows tweaking features and handles dependencies on other components. These useful features of the ports system have increased the popularity of BSD distributions.

Each BSD distribution has their own ports system. Although they all originate from the same root, incompatible features have been added. This requires a port to be done and maintained for each system separately. Since there are thousands of ports, the amount of duplicated work is significant.

Attempts to reunite the ports systems have failed so far. Examining the reasons for this makes clear that the chances for each BSD system to drop their own solution and use a common ports system are very small. The development of solutions that replace the existing ports systems have stalled.

A possible solution is introducing a new system that exists side by side with the traditional ports system. This allows a gradual shift, moving ports to the new system one by one. Since the ports files of new system do not need to be backward compatible, there is a lot of freedom to make choices for a better and more powerful implementation. The goal that it must co-exist with the traditional ports systems makes sure it avoids the pitfalls that stopped previous reuniting attempts from being successful.

A first version of this new system has been implemented. To avoid the complicated mix of Makefile and shell script the recipe format of the A-A-P project has been used. This first implementation shows the advantages and possibilities of the proposed solution, but also the problems that still need to be solved.

1. HISTORY

The first available code for the ports system was written by Jordan K. Hubbard. The date mentioned in the file, still present in derived works, is August 20 1994 [FreeBSD]. This version did not handle dependencies or downloading of files. Development went fast though, by January 1995 there were already 150 ports [Asami].

In the following years the OpenBSD and NetBSD projects have split off a version from FreeBSD and started adding their own features.

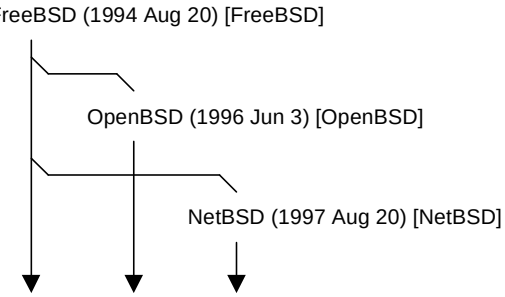

figure 1. History of BSD ports systems

This figure does not show various exchanges of modifications. All three systems have been enhanced over time, which can be seen in the CVS logs [FreeBSD] [OpenBSD] [NetBSD].

NetBSD uses the name "pkgsrc" for their ports system and uses "package" for what others call a "port". To keep it simple we will use the term "port" here for dealing with sources, "package" for an installable set of files with binaries and "source package" for a port that includes the required source files and patches.

Zoularis is the name used for an adaptation of NetBSD pkgsrc to other systems, such as Solaris [Zoularis]. It is not different from pkgsrc in functionality, therefore we will not mention it below.

NetBSD started pkgsrc because the FreeBSD system had a few small problems that needed to be solved (i386 centric, fixed install directory). They apparently didn't try very hard convincing the FreeBSD people to fix this. According to Hubert Feyrer: "it was easier for us to just make the changes we wanted". And FreeBSD apparently wasn't interested in including all the NetBSD improvements either.

OpenBSD first forked off their version with the intention to feed back changes to FreeBSD. A few years back OpenBSD intentionally improved their ports system separately from FreeBSD. Partly to clean up the Makefiles and also to add useful features. Currently there does not appear to be an intention to resync with FreeBSD.

It is clear that incompatibilities between the three ports systems were not added intentionally or because of technical reasons, but are the result of lack of motivation to work together on one system. Being able to quickly change their own version instead of having to convince someone else to include changes also appears to play an important role.

2. BENEFITS OF ONE PORT SYSTEM

The separation into three different ports systems causes several problems:

- Maintenance of each port has to be done three times. This can vary from simple changes, such as updating the version numbers and file names, to handling more complicated issues.
- Bugs and security problems that are not solved by the maintainer of the original sources have to be fixed three times.
- Tricks required to port an application to a BSD system have to be figured out three times.
- Dependencies have to be figured out three times.
- If one BSD system adds a nice feature to their ports system, it is not directly available to the others.

Would there be one ports system, there are several benefits:

- For each port there are more eyes to detect a bug and more hands to fix it. This makes the average reliability of the software considerably higher.
- Ports will be updated quicker and more ports will be available.
- Fewer people are required for maintenance.
- Each BSD system will become more powerful and attractive.
- When a user is making a choice for one of the three BSD systems he no longer needs to exclude systems that do not provide a port he requires. He can make the choice on more relevant issues.

