DebConf 8

MAR DEL PLATA - ARGENTINA

Proceedings of the 8th Debian Conference

debian

DebConf8 Proceedings Team

August 10th-16th, 2008

Preface

These proceedings contain a record of the technical and social events held during 2008 Debian Developers Meeting, DebConf 8, at Mar del Plata (Argentina), an open event aimed to improve communication between everyone involved in Debian development.

This document has been arranged by the DebConf8 Proceedings Team, on behalf of the DebConf8 Organization Team. The authorship, copyright and licensing information of each article is specified in the proper chapter.

In addition to a full schedule of technical, social and policy talks, DebConf provides an opportunity for developers, contributors and other interested people to meet in person and work together more closely. It has taken place annually since 2000 in locations as varied as Canada, Finland and Mexico. Debian Conferences have featured speakers from around the world. They have also been extremely beneficial for developing key Debian software components, including the new Debian Installer, and for improving Debian's internationalization.

More information about DebConf8 and the Debian Developers Meeting can be found on the DebConf website at `http://www.debconf.org/`.

Contents

Chapter 1

Knowledge, Power and free Beer

Andreas Tille

Abstract

Sometimes the meaning of *free* remains vague and unclear for outsiders. There seem to be different prices for something that is free which is confusing. The talk tries to find an answer to the question: What the hell will I get if I buy a box labelled "Debian GNU/Linux"?

1.1 Free Beer

1.1.1 Hard and soft Ware

Hard Ware

In the field of informatics we differentiate between hardware and software. What does these terms mean?

First lets think about the term ware in general. A key feature of a ware - which is something which is selled by a vendor - is its weight. Very often the price of the ware is proportional to this weight. Both parties - vendor and vendee are able to do certain things like biting on the thing or throbbing at it or whatever.

All these methods might be useful to check the quality and thus the value of the ware to buy - or just for fun. In any case the buyer is able to check whether the ware fits his expectations before handing out the money while the seller is able to prove that the features of the thing justifies the price.

The term computer hardware perfectly fits this features. It comprises all of the physical parts of a computer, as distinguished from the software that provides instructions for the hardware to accomplish tasks.

Once the vendee has bought the ware he deserves certain rights on it. He is now the owner of this ware (provided he has paid the full price he and the vendor agreed to). The person who owns something has the pristine right to alter, change or disassemble the thing. If the ware shows some hidden problems, has some faults or gets broken in a certain time the vendor has to guarantee for the promised features. Finally the owner has the right to sale again the ware that was bought some time before.

As a rule duplication of a ware is not worth the effort because the professional manufacturer mostly can do it much cheaper.

Soft Ware

The features of software are quite different. In fact the question comes up whether it really deserves the term "ware". The main feature of weight is missing completely because defining a weight for software makes absolutely no sense. Moreover the person who is about to buy a certain software is barely able to check it completely as it is possible with any ware. Even if there are some vending models to use a software for a limited time before buying it. But these days software is this complex that the verification and comparison of competing products is sheer impossible (compared to find out which salesman trades fresh fruits at the market place).

But even if the buyer payed the price for the software the relation to the thing which was payed is quite different than for a piece of hardware which was discussed above. This becomes clear when somebody would take the time to read the strange text which is always displayed of the installation routine of proprietary software. There must be a reason why this hard to understand text which sounds very juridical is bothering the user who wants just to install a piece of software on the computer. But most users have learned that it works perfectly to just press the `''Next''`-button while ignoring the text.

This behaviour prevents users from recognising that they do not really own what they payed money for, they just got a license to use it, sometimes applying some additional restrictions like that the software can only be used on this specific computer and will only work if you send in some data of your computer to get a license key. In case of a piece of hardware – say a chair – this would mean you pay for the chair but you are only allowed to use it in a certain room next to a specific table and only at the gable end; for the other side a different chair with a license to use at this place is needed.

Moreover the license of proprietary software does not allow to change anything. Sticking to the chair-example, this would mean the user is not allowed to paint it with a different colour. Last but not least selling the software is often not allowed. The reason is obvious: The user just does not own the software - the money the user spend on it was for a license to use but not to sell.

Besides this completely different relation between the owner of a piece of hardware and the license owner of some software there is a further difference: In principle it would not be hard to duplicate software. That's why this is not only strictly forbidden but also prevented by technical means.

Fresh fish

Around 50 BC at the market place of a small village in northwest Armorica the fishmonger *Unhygienix* is infamous for his rotting products. This often causes trouble with *Fulliautomatix* and other inhabitants of the village. This means selling bad smelling fish at the market place is nearly impossible. Each customer would immediately notice that it is to old.

But what about software?

The customer has no sign about how old the code of proprietary software might be. It has no odour or other signs of age which are easily to recognise. Moreover software usually consist of several parts which are developed at different times. It frequently happens that the major parts are quite old and only new features and user interface is really fresh.

But what is the problem here?

If somebody pays for a license a certain amount of money only a fraction is payed for really new code. A larger part of the money is spend for code which yet was paid in former versions. But former versions should not cost money. Users just got the license to use this code for years. But what are the stinky bits in the whole software bundle?

Lets recapitulate: Users are perfectly willing to pay for the work of the programmers. It is a fair deal to pay for a service like software development. But the deal is only fair in the case that the user is able to verify how much time was spent to produce this piece of software which was not payed before.

1.1.2 Patent recipes

The progress

The progress bar is something which keeps impatient users happy. It is the way of programs to inform users that some business is going on and give hints how long this business will last. It is no rocket science. If you see a 100 m sprint you can clearly see start and goal and the runners move like a progress bar from the beginning to the end. So a progress bar can also be viewed as a analogue to real live procedures which are just visualised at the computer monitor. It is hard to tell what the technical invention in the progress bar might be so it hardly can be patented.

But there is `Patent No. EP0394160; Applicant(s):  IBM (US)` which exactly describes the progress bar.

The next example for a so called "technical invention" might be another illustration of the issue: It seems reasonable to store different versions of a document in that way that you include the date in the filename. Provided that this date-tagged file archive is stored on a network drive which might be accessible to colleagues than this “technique” infringes `Patent No. DE10108564` which is just valid in Europe.

So what is the issue in declaring common sense as technical invention?

Big software companies acquire a large portfolio of patents - reasonable but mostly such strange things as mentioned above. In case they want to use a technique which is patented by another big company the deal is as follows: Allow me to use your technique and I allow you to use my technique for free. So a large amount of patents is kind of an insurance not to become sued by a competitor. This sounds very reasonable from the point of big software companies and is the driving force behind them to register as many as possible patents.

But what about smaller software companies that do not have this kind of insurance. Or even worse what happens to the Free Software programmer who has not even the slightest idea that his idea he had from common sense is patented by someone else?

EU: Council versus Parliament

This talk is about Free Software. It is not about politics. But unfortunately the topic of software patents in Europe is a very political topic. Unfortunately not many people do care about this. So it remains mostly not regarded if representatives of national parliaments vote for different things than the national parliament has instructed them to vote in the Council of Europe. Less people notice if the rules of democracy are broken for the sake of big software companies. But this would be material for one or two separate talks.

So here is some homework for the interested reader. Just ask your favourite search engine for the following questions:

1. What is the difference between the Council and the Parliament of Europe?

2. Which panel brought up the issue of software patents and what is the suspicion who is behind this issue?

3. What do we owe the representatives from Poland in the Council of Europe?

4. If you are not from Poland what role played the representatives from your own country in the Council?

5. Which panel has the chance to stop the efforts in the wrong direction.

If you are able to answer all these questions you can go straight to your representative which you voted for the European Parliament and ask him to vote against software patents.

1.1.3 Free of charge

Soft ware is not for sale

In the beginning it was stated that software can hardly be regarded as a ware. This has the consequence that it is hard to sell it for a fair price. There are many people who get this idea but came to different conclusions. Common to these conclusion is the fact that they give away their software at no cost. But what are the differences.

Download here The user can download the ready to run executable. For most users this is fine. They want a program which just does a certain job and do not want to pay money. This is kind of "free beer" - software. You go to the restaurant to get drunk without spending money - free beer works perfectly for this purpose.

But is this really freedom? Does this program do what the user expects it to do? Isn't it just a Trojan horse which spies out user data? Can the user be really sure that it is not harmful?

It just happened that people mixed something into the free beer ...

Do not change The author of a piece of software allows the user to have a look at the code. There might be several conditions under which the user is allowed to look at the code but assume here that having a view on the code and using the program is free of charge.

This at least might ensure the user that the code does not contain any harmful bits which is not the case in the *free beer* flavour of free of charge software.

Free Software Free Software is according to its definition not only free of charge, it also allows to use the code in complete freedom which means the user is allowed to change the code for his own purpose and can even give away the changed version.

If there is such a lot of software free of charge the world must be a better place. So a normal user might wonder: Where's the catch?

The catch: Free $\neq$ free of charge

The term "Free Software" does not say anything about the money a user has to spend to solve a certain task with this very peace of software. The crucial thing is just that the code of the software can be used without any restrictions by anybody. The fact that the code can be used by anybody free of charge does not implicitely mean that it is really useful to solve a certain task immediately. Any piece of software needs adaptations, maintainance and service. This does not come for free.

The *free* in “Free Software” is more in the sense as “free speech” instead of free of charge. Free speech requires time to write out. It needs competence about the topic and sometimes free speech requires courage. All these are values which can not be countervailed with money – but they are certainly worth anything. So there is either some cost of time for the user in person or the user has to pay anybody to do the job.

The exact definition of Free Software is given in the Debian Free Software Guidelines.

1.2 Knowledge

1.2.1 Alternatives

WikiPedia

If there is anything in the world which is easily comparable to the fascination of Free Software by sharing many ideas in common than it is free knowledge. The goal to collect the whole knowledge of mankind for the whole mankind for free is the goal of WikiPedia for short.

If you are an expert in any field or think you know something special just test the quality of this community effort driven encyclopedia: Go to wikipedia.org and look up this special field. There are two basic options:

1. The contents is well written and is right according to your knowledge. Just be happy that you found a new information source and continue checking the contents of other fields if they represent your own knowledge.

2. The contents is not well written, is not precise or even completely wrong - you just are not happy with the contents. There might be people who just surf away and use other sources of information while ignoring this WikiPedia rubbish.

   But once you proceeded through this paper up to this point you hopefully behave differently. You learned something. You just recognised the fact that nothing comes for free. Take a little bit of your spare time and bring your knowledge in precise and well formed sentences and just make WikiPedia better. Your fellow WikiPedia writers will be happy about this and other readers will be even more happy from now on once they stumble over the page which was not satisfying before your editing.

WikiPedia is taken honestly

Figure 1.1 shows a quite interesting spam mail. Well, not the contents itself is interesting, but the fact that spammers do refer WikiPedia to “prove” that

```
Subject: Wikipedia knows the shit
Date: Sun, 27 Mar 2005 16:18:36 -0200
From: Christian
<VPYLGHQ@interacti.net>
To: aids-std@rki.de

Newest penis enlargement system:
- Increases the penis length and girth
- Medically Proven (source: wikipedia)
- No Surgery
- Permanent Results
- Proven Traction Method
- 100% Satisfaction Guaranteed

Find more info here: ...
```

Figure 1.1: Spam mail referring WikiPedia

their rubbish is worth anything. This is kind of: If your enemy takes you honest you must be somebody.

On the other hand it reveals the most common biased opinion about WikiPedia: If *anybody* can write and change articles how can be assured that nobody writes something which is wrong on purpose. (This effect is called "vandalism" in the WikiPedia slang.) The answer is: Many people are watching.

There is a log of every single change of each single page. Reverting some change is done quite quickly and thus the effort of vandalism is high compared to the effort which would be needed to keep WikiPedia clean.

Assume the dirty spammer which sended the spam mail which is displayed in figure 1.1 had edited WikiPedia first before sending out the strange mail. Even if this person would have done this – it is not in WikiPedia any more. This is one example that the auditing process of the WikiPedia activists is working effectively and even if there is vandalism it does not seem to have a major influenze on the whole collection of free knowledge.

Last but not least there is some solution for the boring spam problem: In figure 1.2 the output of this spam filter showes that it catched the beast – a nice example for a useful peace of Free Software.

```
SpamAssassin:  17.1 points, 5.0 required
 pts  rule name   description
----------------------------------------------------------
 0.1  SATIS_GUAR  BODY: Mail guarantees satisfaction
  17  BAYES_99    BODY: Bayesian spam probability is 99 to 100%
```

Figure 1.2: SpamAssassin handled spam mail

1.2.2 Free Software in general

As You Like it

What is the basic need of users of any piece of software? What do users really want to solve a certain problem?

A steady and continuous functionality over long period is the main demand of software users. The program has to be safe against failures to not interrupt the work. It has to have a certain set of functions which are necessary to solve a certain task and it has to provide this functionality for a long period in which the task in question has to be solved. Any upgrades of the software do require new learning which just takes time which is cut of from the time span which is reserved for the main task of the user. It is a common missconception that users always want the latest and greatest version of any software project. A further missconception is to overload programs with more and more functions: The more functions a program has the harder ist will be for the user to recognise which functions are really helpful to solve the task in question.

If these requirements are fullfilled the user does not really matter about the concrete program which is finally choosen to solve a certain task. A user simply wants to solve a task and does not want to bother about the philosophy of the tool that is used for it. Just ask your local carpenter whether he cares about the wood which was used to make a handle for the sledge he is using. Most carpenters will not care whether this piece of wood comes from a forest which should be saved or from anywhere else. So users just want a functionality.

In the case of software functionality is not just the program. It is more than that: it is the program and the service which is provided for it. Users have to decide how much money they want to spend for the given task - usually there is some budget for any task. So if there is a given budget and the program itself comes for free there is more money left for the service. Usually service is quite expensive these days so it makes sense to spend more on this side of the calcualtion.

So far to the advantages from the users point of view. But what about the producer of a piece of software? What would a software company loose if they would release their code which proprietary software companies keep as a secret. There are several reasons which speak for releasing the code as Free Software. It is just the problem that everybody thinks it is impossible because selling proprietary binary code is kind of normal these days. But it is really the most effective form to to make money in software business?

At first there are software companies which make all their money based on Free Software like Red Hat. At second there are companies that see their future in releasing at least parts of their code as Free Software. Well known examples are Novellor even IBM. The rationale behind this is quite simple: The proprietary software market is currently characterised by a hard competition. If a company recognises that it is impossible to beat the competitor it tries to drift to a new field. In several cases this leaded to a release of code as Free Software (Mozilla, OpenOffice, Interbase, MaxDB, E-Directory, ...) and providing service for this software (see above for the relation between functionality service and program code).

As a matter of fact providing services for Free Software is simply cheaper than if the code is not available. It is quite common that there are always some users of a certain piece of software that are keen on trying to solve a problem they have themselves. In case they have understand who Free Software works – sharing ideas, work together, join forces – they provide solutions or fix bugs in the software for free. They just have learned that if they provide their work to the authors the next version will contain their code and will make their own work easier because the program has an enhanced functionality. And this is the real clue: The authors of a piece of Free Software get freelancer for no extra cost. The driving force behind these freelancers who volunteer to provide enhancements is that they need a certain functionality to solve a task. They want to spend this time only once and thus they submit their work to the authors. If the authors are part of a company which provide services for this software in this way they got some freelancers at no cost. Compared to external freelancers for proprietary code which requires hiring educated programmers for a lot of money, signing discretion contracts while being unsure whether they do the same for a competitor etc. the Free Software approach has an extraordinary financial advantage for the authors of the code.

The crux hereby is that you need a wide user base because only a small amount of users will really provide reasonable input. On the one hand this is the case just because in most cases users will not be gifted programmers on the other hand users might simply have not understand the principle of Free Software. They regard software as a ready product which just has to be accepted with all flaws it might have and with all problems it might cause when trying to do the tasks a user want to do. To solve the latter problem texts like this one are written.

A wide user base to attract gifted programmers is normally no problem in case of pieces general software which is needed by every user. So finding programmers for an operating system kernel (like Linux), a web browser (like Mozilla), an office suite (like OpenOffice) etc. works perfectly. The problem changes in case of specialised software which is naturally used by only specific user like people working in certain professions.

Get the facts

When the main global player in software business recognised that there is something else which is penetrating the software market. After a certain time they started a campaign title: “Get the facts”. They paid for doing studies which should prove that proprietary software is better than Free Software. No matter what the result of these studies might have been it shows one important thing: They would not pay for studies against somebody who is not taken honest.

BTW, it turned out that a certain amount of the facts was quite questionable. . .

1.3 Power

1.3.1 Distribution

Linux from scratch

The usual way a user obtains a software bundle is when buying a piece of hardware. This makes perfectly sense because computer hardware is completely useless without software. Usually a computer comes with a proprietary operating system and some additional proprietary programs to solve simple office tasks.

Assumed a user have heard about the existence of Free Software and decided to try it on the new computer. The reasons might vary from the intent to save money (and buying a computer without any software) to just being not happy with the software which would usually come bundled with a computer.

The first thing which is needed is a Kernel of an operating system. There are more than one option but Linux is the most popular choice. In principle the way to obtain a piece of Free Software is to go to the web page where the source is provided, find a compiler which is able to compile this source and install the program that was builded on the machine. It will take a certain amount of time to install the Linux Kernel and some basic tools which make the “GNU/Linux” operating system. This is described in the Linux from scratch project.

The modular nature of Free Software implicitly means that at this point the user does not have a shiny colourful desktop but just a command line interface. The next step would be to get the source and compile the X Window System which is not really a simple task for a beginner. If our user managed the work up to this point he reaches a crossroad: The religion of the desktop environment. The main roads are marked with the signposts “Gnome” and “KDE” but these are not the only ones. The user fails to find a reasonable decision and flips a coin which road to follow. From this point on the screen of the computer looks nice – but there is no program which enables the user to do productive work.

Our brave user now has heard about two shiny new projects: Firefox and Thunderbird. He finally manages to get these programs installed on the computer and can do the first tasks: Browsing the web and managing e-mails. Even if this owner of the new computer started very optimistic with the intent to install Free Software recognises that it was quite a large amount of work to reach this step.

Once it comes to more complex user applications like a comfortable office suite like Openoffice.org, a professional type setting system like $\LaTeX$ or a feature rich image manipulating program like Gimp the user is really tired.

If it finally comes to some server software like a web server and a database server the owner of the computer is completely tired. So many decisions to choose from, so many web sites to go - what is the sense to spend so much time?

Distributions on the market

The answer is: Nobody really wants to spend so much time just to exercise the possibility of Free Software: Do it yourself from the source to the installation for every single piece of software. It is not only time consuming it is even error prone to build a complete system. It just should be done by experts.

A complete system containing all the bits of Free Software mentioned above and much more is called *distribution*. It is builded by experts that know the code, use sophisticated tools to build the software and prepare it to be installed quickly onto users machine. This preparation is done in so called *packages* that contain compiled code of the Free Software projects and some preconfiguration to work together with other components of the whole system. Building a distribution from Free Software is kind of a service. Users may decide whether they want to spend days for doing it all on their own or just pay a certain amount of money to buy a distribution. Most users decide it is worth spending some money for a distribution.

But this uncovers some misunderstanding in the relation of Free Software and money: While (at least most) bits of the distribution are free of charge the whole thing costs a certain amount of money. This is one flavour of the "a thing is more than the sum of its parts" principle. The price which is payed is for the service inside and users pay what they regard worth the money. They just have to sum up the amount of money they could gain with productive work in the same time when doing all things on their own. As a side note it should be mentioned that there are also distributions available for no charge but this is explained in detail below.

It has made clear now why users go to a software store and buy a distribution of Free Software. They buy a certain box which has printed in bold letters: “**XYZ** Linux version **A.B**” where **XYZ** stands for the company that compiled the distribution and **A.B** for the version number the distributor has chosen to mark this release state of the whole bundle of software. New users tend to ignore the fact that this is by no means the version number of Linux which is just the kernel of the operating system and currently has a version number below 3. So if somebody is telling you that he is using “Linux version 8.x or 9.z” he has apparently a misconception about what Linux is and what he really bought in the software store. To express this more clearly Free Software enthusiasts speak from “GNU/Linux” because the complete system is more than just the Linux kernel but the many other Free Software tools around which finally make an operating system.

But once we come to colourful boxes with bold letters on it we are back to the commercial world of market place with pressure of competition and gaining market share. It is a known fact that selling GNU/Linux distribution is a normal business where all the rules apply that are valid for vendors of proprietary software. If a distribution company is not able to make profit it will not be able to survive. The good news is that there is a certain number of companies that survive perfectly since several years. Why is this good news? It just proves that it is perfectly doable to run a business on top of Free Software - which is free of charge in its pieces.

But how to compete with other distributors that are just compiling the same basic set of Free Software applications. There are several distinctions between distributors. One is the local aspect. So if a user from Northern America speaks about his new Linux 5 (see above) he is talking about his copy of RedHat Linux while a user in Germany has not really a more recent version of the Linux Kernel if he is talking about his new Linux 10 because he is referring to his copy of SuSE Linux which has just a different version number scheme. People in France prefer Mandrake Linux and Turbo Linux is the main player in Asia – except for China where RedFlag Linux seems to be used mostly.

To make the jungle of distributions even more impenetrably there are distributions that target more to the general desktop user with the latest office applications and providing interfaces to proprietary applications which normally run on other operating systems using emulators (like Wine or DosEmu). Users are free to pick the distribution that fits their needs best. They have a real choice which they do not really have in the case of proprietary systems because there is basically one for a given hardware platform.

Users have a chance to identify themselves with their distribution of choice and who ever have heard over a flaming discussion between users of different distributions knows what competition means.

The missing link

As it was explained in section 1.2.2 Free Software works most effective if there is a wide user base. That's why also distributors decide to include these Free Software projects that are potentially used by many users. From a distributors point of view this makes perfectly sense because serving a mass market implies the distribution will be bought potentially by a high number of people which keeps the business running. It would be wasteful for the distributor to employ specialists for preparing specialised software of certain fields to sparse specialists the work to find out which software fits their needs and how to install this software.

For a moment this sounds fair enough to build a solid base of often needed applications and leave the remaining things for those people who really need it. But the problem is that because applications of special fields do not have a wide user base and thus did not developed so far that failure prove installation methods and easy updating mechanisms are well developed which enable people who are no computer experts to easily proceed with the task to install this software on their computer. The problem becomes worse if the applications require a complex database or web server setup.

One solution would be that small service providing companies take over this job. It is quite common that the authors of a certain piece of specialised Free Software provide their code for free and sell the service of installation, adaptation, documentation and maintenance.

The other solution is that a group of experts builds a complete distribution by just moving the Free Software development principle to distribution level. In fact this is done and it got the name Debian.

1.3.2 Debian - a distribution builded by a community

Maintainers are experts

In contrast to the GNU/Linux distributions mentioned above Debian is not a company but an organisation. The goal of this organisation is not to sell the distribution. Debian sells nothing. It just produces a free distribution. The people behind Debian who are united in this organisation have just a common goal: To build the best operating system they can afford. While this goal sounds simple the motivation for doing this seems to be missing. While commercial distributors try to earn money Debian people seem to work just for fun.

Well, sometimes it is real fun but there are stronger reasons for the Debian people to work on their common goal: They just get what they need for their own work. They know that they can trust their system and users appreciate their work. The best appreciation users could express is by just providing informations about problems they see, suggesting solutions for these problems or even sending in code which just has to be applied to make Debian better.