One thing that will not change is the effort required for testing. Each port still needs to be tried out on many different systems.

3. FAILED SOLUTIONS

Despite the obvious benefits from using one ports system, the attempts to bring the BSD systems back together failed so far.

The most promising attempt to make a ports and packages system that should replace all others is Open Packages [OpenPackages]. It was created with the goal to reunite. But the project has stalled, the last news item is dated July 2001. Why? Talking with the developers reveals several reasons.

At first the project consisted of a good idea for reuniting the three solutions. Then the contributors got carried away and wanted to do much more. It became too much work and developers started dropping out. An attempt to redefine the project and split it up in manageable pieces has been made, but there do not appear to be developers who will do the work. Out of the six modules five have no project leader. Possible reasons are:

- Developers do not find the extra features that Open Packages offers important enough to spend a lot of time on.
- Implementing the features is quite complicated. There is much "hidden knowledge" in the existing ports systems and few comments or documentation to explain why certain choices were made. Not many people have the skills to do the work.
- The people who do have the required skills prefer working on their own project (esp. the maintainers of the ports systems of the BSD distributions).

Another alternative is OpenPKG [OpenPKG]. This project appears to promise more than what it can actually do. It says it is portable, but it uses GNU bash shell scripts and an incompatible version of rpm. It also uses many specific system utilities. You can say it could be made portable. Its Linux background and lack of co-operating with the existing packages system make it unattractive to replace the BSD ports systems. When used side-by-side it does not handle dependencies between the two systems.

There are a few other ports and/or package systems, but they do not come close to being a useful replacement for the BSD ports systems. [Install Tools]

4. HOPELESS SOLUTIONS

Instead of switching to a new system, a solution would be to have the existing systems come back together. This would require that features from all three existing systems are included into a common version, with the result that the port files are identical. There are several reasons why this is very unlikely to happen.

The current situation is that FreeBSD is the most popular system and has the largest number of ports, while OpenBSD and NetBSD offer more features. This creates a deadlock: FreeBSD is doing well and does not appear to be interested to change their ports system. NetBSD and OpenBSD have more features, thus do not want to use the FreeBSD implementation.

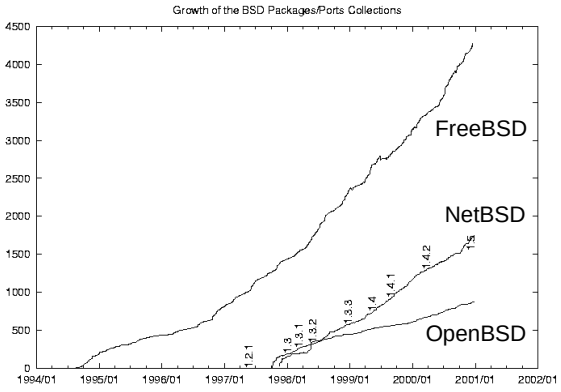

figure 2. Growth of number of BSD ports [Feyrer]

This graph shows the growth of the ports systems over a long period. Recent figures: FreeBSD 7523, NetBSD 3163, OpenBSD 1965. The OpenBSD ports system uses flavours, which reduces their count by an unknown number.

Another important reason is that the developers of each BSD system do not like the idea of giving up the control of their code. They like to be able to decide what goes in and what does not. The releases happen asynchronously, freezes sometimes make it difficult to include updates. Thus a release of one BSD distribution gets in the way of development of the others.

The ports systems are not compatible. Reuniting them means that thousands of ports need to be adjusted. Even when some of the work can be automated, the testing will still be a huge amount of work. None of the BSD systems will want to take this effort in between two minor releases. And doing it for a major release means two versions of each port have to be maintained for a longer time, since a new major release exists besides the previous, stable release branch for more than a year.

How about creating a ports system that is backward compatible with each of the existing ports systems? This could start in such a way that a port contains a big "if" statement, separating the code for each system. Then gradually common

parts can be collected until the system-specific part is made very small. The problem with this is that it would be very complicated to implement. A Makefile is not a programming language. The combination of BSD specific Makefile syntax with shell scripts and the make program re-invoking itself leads to very ugly code. This solution probably leads to the worst code quality possible. And it seems to be impossible to find people who are motivated and have the skills to do this work. Even more so than for the stalled Open Packages project.

5. A NEW SOLUTION

If reuniting is impossible, and existing efforts do not lead to a useful ports system, then what will? What we need is a new application that offers a path for a gradual shift towards a unified ports system. That is the only way to avoid having to rewrite all the ports at once. The existing ports systems will continue to exist, the new solution has to co-operate with it.

In the following we call the existing ports systems "traditional" and the proposed solution the "new" system. Similarly we will talk about a "traditional port" for the existing ports systems and use "new port" for the proposed solution.

Once a port has been made for the new system, it should work on all BSD systems and possibly others. This makes it attractive to create ports for the new system. It is not necessary to transfer all traditional ports to the new system, thus there is no big threshold to start using it.

The required co-operation with the traditional ports systems implies that dependencies can be handled in both directions. A new port can depend on a traditional port and the other way around. And the registration of installed packages must use the existing package system to avoid the need for two sets of commands to find out what packages are currently installed.

This all sounds very nice. The question is how this will be implemented. Using make (BSD or GNU) has the disadvantage of resulting in ugly solutions when it gets complicated. Quoting Jordan Hubbard: "FreeBSD ports is essentially implemented as some very impressive but hairy BSD make(1) macros and can be a little opaque and non-extensible from the perspective of someone looking to extend or re-factor parts of the system" [Hubbard].

Inventing something new specifically for use in the new ports system has the disadvantage of yet-another-file-format. A script language like Python or Perl would work, but still requires adding a lot of application specific functions and variables, thus creating a new file format anyway. And it would be very different from the current ports files, which would discourage quite a few people. Examples are Cons (uses Perl) [Cons] and SCons (uses Python) [SCons].

The solution proposed here uses the A-A-P recipe. This is not the only possible solution, but one that has a good chance of being successful. An alternative might be DarwinPorts [DarwinPorts]. More about that in section 12.

6. INTRODUCING A-A-P

Understanding the proposed solution requires knowing the basic idea of A-A-P recipes.

The A-A-P project provides a portable framework for developing, distributing and installing software [A-A-P]. What is relevant for this paper is the A-A-P recipe. It was specifically designed to replace Makefiles and shell scripts. It provides much more functionality and avoids the dependency on shell commands with specific options and features. A recipe is often portable to many different systems, including MS-Windows.

A recipe is like a Makefile in many ways. The base is formed by specifying the dependencies between files. The build commands of a dependency are used to produce a target file from sources. Everyone using Makefiles will quickly understand the structure of a recipe. Here is an example for compiling a C program:

```
myprog : main.c version.c extra.c
    :do build $source
```

The ":do" command invokes a build action. It detects that the sources are C code and decides to use a C compiler. How the compiler will be invoked depends on the system. This is automatically detected or comes from a configuration file. This separates the system-independent specification of what needs to be done from the system-dependent details. It is also still possible to invoke a shell command when portability is not required.

Some of the advantages of using the A-A-P recipe instead of a Makefile for building a program:

- Signatures are used instead of timestamps, this avoids problems with networked file systems

and files unpacked from an archive. Timestamps can still be used when desired.

- Dependencies on included files are handled automatically. There is no need for running "make depend".
- Building for different systems from one set of sources is handled automatically, a separate directory is used for intermediate results (object files) of each system.
- Rules for line continuation are flexible, backslashes are not often needed. This avoids the most common mistakes. The amount of indent is used to indicate where a command ends. Example:

```
SOURCE =    main.c
            version.c
            extra.c
TARGET =    myprog
```

This actually does almost the same as the previous example. The SOURCE and TARGET variables are turned into a dependency automatically.

For often-used tasks the built-in commands can be used. These are similar to common shell commands, but with extra features. For example, the ":copy" command can copy files specified with a URL (http:, ftp:, scp:, etc.). This greatly reduces the use of specific system commands and improves portability. Some of the built-in commands are:

`:copy`	copy files and directories
`:move`	rename/move files and directories
`:mkdir`	create a directory or directory tree
`:delete`	delete a file or directory tree
`:cat`	concatenate files
`:include`	include a recipe file
`:child`	read a sub-project recipe file
`:execute`	execute a recipe
`:system`	execute a shell command
`:update`	execute build commands when a target is outdated

These commands can not only be used in build commands, but also at the top level. This makes it possible to use a recipe like a shell script. This is one of the advantages of an A-A-P recipe over using a Makefile that is important when implementing a ports system.