But how does this particularly work? The developers of Debian are volunteers who have a special interest in at least one special piece of Free Software. A developer becomes a so called maintainer of a package by just doing the usual work all distributors are doing: Obtain the source of the software from the download area of the software, compile the software and build packages which are easy to install. This is so far no difference to other distributors (note: the packaging format might be different but this is really a technical matter). The difference here is that this maintainer becomes on the one hand the first user of this package – so he is interested personally – and he is the first person to ask if users would run into trouble with this package. It becomes clear that the maintainer is kind of the missing link between the authors of a Free Software project and the final user.

The maintainer is probably one of the most qualified person for this special application inside Debian - and the users know his name. For complex applications usually there is even a group of maintainers who care for larger projects. But each of them knows about the problems that the application might cause in the installation process and tries to solve these with inside scripts and configuration files that reduce the work for the final user to a minimum.

Special applications

But what about the special applications issue which was not solved by commercial distributors who are only able to provide a quite general system? This problem is solved inside Debian because also Debian maintainers work in special fields and need special applications. So if a maintainer works in the field of medicine he will care for the medical applications he needs. Teachers can become Debian maintainers and will probably care for applications that can be used in their schools for education etc. In Debian this principle is called *"DoOcracy"* = the doer decides.

So it turned out that Debian became the largest collection of ready to run Free Software in the internet, containing not only general applications but also programs which only specialists need. They are provided in a way that users can easily install them on their computer. Users of special applications will not be troubled with extra efforts like compiling the source themselves and follow a complicated install procedure.

Lost in the jungle of applications

The largest collection of ready to run Free Software – this sounds great for marketing experts and makes a quite good statistics. But what would this mean for the unexperienced user who just started using Debian and finished the first steps of the Debian installer? Should this poor user start reading the descriptions of 18,000 packages where at best 50% can be understand by a newcomer?

If this would really be the case the applications that are really interesting for the user would be hidden perfectly and it would be nearly the same as if they would be not packaged at all. But there is a light at the horizon.

Custom Debian Distributions

Custom Debian Distributions are completely integrated into Debian. It is a technique which enables a Debian user to focus on these parts of Debian which are really interesting to solve a special task. It is implemented in so called "meta packages" which are simply spoken packages which can only be installed if a set of other packages is installed at the same time. The packaging mechanisms inside Debian care for everything if the user tries to install a meta package.

So in case a user is a teacher and wants to install a computer lab with educational software for the students, focussing to the DebianEduCustom Debian Distribution is the best idea. In this special case the situation has developed that far that there is even a special CD to install Debian-Edu also called SkoleLinux and the whole Debian system is not really needed, but in principle everything which is on the SkoleLinux CD is included in Debian.

Another example is DebianMed. It was prepared for people working in the field of health care. There are a lot of applications that are used for DNA or protein sequences inside Debian. All of these will be installed when one single package called *med-bio* is installed. The same procedure for medical imaging applications: The installation of *med-imaging* causes the installation of all packages that are relevant for users who have to deal with medical imaging tasks.

This technique dispenses users that are working in special fields from the task to do researches which applications do exist and which are relevant for their own task. The packages will not only be installed at this computer but they also get a menu entry which is made visible in a special user sub menu for easy access to the programs in question.

Conclusion

In the first it was explained why Free Software exists and that "free" in this sense does not really mean "free of charge" as free beer. In the second part also free knowledge (WikiPedia) was explained and things people should know when dealing with Free Software and their relation to proprietary software were mentioned. Finally the third part explained which power is behind Free Software if it is used sanely in a distribution which fits the needs of users.

Chapter 2

Solving Package Dependencies: From EDOS to Mancoosi

Ralf Treinen and Stefano Zacchiroli
Laboratoire Preuves, Programmes et Systèmes
Université Paris Diderot, Paris, France
{`treinen,zack`}@{`pps.jussieu.fr,debian.org`}

Abstract

[1] Mancoosi (Managing the Complexity of the Open Source Infrastructure) is an ongoing research project funded by the European Union for addressing some of the challenges related to the "upgrade problem" of interdependent software components of which Debian packages are prototypical examples.

Mancoosi is the natural continuation of the EDOS project which has already contributed tools for distribution-wide quality assurance in Debian and other GNU/Linux distributions. The consortium behind the project consists of several European public and private research institutions as well as some commercial GNU/Linux distributions from Europe and South America. Debian is represented by a small group of Debian Developers who are working in the ranks of the involved universities to drive and integrate back achievements into Debian.

This paper presents relevant results from EDOS in dependency management and gives an overview of the Mancoosi project and its objectives, with a particular focus on the prospective benefits for Debian.

[1]The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°214898.

2.1 Introduction

Building and maintaining a free software distribution is a challenging task. A user expects to be able to install any selection of packages from the distribution on his machine, and that the installation goes smoothly and results in a working system with the desired functionality. Any requirement, for instance the need of installing certain auxiliary packages from the distribution, should be detected by the tools coming with the distribution, and should be satisfied automatically whatever packages the user wishes to install. Incompatibilities in user wishes should be detected and reported back to the user with a satisfying explanation. Software is expected to be readily available in its latest version, of course well-tested without any bugs or any remaining incompatibilities with other software components. All this is expected to work smoothly on a wide range of architectures and system configurations.

It is the task of a package maintainer to do her best to satisfy these expectations. Luckily, a maintainer has at her disposition a sophisticated infrastructure, a knowledge base of policies and best practices, and the support of her fellow developers. On the other hand the maintainer is also faced with upstream authors who usually have their own ideas about how their software is supposed to be compiled, or how it should interact with the rest of the system.

The EDOS research project (for *Environment for the development and Distribution of Open Source software*) had the objective of coming to help and to provide FOSS distributions with better tools to help them do their job. The project was funded by the European Commission under the IST (*Information Society Technologies*) activities of the 6th Framework Programme. Besides several public research institutions from different European countries and some small enterprises in the FOSS business there were two commercial GNU/Linux distributions in the project: Mandriva from France who is building one of the most popular RPM-based distributions, and Caixa Mágica from Portugal who is well-known in Portuguese-speaking countries. This distribution is again RPM-based, and also upstream author of the `apt RPM` tool. For the successor project Mancoosi (for *Managing the Complexity of the Open Source Infrastructure*) Pixart from Argentina joined in with its Debian-based distribution. EDOS started in October 2004 and ended in June 2007. Mancoosi started in February 2008 for a duration of 3 years.

The EDOS project was relatively broad in scope and had workpackages on the following subjects:

- formal management of software dependencies
- flexible testing framework
- peer-to-peer content dissemination system
- metrics and evaluation

We will in this paper let the last three of these workpackages aside since the authors haven't been involved in these, and present from EDOS only the workpackage on dependency management. We decided to focus on the problem of distribution coherence from the release manager's point of view, and therein on one basic question: Is it possible, for a given user selection of packages, to install these when only the packages from this repository are available? We were only taking into account package relationships that are expressed by the metadata of packages (that is in Debian: the `control` file). Relevant results and applications for Debian will be presented in Section 2.2.

The successor project Mancoosi again has several workpackages. The stream on dependency management takes off where EDOS has ended and tries to extend our previous results to build better tools for the system administrator who wants to perform a system upgrade or package installation on a real system. More about this will be discussed in Section 2.3.

EDOS has developed its own terminology which Mancoosi continues to use:

Installer A tool to unpack and configure, upgrade, or remove a locally available package on a local system. In Debian: `dpkg`.

Meta-Installer A tool to resolve (higher level) user requests of installing, upgrading, or removing packages on a system. This tool will have to access possibly remote packages repositories, and construct a sequence of commands for an installer. In Debian: `apt-get`, `aptitude`, `dselect`.

Metadata of a package is the data that can be statically (that is, without performing an actual installation) extracted from a package. In case of Debian this is the contents of a packages `control` file, which flows into APT package lists (`Packages` and `Sources`).

2.2 The Past: EDOS

2.2.1 Formalization of Inter-Package Relations

One of the first objectives of the EDOS project was to establish a simple mathematical model of a (GNU/Linux) distribution. We decided to restrict ourselves in the context of EDOS to relations between packages as they are seen by a meta-installer. Though the model is general enough to describe the essential features of common packaging systems (in particular Debian and RPM) we will focus in the following on the modeling of the package relations as found in Debian.

The Debian policy lists different possible relations between binary packages: Depends, Recommends, Suggests, Pre-Depends, Enhances, and Conflicts. The Replaces relation concerns only the installer (not the meta-installer), and the same seems to be true for the Breaks relation (which wasn't included

```
Package:& a              & Package:& a
Version:& 1              & Version:& 1
Depends:& b, c|d($>$=2) &    Depends:& b(=2)|b(=3),
&&&c(=3)|d(=2)|d(=3)

Package:& b              & Package:& b
Version:& 2              & Version:& 2

Package:& b              & Package:& b
Version:& 3              & Version:& 3

Package:& c              & Package:& c
Version:& 3              & Version:& 3
Conflicts:& b            & Conflicts:& b(=2),b(=3)

Package:& d              & Package:& d
Version:& 1            & Version:& 1

Package:& d              & Package:& d
Version:& 2            & Version:& 2

Package:& d              & Package:&d
Version:& 3            & Version:& 3
```

Figure 2.1: A distribution (to the left) and its expansion (to the right).

in policy anyway at the time of the EDOS project). Relations between source packages and binary packages are not of interest for us. However, we have to take into account Provides (that is, virtual packages), and the fact that relations may be disjunctive (e.g., `a|b|c`), and may be qualified by constraints on version numbers.

We decided to ignore relations that are not essential for a meta-installer in order to decide about installability. This eliminates Suggests and Enhances from our list of interesting relations, and we also decided to ignore Recommends relations. Pre-Depends can for our purposes be identified with Depends.

This leaves us with Depends and Conflicts. The next question was how to handle constraints on version numbers like `>= 1:2.3.4-5`. We decided to not complicate our model with version numbers and their comparison, and to expand version constraints: given a package in a package dependency we replace it by the disjunction of all versions of that package that exist in the current distribution. In case of a conflict we replace the package by the set of all versions of that package. An example of that expansion is given in Figure 2.1.

```
Package:& a                 & Package:& a
Provides:& v
&                           & Package:& b
Package:& b                 & Depends:& w
Provides:& v
Depends:& w                 & Package:& v
&                           & Depends:& a|b

Package:& c                 & Package:& c
Provides:& w                & Conflicts:& d
Conflicts:& w
&                           & Package:& d
Package:& d                 & Conflicts:& c
Provides:& w
Conflicts:& w               & Package:& w
&                           & Depends:& c|d
```

Figure 2.2: A distribution involving virtual packages (to the left) and its expansion (to the right). Version numbers are omitted.

This expansion has the advantage that we get rid of constraints on version numbers, but it has the drawback that this expansion is always relative to a set of available packages. This might pose a problem when one wants to make the expansion incremental. For instance, if the original distribution is extended by a new version 4 of package `d` we would have to reconsider in the expansion all packages that have a relation to `d`. In our example, that means that we have to change the Depends line of package `a` and add `|d(=4)`.

Expansion also introduces explicitly the virtual package which depends on all packages that provide it. Special care has to be taken with conflicts on virtual packages as a package may at the same time provide a virtual package and conflict with it. Section 7.4 of the Debian policy states that in this case the package conflicts with each package providing that virtual package, with the exception that the package doesn't conflict with itself. An example of an expansion involving virtual packages is given in Figure 2.2.

We can now state the formal definition of a package and a repository:

Definition 1 *A* package *is pair consisting of a name and a version number.*

Note that we have not defined what package names and version numbers are, it suffices for us that we can know when two names or version numbers are equal (as we assume that we are working with an expanded repository).