For control flow and expressions Python script can be used. This provides a powerful and well-defined syntax that combines pretty well with the Makefile-like syntax of the recipe (e.g., comments start with “#” and amount of indent is significant). Python libraries provide functionality to handle almost any task and make it possible to avoid system-specific code. Example:

```
SUBDIR = sub
USE_SUBDIR ?= 0
@if USE_SUBDIR:
    SOURCE += `glob(SUBDIR + "/*.c")`
```

The line starting with "@" in this example is a Python command. The USE_SUBDIR variable can be set by a non-Python assignment and is used by the Python command. Thus the variables available in the Python code and the non-Python code are the same. In the last line a Python expression in backticks is used. The Python glob() function expands wildcards and the resulting list of files is appended to the SOURCE variable.

The A-A-P recipe has uploading and downloading functionality built-in. For example, a file can be downloaded automatically by specifying where it is to be obtained:

```
EXTRA_SOURCE = extra.c {refresh =
    ftp://ftp.foo.org/pub/files/extra.c}

xfoo : $SOURCE $EXTRA_SOURCE
```

In this example, if the "xfoo" target is updated and the file "extra.c" does not exist locally, it will automatically be downloaded. The text between { and } specifies an attribute. Attributes provide a generic mechanism to attach meta information to a file name.

Uploading is done in a similar way. A recipe like this is used to update the A-A-P web site:

```
FILES =   `glob("*.html")`
          `glob("images/*.png")`

:attr {publish =
    scp://vimboss@vim.sf.net/vim/%file%}
    $FILES
```

The file names are assigned to FILES using the Python glob() function. The destination of the files is added by attaching the "publish" attribute to the file names. Executing this recipe with "aap publish" will cause each file with a "publish" attribute to be uploaded. The uploading is skipped for files that did not change since the last upload.

A-A-P is a generic tool, a sort of a super-make. You can use it to develop software, distribute files, download, install, etc. More information can be found on the web site [A-A-P].

7. PORTS WITH A-A-P

Since A-A-P recipes are powerful and still resemble Makefiles, they form an excellent base for implementing a new ports system. When using a recipe for a port file, in many cases it will not be necessary to use the extra A-A-P features and the new port mostly looks like a traditional port. When more complicated tasks are to be performed, the recipe file offers the functionality in a nice way. A first implementation of this new ports system has been made.

A user of the traditional ports system usually performs these steps:
1. become root
2. update the whole ports tree with cvsup
3. build and test: "cd group/appname" "make" "make test"
4. install: "make install"

Using a new port the usual steps are:
1. download or update a port recipe (one file)
2. build and test: "aap", "aap test"
3. try it out: "aap install DESTDIR=$HOME"
4. install: "aap install"

The new port works in a similar way as the traditional port. Dependencies will be handled where needed, files are downloaded and patches applied. The most important differences of using a new port are:
- It can be run anywhere, it does not need to happen in /usr/ports.
- There is no need to obtain the whole ports tree before installing one port. Only parts that are actually used will be updated.
- Not doing the downloading and building as root is much more secure. For installing a package (also for dependencies) the root password must be entered once.
- To try out a port it can be installed for one user.
- Updating to a new version is simply done with "aap refresh".

Not all of this is easily implemented and quite a few choices need to be made. The most important issues will be discussed in the following sections.

8. USING PACKAGES

There are two basic methods for installing a port:
1. The port directly installs the files to their final location. A binary package can be created after this. Recording the port as being installed is done separately from the actual install.
2. The port installs the files into a temporary directory. This is often called a "fake install". A binary package is created from these files. The binary package is then installed and registered as being installed.

The second method has many advantages. It avoids accidentally overwriting existing files. The first method is actually impossible when a binary package is to be created without installing it, an already installed package using the same files would be corrupted. The second method also makes sure that installing the port gives the same results as installing the binary package.

A disadvantage of the second system is that for some ports it involves extra work to make the installation put the files in the temporary directory instead of /usr/local. This is a small price to pay, therefore the choice was made to use the second method.

Since the package administration is not the same on all systems, A-A-P leaves the work of installing the binary package to existing system tools.