Definition 2 *A* repository *is a tuple $R = (P, D, C)$ where P is a set of packages, $D : P \to \mathcal{P}(\mathcal{P}(P))$ is the dependency function (we write $\mathcal{P}(X)$ for the set of subsets of X), and $C \subseteq P \times P$ is the conflict relation. The repository must satisfy the following conditions:*

- *The relation C is symmetric, i.e., $(\pi_1, \pi_2) \in C$ if and only if $(\pi_2, \pi_1) \in C$ for all $\pi_1, \pi_2 \in P$.*
- *Two packages with the same name but different versions conflict, that is, if $\pi_1 = (u, v_1)$ and $\pi_2 = (u, v_2)$ with $v_1 \neq v_2$, then $(\pi_1, \pi_2) \in C$.*

In this definition, the function D yields for any package the set of all its dependencies. All these dependencies must be satisfied simultaneously. If any such dependency is a set with more than one element than this set is understood as a set of alternatives. The last restriction, stating that two different versions of the same package are in an implicit conflict, is specific to Debian (RPM does note have this *a priori* restriction).

It is now straightforward to translate an expanded `Packages` file into a repository according to Definition 2. For the expanded `Packages` file on the right of Figure 2.1, for example, we obtain (P, D, C) as follows:

$$\begin{array}{rcl}
P & = & \{(a,1),(b,2),(b,3),(c,3),(d,1),(d,2),(d,3)\} \\
D(a,1) & = & \{\{(b,2),(b,3)\},\{(c,3),(d,2),(d,3)\}\} \\
D(b,2) & = & \emptyset \\
 & \cdots & \\
C & = & \{((b,2),(b,3)),((b,3),(b,2)),((c,3),(b,2)),((b,2),(c,3)),\ldots\}
\end{array}$$

Definition 3 *An* installation *of a repository $R = (P, D, C)$ is a subset I of P, giving the set of packages installed on a system. An installation is* healthy *when the following conditions hold:*

- ***Abundance:*** *Every package has what it needs. Formally, for every $\pi \in I$, and for every dependency $d \in D(\pi)$ we have $I \cap d \neq \emptyset$.*
- ***Peace:*** *No two packages conflict. Formally, $(I \times I) \cap C = \emptyset$.*

Definition 4 *A package π of a repository R is* installable *if there exists a healthy installation I such that $\pi \in I$. Similarly, a set of packages Π of R is* co-installable *if there exists a healthy installation I such that $\Pi \subseteq I$.*

Note that because of conflicts, every member of a set $X \subseteq P$ may be installable without the set X being co-installable. One can even show that not co-installable sets of minimal size can be arbitrary large: Let, for a given number n, R_n be the following repository:

$$\begin{array}{rcl} P & = & \{a_1, \ldots, a_n, b_1, \ldots, b_n\} \\ D(a_i) & = & \{\{b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n\}\} \\ D(b_i) & = & \emptyset \\ C & = & \{(b_i, b_j) \mid i \neq j\} \end{array}$$

In this repository, every package a_i depends on the disjunction of all packages b_j with $j \neq i$. Hence, any incomplete collection of packages a is co-installable: if package a_i is a package missing from that collection then we can simply satisfy all dependencies by installing package b_i. Installing all packages a together, however, would require to install at least two different packages b. Since any two different packages b are in conflict this is not possible.

The desirable property that we want to ensure for a repository R is the following:

Definition 5 *A repository R is* trimmed *if every package $\pi \in R$ is installable with respect to R itself.*

In Debian lingo this translates to the fact that no package in the repository is "broken", i.e. that there is at least one possible installation in which any given package is installable. If this is not the case then that particular Debian distribution will be shipping packages that users will never be able to install.

2.2.2 Results, Tools, and Applications

Result: Installability is NP-complete

Based on the formalization given in Section 2.2.1 one can now quite easily show that the problem whether a given package is installable in a given repository is logarithmic-space equivalent to the famous SAT problem. This means two things:

1. One can construct for any installability problem a SAT problem such that the former has a solution if and only the latter has a solution [EDO05, MBC+06].

2. One can construct for any SAT problem an installability problem such that the former has a solution if and only the latter has a solution [EDO06].

The "logarithmic space" qualifier means that the construction can be done with auxiliary memory of size logarithmic in the size of the given problem. This is necessary to transfer complexity results from one problem to the other.

For instance, in order to translate an installability problem into a SAT problem we will interpret a package `p` as a Boolean variable with the intuitive meaning that package `p` is installed in the chosen solution. Dependencies are translated as implications: If package `p` depends on `a,b,c|d,e|f` (which would be written $D(p) = \{a, b, \{c, d\}, \{e, f\}\}$ according to Definition 2) then this translates to the Boolean implication:

$$p \rightarrow (a \wedge b \wedge (c \vee d) \wedge (e \vee f))$$

A conflict, say between packages `a` and `b`, is expressed as the formula $\neg(a \wedge b)$. The formula p expresses that the package p has to installed. This encoding opens the way to using existing SAT solving techniques to the resolution of installability problems (see Section 2.2.2). Since one has reductions in both directions one obtains an exact worst-case complexity:

Theorem 1 *The problem whether a given package is installable in a repository is NP-complete.*

On a theoretical level this means that checking installability is infeasible *in its full generality.* In practice it means as little as that it is a challenging problem since in practice one does not encounter randomly chosen repositories. The repositories we encounter in reality have a quite particular structure. For instance we will certainly have few packages with a very high number of reverse dependencies, and a large number with very few reverse dependencies. Indeed, the implementation developed in the EDOS project is surprisingly efficient (see Section 2.2.2). This apparent contradiction between theoretical very bad *worst-case* complexity on the one hand and the existence of implementations that are surprisingly fast for *selected problem instances* is quite common in computer science.

Tools: edos-debcheck, pkglab and ceve

The `edos-debcheck` utility (available in Debian in the package of the same name) takes as input a package repository and checks whether one, several or all packages in the repository are installable with respect to that repository. This utility is based on the SAT encoding mentioned in Section 2.2.2 and employs a customized Davis-Putnam SAT solver [ES04]. Since all computations are performed in-memory and some of the encoding work is shared between all packages considered this is significantly faster than constructing a separate SAT encoding for the installability of each package, and then running an off-the-shelf SAT solver on it. For instance, checking installability of all packages of main testing/amd64 takes only 5 seconds on a dual-core amd64 (emitted warnings about bad package version numbers and other irregularities are omitted):

```
edos-debcheck </var/lib/apt/lists/._main_binary-amd64_Packages \
  > out
Parsing package file...  1.2 seconds   21617 packages
```

```
Generating constraints...  2.3 seconds
Checking packages... 1.5 seconds
4.692u 0.324s 0:05.03 99.6% 0+0k 0+0io 0pf+0w
```

An explanation in case of non-installability is given, see Figure 2.5 for an example. We have also developed an RPM version of this tool called `edos-rpmcheck`.

`pkglab` is an interpreter for a query language that combines basic queries to edos-debcheck, resp. edos-rpmcheck, with a functional language which allows to use constructions like `map` to manipulate conveniently lists of packages. The interpreter allows to assign intermediate results to variables. We are planning for the future a major overhaul of the query language with the goal of making it more useful as a scripting language for applications like the one described in Section 2.2.2. The interpreter can load repositories that have been preprocessed by the `ceve` parser which can parse and analyze both Debian and RPM repositories. The Debian package for `pkglab` is pending while the `ceve` package is currently available in experimental.

Application: Finding Uninstallable Packages in Debian

`edos-debcheck` is currently used to monitor the state of Debian's distributions (*unstable*, *testing*, *stable*), as well as Skolelinux and Debian GNU/kFreeBSD. The results of the analysis are available at `http://edos.debian.net/edos-debcheck`.

There are different reasons why non-installable packages actually exist in these distributions. One important reason is that most of the binary packages are architecture dependent, that is there is one package per architecture. As a consequence, when accessing the reasons for non-installability of packages we have to take into account all possible Debian architectures.

The meta-data of a binary package are generated during the package compilation from the meta-data in the source package, and may depend on the actual compilation environment or conditional code in the source package. As a consequence, the metadata of a package with the same package name and version may vary from architecture to architecture.

- The *unstable* distribution is in fact the staging ground for building releasable distributions. Packages that depend on each other enter this distribution in an arbitrary order which depends on when a developer uploads a package, or on when a package is compiled and uploaded by an autobuilder (these are daemons that compile packages for the various architectures). For instance, package a may depend on package b, and the developer of a uploads a package for the architecture `i386` while the developer of b uploads his package for `amd64` (he should have tested package b using a locally built binary package of a on `amd64`). In this case, a is uninstallable in the repository for `i386` until the `i386` autobuilder daemon uploads the binary package for b. This is illustrated by Figure 2.3, the numbers of uninstallable packages in sid are indeed varying from day to day.

As a consequence, transient non-installability errors are normal in the *unstable* distribution. Persistent errors, however, indicate a potential problem.

- A package a may depend on package b, but b is not available on all architectures a is available on. This may be due to the fact that there is a problem with compiling b on some architectures, or that a has a too liberal architecture specification.

- A special case of the latter is that a has its architecture set to `all`. This indicates a binary package that is in fact the same on all architectures, and hence exists only once in the package pool. Package a may, however, depend on a package b which is architecture *dependant* but does not exist for every architecture. Introducing a field "Installs-to" in the syntax of control files (as proposed in Bug report #436733[2]) would allow to fix this.

  Packages which aren't installable on any of the architectures of a distribution are more likely due to an error. This may happen with packages that are installable in some architecture that has been part of a distribution in the past, but which has been removed since then. Another possible reason is dependency on a package that had to be removed from a distribution, for instance due to licensing problems or grave bugs.

Application: Debian Weather

This is more of a fun application. Based on the numbers of the tool described in Section 2.2.2 a "weather report" of Debian is generated which indicates the percentage of non-installable packages for the different distributions and architectures. The interpretation is as follows:

clear	$< 1\%$
few clouds	$1\% \ldots 2\%$
clouds	$2\% \ldots 3\%$
showers	$3\% \ldots 4\%$
storm	$> 4\%$

An example weather report is given in Figure 2.6. Applets for Gnome and KDE are available.

The daily updated Debian weather is available on the web at `http://edos.debian.net/weather`.

[2]`http://bugs.debian.org/436733`

unstable/main:

Date	alpha	amd64	arm	armel	hppa	hurd-i386	i386	ia64	m68k	...	**some**	**every**
22/06	949(325)	121(80)	604(126)	609(103)	613(132)	4445(1333)	228(131)	456(120)	8943(4583)	...	10222(5163)	41(12)
Δ	+20/−2	+7/−11	+22/−24	+28/−81	+24/−34	+10/−38	+31/−7	+26/−21	+21/−10	...	+44/−5	+0/−7
21/06	931(312)	125(78)	606(132)	662(117)	623(141)	4473(1339)	204(109)	451(121)	8932(4586)	...	10183(5141)	48(12)
Δ	+44/−0	+1/−1	+18/−7	+52/−12	+84/−0	+44/−2	+56/−0	+58/−0	+34/−5	...	+13/−22	+0/−1
20/06	887(287)	125(78)	595(121)	622(108)	539(112)	4431(1337)	148(92)	393(103)	8903(4585)	...	10192(5150)	49(13)
Δ	+90/−5	+6/−65	+17/−77	+21/−14	+14/−63	+15/−2	+19/−65	+13/−64	+26/−15	...	+28/−9	+1/−2
19/06	802(273)	184(83)	655(129)	615(109)	588(113)	4418(1338)	194(94)	444(107)	8892(4583)	...	10173(5148)	50(13)
Δ	+6/−0	+2/−7	+2/−113	+1/−8	+5/−18	+2/−221	+3/−3	+5/−7	+1/−37	...	+1/−207	+1/−0
18/06	796(270)	189(87)	766(145)	622(114)	601(120)	4637(1380)	194(96)	446(109)	8928(4588)	...	10379(5187)	49(13)
Δ	+5/−0	+4/−8	+115/−76	+5/−64	+0/−21	+6/−3	+4/−1	+1/−76	+5/−5	...	+25/−2	+0/−0
17/06	791(268)	193(92)	727(157)	681(142)	622(132)	4634(1379)	191(93)	521(132)	8928(4589)	...	10356(5167)	49(13)
Δ	+12/−12	+11/−1	+14/−57	+15/−74	+67/−105	+4/−32	+4/−42	+9/−67	+16/−1	...	+8/−19	+0/−1
16/06	791(263)	183(82)	770(175)	740(154)	660(156)	4662(1380)	229(96)	579(145)	8913(4575)	...	10367(5179)	50(13)

Figure 2.3: Summary of results of running edos-debcheck on unstable/main between June 16 and June 22, 2008. The architectures *mips*, *mipsel*, *powerpc*, *s390*, and *sparc* are omitted from this table for lack of space.
In each day's listing, the first number is the number of non-installable packages, while the number in parentheses is the number of non-installable packages that are architecture-specific. Lines marked Δ give the number of packages becoming uninstallable the following day (+), resp. that are no longer uninstallable (-). This field is colored red when the total number of uninstallable packages is increasing, green when that number is decreasing.

Results of a current run can be found at `http://edos.debian.net/edos-debcheck/unstable.php`.

testing/main:

Date	alpha	amd64	arm	armel	hppa	i386	ia64	mips	mipsel	powerpc	s390	sparc	**some**	**every**
23/06	367(7)	14(2)	217(4)	348(21)	369(9)	12(4)	48(3)	267(3)	269(3)	21(3)	56(3)	24(3)	628(32)	8(2)
Δ	+0/−0	+0/−0	+0/−1	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−3	+0/−0	+0/−0	+0/−0	+0/−0
22/06	367(7)	14(2)	218(4)	348(21)	369(9)	12(4)	48(3)	267(3)	269(3)	24(4)	56(3)	24(3)	628(32)	8(2)
Δ	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−3	+0/−3	+0/−0	+0/−3	+0/−3	+0/−0	+0/−0
21/06	367(7)	14(2)	218(4)	348(21)	369(9)	12(4)	48(3)	270(4)	272(4)	24(4)	59(4)	27(4)	628(32)	8(2)
Δ	+0/−0	+0/−3	+0/−3	+0/−9	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−7	+0/−3
20/06	367(7)	17(3)	221(5)	357(24)	369(9)	12(4)	48(3)	270(4)	272(4)	24(4)	59(4)	27(4)	635(35)	11(3)
Δ	+7/−0	+3/−0	+4/−3	+3/−27	+4/−0	+3/−0	+3/−0	+5/−11	+5/−0	+5/−0	+5/−0	+5/−0	+5/−16	+3/−0
19/06	360(5)	14(2)	220(6)	381(31)	365(8)	9(3)	45(2)	276(2)	267(2)	19(2)	54(2)	22(2)	646(42)	8(2)
Δ	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0
18/06	360(5)	14(2)	220(6)	381(31)	365(8)	9(3)	45(2)	276(2)	267(2)	19(2)	54(2)	22(2)	646(42)	8(2)
Δ	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0	+0/−0
17/06	360(5)	14(2)	220(6)	381(31)	365(8)	9(3)	45(2)	276(2)	267(2)	19(2)	54(2)	22(2)	646(42)	8(2)

stable/main:

Date	alpha	amd64	arm	hppa	i386	ia64	mips	mipsel	powerpc	s390	sparc	**some**	**every**
23/06	184(0)	13(0)	96(2)	189(0)	0(0)	67(0)	185(0)	186(0)	13(0)	183(0)	144(4)	235(6)	0(0)

Figure 2.4: The same statistics as in Figure 2.3 now for testing and stable (only one day shown since no variation).

Package	Since	Version	Explanation
...	...	...	...
calendarserver	20 Jun 08	1.2.dfsg-3	`calendarserver (= 1.2.dfsg-3) depends on python-twisted-calend... (>= 0.2.0.svn19773-3) {NOT AVAILABLE}`
camping	21 Jun 08	1.5+svn242-1	`camping (= 1.5+svn242-1) depends on rails {rails (= 2.0.2-2)} rails (= 2.0.2-2) depends on rdoc (>> 1.8.2) {rdoc (= 4.2)} rdoc (= 4.2) depends on rdoc1.8 {rdoc1.8 (= 1.8.7.22-1)}`
...	...	...	...
rdoc1.8	21 Jun 08	1.8.7.22-1	`rdoc1.8 (= 1.8.7.22-1) depends on ruby1.8 (>= 1.8.7.22-1) {NOT AVAILABLE}`
...	...	...	...
shoes	21 Jun 08	0.r396-4	`shoes (= 0.r396-4) depends on libgems-ruby1.8 {libgems-ruby1.8 (= 1.1.1-1)} libgems-ruby1.8 (= 1.1.1-1) depends on rdoc1.8 {rdoc1.8 (= 1.8.7.22-1)}`

Figure 2.5: An excerpt from the list of uninstallable packages in sid/i386 main for June 22, 2008. In the explanation field, available versions of a package are indicated between curly brackets. Lines may refer to packages shown non-installable elsewhere, like the packages `camping` and `shoes` being not-installable because it need `rdoc1.8`. Package names written in *italics* in the left column have Architecture=all.

Results of a current run can be found at `http://edos.debian.net/edos-debcheck/results/unstable/latest/i386/list.php`.

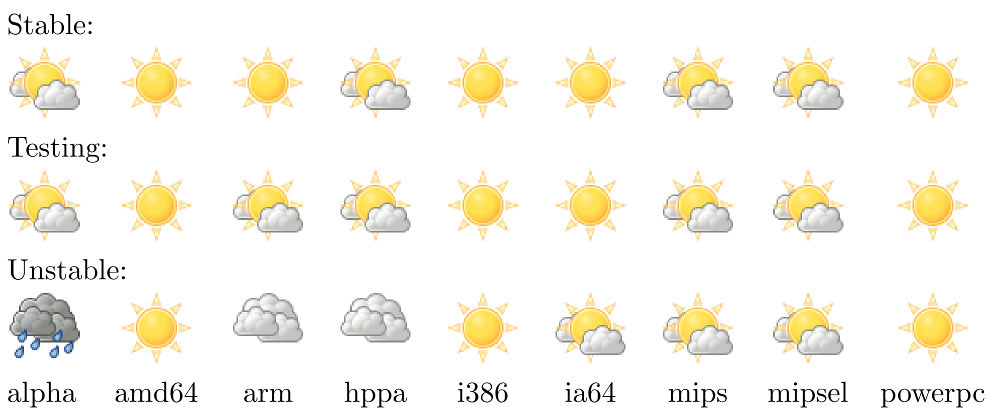

Figure 2.6: The Debian weather for June 27, 2008: Mostly sunny in stable and testing, at places overcast and rainy in unstable.

Application: Finding File Conflicts in Debian

A Debian installation has the concept of files owned by packages. If one tries to install a new package that would hijack a file owned by another package this will make (with some exceptions, see below) the installation fail, like this:

```
Unpacking gcc-avr (from .../gcc-avr_1%3a4.3.0-1_amd64.deb) ...
dpkg: error processing
 /var/cache/apt/archives/gcc-avr_1%3a4.3.0-1_amd64.deb
 (--unpack):
 trying to overwrite '/usr/lib64/libiberty.a', which is also in package
 binutils
dpkg-deb: subprocess paste killed by signal (Broken pipe)
Errors were encountered while processing:
 /var/cache/apt/archives/gcc-avr_1%3a4.3.0-1_amd64.deb
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

Our aim is to detect these errors by analyzing the Debian distribution, hopefully before they actually occur on a user machine.

An obvious naïve solution would be to try to install together all pairs of packages that occur in the distribution. Debian amd64/testing has currently about 21.000 packages, that would make about 200.000.000 pairs of packages to test, which clearly is not feasible.

A first idea towards a better solution is to only consider those pairs of packages that actually share at least one file. Luckily, the information which package contains which file is available in the file `Contents` of the distribution. This file contains stanzas like

```
...
bin/fbset                       admin/fbset
bin/fgconsole                   utils/console-tools,utils/kbd
```