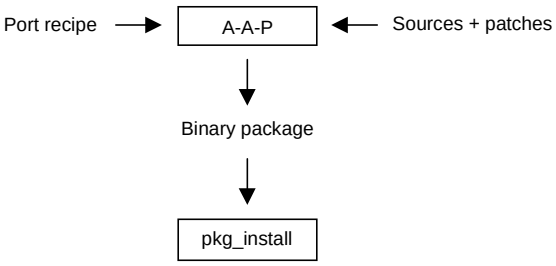

figure 3. Connection between A-A-P and the package system

A disadvantage of this system is that not all package tools support sufficient features. Desired features are:
- Dependency handling on a range of package versions and with wildcards.
- Possibility to install two versions of the same package at the same time.
- Support for the sequence: Install version 1.1, install version 1.2, verify that version 1.2 works well, delete version 1.1.

These are generic problems and separate from the porting issue. They should be solved in the package tools. Adding another set of package commands next to the existing ones is not a good idea, since the existing commands will not be aware of packages installed with the new commands. This must really be solved by

improving the existing package tools. Until this has been implemented the new ports system will accept the limits of the existing commands. Some issues could be handled by adding a pre-install script to the package, e.g., for handling dependencies with wildcards. However, this causes new problems, the time would be better invested in improving the package system.

A source package is nothing more than an archive containing the port recipe with all required source files and patches. No downloading will be needed then. Otherwise the building and installing works just like using the port recipe.

9. DEPENDENCIES

The dependency checking is split up in two parts:

1. Verifying the dependencies can be met. This happens before archives and patches are downloaded, so that wastefully downloading something that will not work is avoided.
2. Installing ports and/or packages that are required happens just before they are needed. This reduces cyclic dependency problems.

There is no need to update all ports before installing one. To figure out the dependencies only the port recipe has to be obtained. Normally, when a recipe can be found on the system that meets the dependency it is used. It is also possible to specify that the latest version of the recipe must be obtained.

There is no need to specify the directory (e.g., "editors/emacs20-mule-devel") for a dependency. This has always been confusing, especially for ports that exist in more than one place or are moved. A unique name is required anyway. This also allows including a version number in the directory name or adding a subdirectory with versions, so that several versions of a port can co-exist. A simple, automatically generated index file is used to locate an application locally. The same can be done on a server that provides ports for downloading.

Alternatively, unofficial ports can be obtained from various locations. This is especially useful for ports under development that depend on ports that have not been committed yet. For stable ports this should not be used.

As mentioned above, the dependencies that can be specified in a package are not always sufficient. It might be necessary that the dependencies in the binary package specify fixed version numbers, thus are less flexible. Therefore the dependencies of the ports recipe will be used when the recipe is available.

Besides the dependencies on ports and packages that need to be installed, the A-A-P recipe also offers features to check for installed tools and decide how to build the application. This automatic configuration is useful to reduce the number of dependencies for applications that do not use autoconf and do allow specifying optional features when building.

10. BACKWARDS COMPATIBLE

To be able to work properly side-by-side with the existing ports system, the dependency of a new style port on a traditional port must be handled. This is not different from dependencies between traditional ports. A-A-P will invoke the traditional ports tools. The knowledge of how this is done on different systems is build into A-A-P. The user will only need to do a bit of configuration if he is not using the standard setup.

The dependency of a traditional port on a new port requires a bit more work, since the traditional port does not know about the existence of the new ports system. A wrapper port is required to make the connection. Most of this wrapper is the same for all wrapper ports, since the actual building is done with the port recipe. There is no need to specify items like MASTER_SITES, for example. What the wrapper port still needs to do:

- Specify the required items, such as port name and version number.
- Specify the dependencies. Not all of them need to be included, the recipe can also handle them. Including them in the wrapper port has the advantage that several tools will be able to find them.
- Specify a dependency on A-A-P itself, so that it will be installed when necessary.

```
# A-A-P port recipe for Vim
AAPVERSION =     1.0

PORTNAME =       vim
PORTVERSION =    6.1
MAINTAINER =     Bram@vim.org

CATEGORIES =     editors
PORTCOMMENT =    Vim - Vi IMproved, the text editor
PORTDESCR << EOF
This is the description for the Vim package.
A very nice editor indeed.
You can find all info on http://www.vim.org.
                 EOF

# Where to obtain an update of this recipe from.
AAPROOT =        http://www.a-a-p.org/vim
:recipe {refresh = $AAPROOT/main.aap}

WRKSRC =         vim61                          # Vim does not use vim-6.1
DEPENDS =        gtk>=1.2<2.0 | motif>=1.2      # GTK 2.0 does not work yet
BUILDPROG =      make

# This is used when CVS is available
CVSROOT ?=       :pserver:anonymous@cvs.vim.sf.net:/cvsroot/vim
CVSMODULES =     vim
CVSTAG =         vim-6-1-003

# This is used when CVS is not available or when disabled with "CVS=no".
MASTER_SITES =   ftp://ftp.vim.org/pub/vim
PATCH_SITES =    $MASTER_SITES/patches
DISTFILES =      unix/vim-6.1.tar.bz2
                 extra/vim-6.1-lang.tar.gz
PATCHFILES =     6.1.001 6.1.002

#>>> automatically inserted by "aap makesum" <<<
do-checksum:
     :checksum $DISTDIR/vim-6.1.tar.bz2 {md5 = 7fd0f915adc7c0dab89772884268b030}
     :checksum $DISTDIR/vim-6.1-lang.tar.gz {md5 = ed6742805866d11d6a28267330980ab1}
     :checksum $PATCHDISTDIR/6.1.001 {md5 = 97bdbe371953b9d25f006f8b58b53532}
     :checksum $PATCHDISTDIR/6.1.002 {md5 = f56455248658f019dcf3e2a56a470080}
#>>> end <<<
```

11. EXAMPLE

The example shows some of the A-A-P port recipe features. Most of the variables are the same or similar to the traditional ports. A few items deserve an explanation:

- AAPVERSION indicates the version of A-A-P this recipe was written for. When the version of A-A-P actually used is older it will produce an error. When it is newer it will behave like the indicated version would.
- PORTCOMMENT and PORTDESCR are included in the recipe. Only one file needs to be downloaded to obtain a port.
- The ":recipe" command specifies where an update of the port recipe is available. The command "aap refresh" will get it.
- DEPENDS specifies that either GTK or Motif is required, both for building and running Vim. For GTK the version must be 1.2 or higher, but below version 2.0. Motif version 1.2 and higher is accepted. If neither is currently installed the first package mentioned is installed, in this case GTK, with the highest acceptable version that can be found.
- Vim is configured and build with "make", this is specified by setting BUILDPROG. If configure would have to be run first a "pre-build" target is to be defined. This allows the port maintainer to perform the configuration exactly as he wants to, without the need to know about special variables.
- CVSROOT indicates the files are available through CVS. This is the preferred method to obtain the source files, because it includes all the latest patches. "CVS=no" can be used to disable using CVS.
- When CVS is not used the DISTFILES are downloaded. The checksums are also included in the port recipe. This is done by the port maintainer with the command "aap makesum".
- There is no list of installed files. It is generated automatically by doing a fake install and finding the files ending up in the fake root. For Vim this works as expected. For other applications it might be required to specify the files explicitly.

12. WILL IT WORK?

The big question is whether the proposed solution will actually catch on and a substantial number of ports will become available. Will A-A-P succeed where others have failed? The above text has explained that there is no fundamental showstopper. But the solution is not without disadvantages:

- Python is required. Not everybody likes it, the performance is less than with a C program and it is not a standard part on all systems.
- Yet another tool to learn to use.
- It does not solve the problems with packages.

There are many advantages:

- Using Python is much better than a mix of BSD make and shell script.
- No tricky solutions are needed, such as how a different master site list is selected by adding ":2" to the file name; the sites to be used for a file can be specified directly with an attribute.
- It is easy to use several versions of a port (stable, current, alpha).
- Only the actual install on the system needs to be done by root.
- Ports can work on many Unix systems.
- A-A-P is still under development, this provides the possibility of adding up all knowledge of existing ports systems. There is much freedom to specify the ports recipe format in a good way.

The proposed new ports system with A-A-P looks more attractive than other solutions. Especially the possibility to use it side by side to a traditional ports system, this allows users to try it out and get used to it. Still, whether it will attract a substantial audience remains unpredictable. When the A-A-P recipe is used for other purposes (developing and distributing software) it also becomes more likely that a ports system based on it will be successful. This should become clear the coming year.