```
...
etc/default/nvidia-kernel       contrib/x11/nvidia-kernel-common
...
```

In this file, information is indexed by path names of the files (omitting the initial slash). For every file a comma separated list of packages containing that file is given where packages are indicated with their section (a classification of packages by type, like `games` or `admin`), and probably the component if it is different from `main` (which can currently be `contrib` or `non-free`). For instance, the file `/bin/fgconsole` is provided by the packages `console-tools` and `kbd` which both are in section `utils`. In fact the `Contents` file that can be found on a Debian mirror may be slightly out of date as this file is generated only once per week.

The `Contents` file of amd64/testing (as of May 2008) contains about 2.300.000 entries. It is a trivial programming exercise to compute from this file a list of pairs of packages that share at least one file.

Sharing a file does not necessarily mean a bug. There a several reasons why it may be OK for two packages, say `A` and `B`, to share a file, say F:

1. The two packages are not co-installable by the package relationships declared in their distribution, in the sense of Section 2.2.1.

2. One of the packages, say `A`, declares that it has the right to replace files owned by `B`, by having in its control file a stanza `Replaces: B`.

3. One of the packages, say `B`, *diverts* the file `F` that it shares with package `A`. This means that if package `B` is being installed on a system already containing package `A` then `A`'s version of file F will be renamed; file `F` will be restored to its original name when package `B` will be removed. File diversions are declared by invoking the tool `dpkg-divert` from a maintainer script which will simply register the diversion request in a system-wide database. This database is consulted by `dpkg` when installing files. Diversions are not declared in the package control file.

We proceed in two stages in order to find the actual file overwrite problems:

1. Co-installability is checked with the `pkglab` tool (see Section 2.2.2). This is the only tool that can detect "deep" conflicts between packages. This first phase gives us a reduced list of pairs of packages.

2. Knowing which files are diverted by a package poses different problems: diversions are registered by the so-called postinst script of a package, one of the maintainer scripts that are executed during installation (or upgrade, or removal) of a package. This leads to two problems:

   a) Execution of the postinst script depends on the current state of the system, and can in general not be described by a simple list of files.

b) The postinst script is written in a Turing complete language (usually Posix shell or bash), which means that exact semantic properties are undecidable.

For this reason, we try in the second phase to install each of the pairs of packages remaining after the first phase in a chroot, using `apt-get install`. We then search the install log for file overwrite errors.

The following statistics is from the first run performed on April 16th, 2008, on amd64/sid:

Theoretical pairs of packages according to the distribution	200.000.000
Pairs of packages sharing a file according to `Contents`	867
Co-installable pairs among these according to `pkglab`	102
File overwrites detected	27

Checking co-installability with EDOS pkglab took 30 minutes and gave a 88% reduction of the search space. Testing the installation of the remaining 102 pairs of packages still took 2.5 hours. This measures where taken with a dual-core amd64 at 1.6GHz, using a local Debian mirror access over a fast LAN.

Detected bugs are tracked in the Debian bug tracking system, and marked there with user `treinen@debian.org` and usertag `edos-file-overwrite`.

2.3 Present and Future: Mancoosi

2.3.1 An Overview of the Mancoosi Project

Mancoosi picks up the baton from where EDOS left it. So, where to go from EDOS? Even though some of the theoretical achievements of EDOS still have some way to go before reaching the practice of all distributions (including Debian), adoption of EDOS results is ongoing and is actually extending past the distribution universe; a noteworthy example is the Eclipse platform, which is moving to SAT solving to solve inter-plugin dependencies.

On the contrary, one side of the complexity issues introduced by the overwhelming amount of packages in GNU/Linux distributions has been neglected by EDOS and is still in need of both research and tool development: the *user side* of a distribution. While EDOS has focused on the *distribution editor side* (i.e. on who is actually creating the distributions), Mancoosi focuses on who is actually using a distribution, in particular *system administrators*.

It is well-known that distributions raise difficult problems for administrators. Distributions evolve rapidly by integrating new versions of software packages that are independently developed. System upgrades may proceed on different paths depending on the current state of a system and the available software packages, and system administrators are faced with choices of upgrade paths, and possibly with failing upgrades. All together, these intertwined problems are referred to as the *upgrade problem*. The Mancoosi project aims at developing tools for the system administrator that address the upgrade problem.

What does constitute an upgrade problem from the point of view of a system administrator? Intuitively, any possible change to the database of locally installed packages constitutes an upgrade problem. Such changes are usually requested to a meta-installer and are well-known to any system-administrator. Some examples:

- `apt-get install wesnoth`
- `aptitude upgrade cappuccino`
- `apt-get dist-upgrade`
- `aptitude purge emacs22`
- `wajig install vim-full`

Each of the above examples poses a simple upgrade problem. Way more complex upgrade problems can be formed by combining simpler problems (e.g. posing all the above requests together to a single meta-installer). Yet more complex problem can be created by exploiting meta-installer specific features such as requiring specific package versions or origin suites (think at `apt` pinning).

A basic principle of the Mancoosi project was that the upgrade process can be decomposed into two parts: dependency resolution and upgrade deployment. While dependency resolution can be thought of as a static phase, where without altering the package database a meta-installer has to figure out if and how to implement the user request, upgrade deployment is more dynamic and consists of several sub-activities: package download, package unpacking, maintainer scripts execution ...

According to this distinction, the two main avenues pursued by Mancoosi are:

rollback support Upgrade deployment can fail for various reasons easily encountered in system administrator nightmares (disks running out of space, 404 while downloading a package, maintainer script failures, file overwrites among unrelated packages, ...). Depending on how bad the error is, a common attempted solution is that of *rolling back* the system, partially or completely, to a safe state which predates the upgrade attempt. Unfortunately, support for upgrade attempt rollback is basically inexistent in state of the art installers. Note that the need for a rollback may also occur some time after an upgrade (even days or weeks), and that in that case one only wants to undo the package upgrade but not any other system changes that have been applied in the meantime. This means that we are looking for solutions beyond mere file system snapshots.

Mancoosi aims at developing mechanisms that provide for rollback of failed upgrade attempts, allowing the system administrator to revert the system to the state before the upgrade. In particular, rollback is the topic of Mancoosi work packages 2 and 3.[3]

dependency solving The first part of the upgrade problem is implemented by state of the art meta-installers, but each of them has deficiencies (e.g. incompleteness: the inability to find an upgrade path each time one upgrade path does exists).

Mancoosi aims at developing better algorithms to plan upgrade paths based on various information sources about software packages and on optimization criteria. Dependency solving is the topic of Mancoosi work packages 4 and 5.

As the authors are only marginally involved with rollback support, that part of the project will not be discussed any further in this paper. We will for the rest of this paper concentrate on dependency solving.

2.3.2 Dependency solving

As already mentionend, the overall goal of this part of Mancoosi is improving dependency solving in state of the art meta-installers, solving some of their deficiencies. More precisely, Mancoosi plans to address three requirements which are believed to define the ideal to which any given meta-installer should tend to: completeness, optimality, efficiency.

Completeness

The first of these requirements can be defined as follows:

Definition 6 *A meta-installer is* complete *wrt. dependency solving iff for each possible upgrade problem which has a solution, the meta-installer is able to find such a solution.*

Even though not enough details have been given to fully formalize completeness in this paper, the intuition should be clear: once the system administrator poses an upgrade problem to its meta-installer of choice, the meta-installer tries to solve dependencies to fulfill the user request to determine which changes should be made to the set of installed packages. *If* a healthy installation satisfying the user request does exist, then the meta-installer should be able to propose it as *a* possible way of fulfilling the user request.

[3]`http://www.mancoosi.org/work.html`

Surprising as it might sound, most state of the art meta-installers are not complete. For instance, upon receiving a request like `install p`, `apt-get` always tries to install the latest version of `p` among those available in the package universe formed by APT repositories. In case the version requirements of (latest) `p` are not satisfiable it might well be that requirements of (previous) `p` are indeed satisfiable. In such and similar cases the user is left with the feeling that there is no way to satisfy her request, while this is actually not the case: this is a lack of completeness that should be addressed to improve user experience with meta-installers.

Note that the given example is just a paradigmatic one, more complex examples built on top of the limited back-tracking capabilities of other meta-installers can also be provided [EDO06] (see also `http://www.mancoosi.org/edos/manager.html` for an analysis of the situation in the year 2006). The general point stressed here is that legacy meta-installers which are advertised as *the* tools for system-administrators to interact with the package database of their machines should be able to solve dependency problems each time it is possible to do so.

Optimality

Once it can be taken for granted that any possible solution to a dependency problem can be found, it is natural to ask *which* among all the possible solutions has to be preferred over the others.

Note that for any given upgrade problem there are in general several possible solutions. If you consider again the `install p` request posed to apt-get above, a possible solution for it is to install the version of `p` whose dependencies are satisfiable together with all its (transitive) dependencies and be done with that. Another valid solution is to install the same set of packages together with a package `z` which is completely unrelated to `p` and that does not inhibit a healthy installation. Whereas in these two cases it seems obvious that the former has to be preferred, in the general case there are non obvious choices to be made. Anyone who has already been faced with `aptitude` interactive solution discrimination knows that: in satisfying dependency problems coming from user requests, trade-offs have to be made.

In fact, even before discussing *how* the optimal solution has to be found among all alternative solutions of a given upgrade problem, there is a need to understand which criteria should be used to define the optimality of a given solution. At the moment some fixed criteria which are likely to address most user needs are being considered; here is a handful of examples:

- minimize the amount of extra-packages installed with respect to those explicitly mentioned in the user request,
- minimize the download size of packages required to deploy the upgrade solution,

- minimize disk usage after the upgrade (a frequent need for Debian-based embedded distributions),
- upgrade as many packages as possible to the latest available version.
- . . .

Of course different optimization criteria can be in conflict one with another. If on one side this brings the upgrade problem in the vibrating research field of multi-criteria optimization, it also raises the issue of which interface should be given to users to specify their optimization preferences. Moreover, the set of possible optimization criteria should be open-ended as specific user needs arise every day: APT pinning is a practical example of user requests that should be taken into account while choosing an optimal solution, countless other user-specific requirements can be imagined (e.g.: when you have a choice among two packages choose the one with less RC bugs, or even blacklist packages maintained by Random J. Developer as you don't trust him . . .). For this reason Mancoosi will also be developing a cross meta-installer language to specify optimization criteria with a well-defined semantics, to be used by system-administrators to specify their preferences.

Efficiency

Once it is settled what properties we want from the ability of a meta-installer to solve dependencies (completeness and optimality), the attention can be turned to how we would like the given tool to reach a solution . . . and of course we want it to be efficient in finding it. Even letting aside the optimization part, dependency solving is per se a NP-complete problem (see Section 2.2.2) hence we cannot hope for a definitive algorithm or implementation delivering upgrade problem solution instantaneously in any given case.

Nevertheless we should strive for the most possible efficiency and in this respect the EDOS results have been encouraging. Mancoosi will focus on finding efficient algorithms which not only take into account package installability "in the void" (i.e. in some, not specified a priory, installation), but rather which address upgrades starting from an existing user installation.

2.3.3 A solver competition

Promising to find *the* most efficient algorithmic solution to the upgrade problem, implementing both completeness and optimality in the setting of the Mancoosi project would have been inconsiderate. This is why Mancoosi chooses a different path: try increasing the sensibility of the relevant research communities on the upgrade problem. Historically, the organization of periodic competitions has been a training factor in pushing further the state of the art in algorithms and tools for complex problems such as SAT. Examples like the SAT competition[4] and SAT race[5] attract yearly research and practitioners willing to challenge their tools with competitors to determine which is the “best” both in terms of solver capabilities and in terms of execution speed.

Mancoosi will follow a similar path for the upgrade problem faced routinely by meta-installers. A competition of dependency solvers will be organized and is planned to be held in parallel with a research conference on related fields (SAT-solving, linear optimization, . . .). While it is too early to have detailed information on how the competition will be run and organized, some aspects are already clear.

Upgrade problem database To run a solver competition you need a corpus of problems that will be used to challenge the various competitors. In the Mancoosi case the corpus will be called UPDB for Upgrade Problem DataBase. The way in which it will be assembled is different from other competitions. Instead of creating artificial problems by hand (that would be not only challenging given the typical size of a distribution repository, but also bear the risk of creating irrelevant problems) the corpus will be composed of problems submitted by users who encountered these.

All in all, the architecture is similar to that of the Debian Popularity Contest:[6] users interested in participating will be asked to install some special-purpose packages which provide the software to gather data and submit it to a central repository. In some cases it will probably be necessary to install modified versions of meta-installers which have been changed to log enough information to fully describe an upgrade problem. The architecture of problem submission to UPDB is depicted in Figure 2.7.

As various distributions are taking part in the Mancoosi competition, each of them will be providing a staging repository to which problem submissions will be addressed. One such repository will be set-up for Debian users as well. As the format of the initial submission is distribution-specific, a further conversion step into a common format used to encode problems is needed. Once the conversion has been done, the upgrade problem is fully abstracted over the origin distribution and can be fed as input to the various solvers which will be taking part in the competition.

The Mancoosi project will be both organizing the competition (and this is the topic of work package 5) and participating in it (work package 4) with a research team which is expert in SAT solving and optimization techniques and which will be developing ad-hoc algorithms for the upgrade problem as faced in distributions.

[4]`http://www.satcompetition.org/`
[5]`http://www-sr.informatik.uni-tuebingen.de/sat-race-2008/`
[6]`http://popcon.debian.org/`

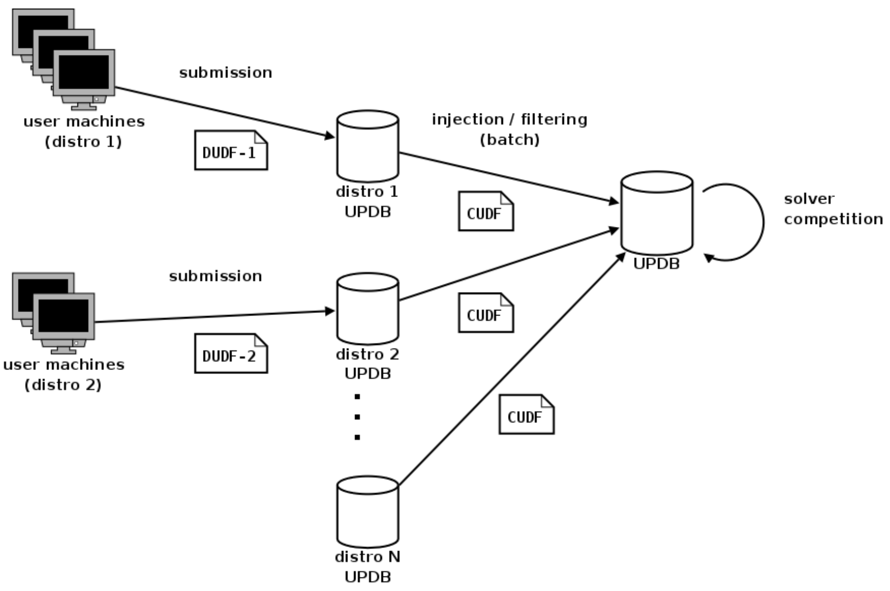

Figure 2.7: Data flow of UPDB submissions, from users to the corpus of problems for the competition

Types of competitions Different kinds of competitions will be held. In the beginning it is planned that the optimization criteria will be fixed and each competitor will specifically be participating in a selection of them. For example it is likely that we will be having categories like: no optimization (just solve the upgrade problem no matter what), minimize the download size of required packages, minimize disk usage, and so on.

Upgrade Description Formats As it can be observed in Figure 2.7, different format specifications are required before being able to start collecting upgrade problems from users (that notwithstanding specification implementations, which will be required as well). Such specifications are work in progress and are available in the Mancoosi public repository available at `http://gforge.info.ucl.ac.be/plugins/scmsvn/viewcvs.php/trunk/updb/doc/cudf/?root=mancoosi.`

The first specification *DUDF (Distribution Upgrade Description Format)* is meant to describe the format used for the actual submission of upgrade problems from user machines to the repositories set up by each distribution interested in collecting upgrade problems. As the format is in the end distribution-specific, the specifications describe the overall structure and basic principles of a submission document, the actual details will be filled in by each distribution according to the user installers and meta-installers. Interested distributions are encouraged, once the final version of DUDF will be ready, to publish notes describing exactly how they are implementing the distribution-specific part of

DUDF.

Roughly, a DUDF document has the following parts:

1. local package status on the user machine
2. current package universe as known to the meta-installer
3. requested action
4. user desiderata (i.e. optimization criteria)
5. various identifiers (e.g.: distribution identifier, installer name and version, meta-installer name and version, . . .)
6. outcome of the meta-installer (a new local package status in case of success, a failure message otherwise)

A hypothetical (and incomplete) mapping to Debian for the `apt-get`, just to give a practical intuition of what can constitute a DUDF submission, is as follows:

1. `/var/lib/dpkg/status`
2. the set of APT binary package lists as stored under `/var/lib/apt/lists/`
3. the given APT command
4. current APT pinning settings
5. “debian”, “apt-get”, v$x.y.z$, “dpkg”, . . .
6. “broken packages, the following packages can not be installed, ...”

As sending all the above information can be costly in terms of submission size, DUDF implements some space-optimizations. The most important optimization is based on the assumption that most package lists composing a given package universe are usually only mirrored on a local machine and are available elsewhere. Hence, by keeping distribution-specific historical mirrors of a given distribution, instead of sending whole package lists, a DUDF submission may just contain package list checksums that can later be looked up in historical mirrors to recreate the package lists as available on user machines. In the specific case of Debian, Mancoosi will be keeping historical mirrors of APT lists for the most widespread `apt-get` repositories: not only the official stable/testing/unstable Debian suites, but also volatile, backports, debian-multimedia, . . .

The second, and last, document format involved with the solver competition is *CUDF (Common Upgrade Description Format)*. That is the format in which the actual inputs from competition participants will be encoded in. Contrary to DUDF, CUDF is distribution agnostic as well as agnostic to any specific installer or meta-installer. A requirement for any given DUDF document is that it can be converted to CUDF, during that conversion step all performed space-optimization will be expanded to obtain a self-contained description of an upgrade problem.

2.3.4 Debian and Mancoosi

As already mentionend there is no "official" relation between the Mancoosi and Debian projects; however, there are Debian developers in the ranks of Mancoosi which are interested in giving back to Debian as much as possible of Mancoosi achievements. This section lists the foreseeable points of contact between Mancoosi and Debian, it also points to the available resources for interacting with Mancoosi from the Debian side.

Probably the main point of interest for Debian in Mancoosi is the possibility to improve the available algorithms and tools for dependency solving, both from the point of view of performance and the point of view of capabilities. To be delivered in Debian, the possible forthcoming achievements will need cooperation among the algorithm developers and the developers of meta-installers used in Debian (apt-get, aptitude, . . .). The Debian developers involved in Mancoosi have already taken contact with members of the respective development teams. Collaborations are needed mainly in two areas:

common solver API It is unlikely that Mancoosi will have the energy to port novel dependency resolution algorithms to multiple meta-installers, it is more likely that only a proof of concept implementation for a single tool will be developed. As Debian is also about diversity, it would be preferable to have implementations for all the mainstream meta-installers. To this end a side-result that will be pursued is the development of a common API to let whatever meta-installer interact with an *external dependency solver*. This way it would be possible to develop separately meta-installers and plug them into different tools. Such an achievement, if reached, would also mean that it will be possible to exchange solvers which already exist among different tools, gaining flexibility in the overall package manager implementation.

dependency solving logging Once the specification of DUDF will be finalized, its implementations will basically consist of patches (or plugins, where feasible) for meta-installers enabling them to save in DUDF format solving attempts originated from upgrade problems. As it will be beneficial to have a common format for logging such attempts (e.g. for bug reports against apt-get, aptitude, . . .) we hope to spread DUDF implementations in whatever meta-installer is currently used in Debian.

On a less implementative side, Mancoosi is welcoming comments from the Debian community on all aspect of the project. In particular, at the time of this writing we are interested in comments on what will constitute *interesting optimization criteria* as those anticipated in Section 2.3.2. The corpus of collected optimization criteria is likely to be used as the set of categories to run the first solver competition. Do not hesitate to get in touch with the

Mancoosi project if you have suggestions on this topic or on anything else related to the project!

To **get in touch with Mancoosi** there are various ways.

- The official *website* gives general information on the Mancoosi project, it is available at `http://www.mancoosi.org`
- The mailing list to archive public discussions about Mancoosi is mancoosi-discuss: `http://sympa.pps.jussieu.fr/wws/info/mancoosi-discuss`
- Then there are also *Debian-specific contacts*
  - `http://mancoosi.debian.net` has been set-up as a web archive of resources for the Debian project offered by Mancoosi. At the moment it just contains the historical mirror of APT's binary package lists which will be used to implement the space-optimization of DUDF.

    It also contains an apt-get repository of unofficial Debian packages meant as a staging area for packages not (yet) accepted in the Debian archive, or simply not suitable/interesting enough for it.
  - the email contact `debian@mancoosi.org` is the main contact to get in touch with Mancoosi for Debian-related issues, questions, comments ... Drop a mail to it for more information!

Bibliography

[EDO05] EDOS Project Workpackage 2 Team. Report on formal management of software dependencies. EDOS Project Deliverable Work Package 2, Deliverable 1, September 2005. `http://www.edos-project.org/xwiki/bin/Main/Deliverables`.

[EDO06] EDOS Project Workpackage 2 Team. Report on formal management of software dependencies. EDOS Project Deliverable Work Package 2, Deliverable 2, March 2006. `http://www.edos-project.org/xwiki/bin/Main/Deliverables`.

[ES04] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[MBC$^+$06] Fabio Mancinelli, Jaap Boender, Roberto Di Cosmo, Jérôme Vouillon, Berke Durak, Xavier Leroy, and Ralf Treinen. Managing the complexity of large free and open source package-based software distributions. In *ASE 2006*, pages 199–208, Tokyo, Japan, September 2006. IEEE CS Press.

Chapter 3

Best practises in team package maintenance

Gregor Herrmann

Abstract

Team maintenance for (groups of) packages is en vogue; dozens or perhaps hundreds of packaging teams care for a subset of the Debian archive in a collaborative style. This BOF offers the opportunity for members of different packaging teams to exchange their experiences, share their success and problem stories, and in general learn from each other.

3.1 Introduction

If you are an active member of a team that collaboratively maintains packages in Debian and you plan to attend the BOF "Best practises in team package maintenance" during DebConf8, please take a few minutes to think about the following questions.

The BOF will start with short presentations (just a couple of minutes) along these questions, ideally one contribution by each team with members present; these presentations are not intended to be formal in any way, but instead to inform the other participants about the various teams and to serve as a basis for discussing points of common interest.

If you have a chance to discuss your team's mini-presentation with other team members (before DebConf or in Mar del Plata over a nice cup/glass of `$preferred_drink`)—even better!

3.2 Questions

3.2.1 Basics

- Name of the team
- Area of work
- Approximate number of members
- Approximate number of maintained packages

3.2.2 Work flow

Communication

- main/preferred ways of communication within the group (mailing list, IRC, commit messages, web platform, . . .)

Members

- How do you recruit and integrate new members?
- How does mutual support work?
- Is there some formal "team leadership"?
- Do you use the "Debian Maintainer" concept?

Review/uploads/sponsoring

- How do you handle reviewing/uploading packages which are prepared by non-DDs?

Infrastructure

- Where/how do you keep your packages (version control, . . .)?

Policy/rules

- Do you have written/unofficial guidelines for your packages?
- How do they work out in practise?

Tools

- Do you use any "unusual" tools in your work (scripts, web apps, . . .) that help you with regard to upstream releases, bugs, packaging itself, . . . ?

3.2.3 Experiences

Best practises/successes

- Is there something you can share with other teams that works just great for your team and might help others?

Challenges/failures

- Is there something you're not happy about in your team and where you would appreciate input from other teams?

3.2.4 Other teams/Debian

- Is there a need for improving communication/cooperation with other teams? How?
- Do you have any common projects in mind that might help all packaging teams?
- Is there something with regard to the relationship between Debian Policy and teams you would like to see changed?

Chapter 4

Custom Debian Distributions

Andreas Tille

Abstract

The idea of Custom Debian Distributions was born at DebConf 3 in Oslo and has turned now into a solid tool set that can be used to organise packages targeting at a specific work field inside Debian in a quite efficient way. After five years it is time for a report about status and success as well as continuing to spread the idea amongst people to enable them to spend a minimum effort for the adoption of the tools to get a maximum effect in maintaining a CDD.

One goal of Custom Debian Distributions is to form a group of Debian developers who care for a specific set of packages that are used in the day to day work of a certain user group. The fact that Debian has grown to the largest pool of ready to install packages on the net has led to the side effect that it is hard to maintain for beginners. A Custom Debian Distribution adds some substructure to the currently flat pool of 15000+X packages without a user oriented structure. These substructures are intended to put a focus on special user interest. These substructures are not based on technical matters like Debian installer team, porting teams or teams that are focussing to implement programming language policies.

There are some similarities to Debian-i18n which also has the pure goal to serve the needs of certain end user groups with the difference that the users are grouped not according to their field of work but according to their language. In fact it makes even sense to create CDDs for languages that require certain technical means to optimally support the language regarding direction of writing, special fonts etc. It is known that some countries in Asia builded Debian derivatives for this purpose but in principle it is not necessary to derive - the better solution is to make Debian more flexible by starting a CDD effort inside Debian.

The talks will give some examples from the success of CDDs like Debian Edu and Debian Med. One very important outcome of the CDD effort is the ongoing reunification of Linex - the Debian derived distribution that is used in all schools in Extremadura - with Debian Edu. This

step means that Debian gets a very large implementation in all schools of Extremadura while on the other hand the effort of development for the people who invented Linex will be reduced. Debian featuring Debian Edu now has a very good chance to become a really good international school distribution because it has roots in five countries (Norway, Spain, France, Germany and Japan) and might become attractive for many more.

The success stories of CDDs would not have been possible outside Debian and thus leaving the path to build Zillions of Debian derivatives that reach a very small user base and working together inside Debian is the main idea of the talk. To make this idea more attractive in the second part of the talk a description of tools that were developed in the CDD effort will be presented. Especially the newly developed web tools that give a good overview about the packages that are useful for a certain field of work and the QA tools that enable the CDD team members to easily get an overview about packages that need some action. So if people are not yet convinced that a CDD for their purpose makes sense we will catch them by the tools they might get for free if they follow the proposed strategy.

4.1 Symbiosis between experts and developers

As it was explained in last years CDD talk the basic goal of Custom Debian Distributions is to enable the user to focus on the packages that are really needed for his day to day work leading him friendly through the jungle of `Debian`'s $>$ 15000 packages. A user that is working in a certain field is only interested in a defined *subset* of packages and the CDD that is concerned about this field tries to prepare the computer optimally to install this subject with adapted configuration and easily accessible applications. So CDDs are taking care of groups of specialised users turning `Debian` into a useful tool adapted to their requirements for day to day work and making it to the distribution of choice for their use cases. It should enable and easy installation and automatically configuration whenever possible to make the needed work to fit the intended purpose as small as possible.