An alternative for using A-A-P might be DarwinPorts [DarwinPorts]. This project also decided that a script language is needed to avoid the problems with Makefiles, they chose TCL. The file format looks more different from a traditional port than the A-A-P recipe, but not as much as Cons or SCons. What makes it interesting is that the "father of BSD ports" Jordan Hubbard is involved in DarwinPorts. However, it is still new and currently only working for Mac OS X 10.2. Support for FreeBSD is planned and the people behind OpenPackages recently expressed they will join with DarwinPorts. The main drawback of DarwinPorts is that it does not co-operate with the existing ports and packages systems. It registers installed packages in its own way, storing TCL procedures instead of shell scripts. Making the switch from the traditional package system to DarwinPorts will be difficult.

13. CONCLUSION AND CURRENT STATUS

The proposed solution is to create a new ports system with A-A-P. This system has enough similarities with the traditional ports systems to avoid a steep learning curve and at the same time offers many improvements. This solution does have a good chance of providing a united ports system for the BSD systems. The possibility to use it next to the existing ports systems avoids many of the problems that made other solutions fail.

A-A-P is still under development. Version 1.0 is expected spring 2003. The author of this paper will be working full-time on A-A-P. This means the project will not stall. The speed of developments will depend on contributions from others.

The A-A-P ports system currently works for a few examples. Before a large number of ports are to be made, the syntax of the port recipe must be ascertained. This requires that useful features from the various ports systems are included and the consistency of the result is checked. Before it can be used for stable systems a lot of testing is required. Thus there is still quite a lot of work to be done.

In between the writing of this paper and the presentation on the European BSD conference 2002 more progress will have been made, an update will be given in the presentation. Further progress will depend on reactions on this paper.

REFERENCES

[A-A-P]	The A-A-P project: http://www.A-A-P.org
[Asami]	Usenix 1999 presentation by Satoshi Asami: http://www.usenix.org/events/usenix99/full_papers/asami/asami.pdf http://people.freebsd.org/~asami/presen/usenix99/html/index.html
[Cons]	http://www.dsmit.com/cons/
[DarwinPorts]	http://www.opendarwin.org/projects/darwinports/
[Feyrer]	NetBSD packages growth compared to FreeBSD and OpenBSD, made by Hubert Feyrer: http://netbsd.org/Documentation/software/pkg-growth.html
[FreeBSD]	FreeBSD CVS log for bsd.port.mk: http://www.freebsd.org/cgi/cvsweb.cgi/ports/Mk/bsd.port.mk
[Hubbard]	DarwinPorts FAQ: http://www.opendarwin.org/projects/darwinports/faq.php
[Install Tools]	Overview of tools: http://www.A-A-P.org/tools_install.html
[NetBSD]	NetBSD CVS log for bsd.pkg.mk (long!): http://cvsweb.netbsd.org/bsdweb.cgi/pkgsrc/mk/bsd.pkg.mk
[OpenBSD]	OpenBSD CVS log for bsd.port.mk: http://www.openbsd.org/cgi-bin/cvsweb/ports/infrastructure/mk/bsd.port.mk
[OpenPackages]	http://www.openpackages.org/
[OpenPKG]	http://www.openpkg.org/
[SCons]	http://www.scons.org/
[Zoularis]	http://www.netbsd.org/zoularis/

RELEVANT LINKS

FreeBSD CVS log for ports/INDEX with Asami' s song texts:
http://www.freebsd.org/cgi/cvsweb.cgi/ports/INDEX
FreeBSD porters Handbook: http://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook
OpenBSD: "Building an OpenBSD port" http://www.openbsd.org/porting.html
OpenBSD: "Important differences from other BSD projects" http://www.openbsd.org/porting/diffs.html
NetBSD packages collection (pkgsrc): http://www.netbsd.org/Documentation/software/packages.html
NetBSD pkgsc documentation (well written, mentions differences from FreeBSD):
ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc/Packages.txt
NetBSD bsd.pkg.mk: ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc/mk/bsd.pkg.mk

BIOGRAPHY

Bram Moolenaar has worked on open-source software for more than ten years. He is mostly known as the creator of the text editor Vim. Currently he is working on a project called A-A-P, which is about creating, distributing and installing (open source) software. His background is in computer hardware, but these days mostly works on software. He still knows on which end to hold a soldering iron though. In the past he did inventions for digital copying machines, until open-source software became his full-time job. He likes travelling, and often visits a project in the south of Uganda. Bram founded the ICCF Holland foundation to help needy children there. His home site is www.moolenaar.net.