The tricky part in developing a CDD is now to tie a solid network of Debian developers, upstream developers ("developing experts") and users (experts in a defined field). It has turned out that gathering upstream developers into a CDD team is quite often not very hard. There are several upstream developers who try to become Debian Maintainer status - a concept which turned out to be quite successfully. The rationale behind this is if it comes to field specific software it is often written by experts in this field to solve the tasks of their daily work. Observations have shown that this software while showing great features regarding the task which should be solved there are often weak parts in the build system or in the general handling of using libraries or wide spread tools. This is exactly the point where Debian developers have good experiences and are able to provide technical help.

So it happens quite often that upstream developers of field specific applications are quite happy if Debian developers want to build Debian packages of the software because they anticipate enhancements of their build system and a security audit. Last but not least they expect a wide distribution of their work to reach a large user base easily.

4.2 Attracting people by providing interesting technical base

The acceptance of new methods is drastically higher if the techniques provided are convincing enough and provide interesting features for the target audience. Considering this we tried to develop simple ways to categorise packages that are useful for certain tasks. This is done in so called tasks files which are processed using the `cdd-dev` package to build metapackages. The other application of these tasks files is building internationalised web pages which display the packages that are relevant for a certain CDD task with the descriptions of the packages. The translation for the descriptions are drawn from the Debian Description Translation Project and the more complete the DDTP translation of packages that are relevant for a CDD are the better is the translation of the web pages featuring the CDD tasks. Thus by adding another use case of DDTP translations the effort might become additional participants and the quality of translations - especially those of specific packages which need some expert knowledge for proper translation - might increase.

The internationalised web pages which are generated automatically out of the information inside the tasks files is a key documentation feature which is a really helpful tool for developers of the CDD as well as very informative for users because they immediately get an overview about all ready to install software that might be helpful for their day to day work. Thus we try to promote these pages as the main entry point for information for our users what we have done, which work is in progress and what's on our TODO list. This information might give them a good motivation to join the project. The first step might be to provide better translations for the package descriptions which is certainly a task which is better done by experts using the package themselves instead by Joey Randomtranslator who tries his best in a word by word translation but if he does not know the real usage of the package it is hard to provide a really useful translation.

Once we were able to rise users interest they might be interested to do the next step to install and try the packages in question. This is the point where the upstream developer of the software becomes a new user which might report bugs or give hints for enhancements and might become a coworker finally. This way Debian, or more specifically the CDD that supports the specific field, has helped to increase the user base of a software and thus the potential developer base.

The process to establish a certain piece of Free Software described above seems to be quite straightforward but without a linking instance like a CDD in between upstream developer and users the propagation of specialised Free Software is often everything else than straightforward. If the idea of Free Software reaches a specialist who is working on a specific solution he is happy to release the code on his private web page – and that it is just there. It is not very common to use well known source code repositories like Savannah or SourceForge or even implementing a version control system to promote group development. In contrary if an other specialist is seeking for a solution for the same problem he has to invent extended Google queries to find the project in question – if he has the idea to seeking and before he is starting simply from scratch.

Some volunteers have realised this situation and provide extensive link lists either on static HTML pages or Wikis to enable others to join the catalogue effort. The problem here is that these link lists are often incomplete and what matters even more are not directly connected to immediately installable and executable binaries.

There are some similar efforts like CDD in other distributions for instance there is a comparable effort to package biological Free Software done by Gentoo or FreeBSD because also other distributors realised the problem described above. The difference between such kind of installable software collections and a CDD is that a Custom Debian Distribution tries to do more than just packaging specific software. It is rather about forming a team of maintainers who try to build a consistent system around several tasks in a specific field, care for easily installation using metapackages, making sure that everything works together smoothly and working actively as missing link between upstream developers and users.

4.3 CDD is more than packaging specific software

The main work of a distributor is providing precompiled binaries of Free Software, caring for smooth installation and upgrades as well as security fixes for the distributed packages. In the case of Debian which maintains the largest pool of ready to install binary software packages this is quite a large amount of work. The huge amount of software includes larger subset of software for very specific use and this is the playground of CDD packaging teams who closely work together to bundle their competence on packages with a specific user base.

It turns out that there is a good chance of cooperation between CDDs on a technical level because several jobs to do are at least similar. This idea is the basis of the whole CDD effort: Making sure that the wheel that drives a certain CDD is only invented once and adopted for all others. There is a similar situation in the internationalisation teams: There is a well defined group of users (speakers of a certain language) who need special support (translations at various places) and it make sense if language teams just work together and use common tools like DDTP server and others.

While this translation work is one part of the internationalisation team it can not stop here. It is also about proving that Debian is flexible enough to incorporate this kind of changes instead of forcing users to make language based forks of Debian. Unfortunately there are many people out there who feature a wrong concept of using Free Software. They read the license as they are allowed to modify the software as modifying their local copy and tweak it until it fits their needs. They treat this constant forking as the normal way to customise their distribution. The internationalisation team has done a good job in propagating the idea that it is better to include translations into Debian than adding translations over and over for every new release of Debian.

In principle the maintainers of a CDD have the same job: Attracting derivers who have not yet understood the power of internal customisation inside a CDD to reach their goal – optimal support of their users – more efficiently. The most convincing example how a CDD managed to merge a derivative back to Debian is that the educational branch of LinEx is working on unification with Debian Edu since end of year 2007.

4.4 Techniques

The techniques used are based on sorting certain packages of the Debian pool into certain remits or in the terminology we have chosen in CDDs "tasks". So the tasks files are listing dependencies from Debian packages and different tools are using these as common source of information.

4.4.1 Building metapackages and tasksel descriptions

The `cdd-dev` can be used to build a set of metapackages and tasksel descriptions. Because the technique is described in very detail in the article about CDD there is no need to repeat the content here. I would rather ask you to follow the link and especially read section *6.1 Metapackages*.

The build process using `cdd-dev` also included the creation of a `CDD-tasks` package which contains information for tasksel to enable the tasks of a CDD via tasksel.

4.4.2 Web pages based on common scripts

A quite new feature are the web pages that are builded based on the very same information that is used to build the metapackages. This enables users to get a very quick overview because they see all packages included in the task together with the description. Because the target audience does not necessarily is comfortable with English language the descriptions are even translated in case there is such a translation provided by the Debian Description Translation Project. Such pages are available for the following projects:

- Debian Edu
- Debian GIS
- Debian Junior
- Debian Med
- Debian Science

In addition to the packages existing inside Debian there is an easy way to specify prospective packages that should be included into Debian in the future. These packages are listed on the web pages as well. To read more about this feature just have a look into the article about CDD especially section *8.1 Existing and prospective packages*.

There is no need to copy the information of the just existing article about CDD which is continuously updated and thus this document ends here with the strong recommendation to read the technical details there.

Chapter 5

The Debian Videoteam — Behind the Scenes

Holger Levsen, Herman Robak, and Eric Dantan Rzewnicki

Abstract

The talk will cover the hardware, software and manpower required for a typical dvswitch usecase. Some caveats and challenges regarding sound, lighting and rigging will be discussed, and live demonstration of dvswitch in use will be given.

5.1 Introduction

People We would like to bind up as few people as possible in the video team. Our activity is a service, not a playful hacking thing.

Post production takes for ever, if it ever happens. We strive to get as much as possible done instantly.

Hardware The hardware should be cheap and light. Commodity hardware is more convenient than specialty hardware. Two non-commodity pieces that we now consider "must have": Scan converter (VGA to DV) and wireless mic. The laptops and the DV cameras can be borrowed by/from participants or the local team.

Software The software should be a minimal set of packages running on Debian Stable. Dvswitch fits that bill.

TODO Talkback: The camera operators should have headsets, and receive oral instructions from the mixer operator/producer.

Tally lights: The cameras should have a red lamp telling everyone (especially the camera operator) when they are "on".

Caveats Lighting problems. Good room lighting should provide a comfortable light level and adequate light for cheap cameras, and still not wash out the projector screen. More often than not, this is not the case.

Lighting workarounds. Spot light on the speaker, turn off the rest during the talk. Turn the lights on again for the Q&A session, assuming the screen won't be much needed then.

Recording the sound from the audience takes some care. We want reactions from the audience to be audible, but general murmur and air condition noise should be cut out.

The crowd cam should see *everyone* in the audience. This may require a tall platform, or (fancy!) a jib arm.

5.2 What the software does

`dvsource` Grabs video from a DV camera over firewire, and sends it over TCP to the `dvswitch`. Dvswitch may run on a different computer, usually nearby.

`dvswitch` The heart of the system is dvswitch.

`dvsink` Receives the output from dvswitch

5.3 Live demo

- Show the hardware
  - Laptops
  - Cameras, tripods, firewire
  - Scan converter
  - Switch, ethernet cable
  - Sound mixer
- Show the setup
  - Speaker cam, head mic
  - Crowd cam, crowd mic
  - Cabling (gaffa, gaffa, gaffa)
- Demonstrate editing
  - Recursive video (gag)
  - Cutaways (let the cam-op frame the shot)
  - Eye direction (when two shots won't intercut well...)

- Sound level (open and fade out crowd mic)
- PiP? Text overlays?
- Disorienting each other and the stream team (gag)
- ”Do you have goatse there?”

Chapter 6

DebConf9 Caceres

César Gómez Martín, José Antonio Recio Cuesta, Carmen Cordero Mata - Centro de Nuevas Iniciativas

Junta de Extremadura - FUNDECYT

{*cesar.gomez,anto.recio,carmen.cordero*}
@{*juntaextremadura.net*}

Abstract

In 1997 Extremadura started working on the Regional Strategy of Information Society of Extremadura (Infodex European Project) and in 1998 the President of Extremadura proposed a strategy for the regional development model taking advantage of the possibilities offered by the New Information and Communication Technologies. The strategic actions were: 34 Technological Literacy Centers, 1 PC per two students in 2005, 60 new enterprises created in a business incubator and 1 business fair per year to promote e-commerce. To be able to achieve those purposes Extremadura needed adaptability, economy, feasibility, security and universal access of citizens to the tools. That is the reason why a free software based operating system called gnuLinEx and based in Debian was chosen. Extremadura is obtaining benefits from Debian and wants to contribute back organizing worksessions like the ones held in Extremadura during 2006, 2007 and 2008. These sessions are made possible by the gnuLinEx team who arrange government sponsorship to cover attendees' costs and taking care of all the logistical, accomodation, food and hacking needs. We think that DebConf9 in Cáceres (Extremadura) will be very helpful for Debian and for gnuLinEx as well.

6.1 Introduction

Extremadura is a $41,634km^2$ region located in the south-west of Spain, near the Portuguese border, with over a million inhabitants. Extremadura is in the middle of a triangle formed by Madrid, Seville and Lisbon, that is the reason why its location is very important for all the communications between the three biggest cities in the south of the Iberian Peninsula. Its population density is 26 inhabitans/km^2, it is not a high density when compared to the population density of Spain (88.59 inhabitants/km^2) or European Union (112 inhabitants/km^2). Apart from services (60%), farming is the second sector employing workers, with 16.6% of all employed persons in the region. To understand the situation of Extremadura, it is necessary to remark that Extremadura is also the poorest region of Spain with an unemployment rate of 16%.

The regional economy has improved in the last years, being the Spanish Region which has been converging at a faster pace with other economies within the European Union, in the 1985-1999 period. Extremadura has benefited from the European Union Cohesion Funds and has used them to implement several projects regarding education, social issues and businesses. These projects, in order to make the Region catch up with the Information and Communication Technology Revolution, are favouring its development on the basis of equality and freedom, and are preparing it to firmly face all the changes the Knowledge Revolution will spark.

6.2 Information Society Project in Extremadura

In 1997 the regional government of Extremadura started working in the articulation of a Regional Strategy of Information Society for Extremadura with the creation of the Infodex European Project, an organization financed in equal parts by the regional government of Extremadura and the Structural Funds under the Regional Information Society Initiative (RISI). Infodex was aimed to study the situation of Extremadura and to identify all the requirements to develop an IT strategy. Juan Carlos Rodríguez Ibarra, the President of Extremadura's Regional Government proposed in 1998 a strategy for the Regional Development Model taking advantage of the possibilities offered by the "New Information and Communication Technologies" (NICT's). In 1999, Infodex, designed the "Director Strategic Plan for the Development of the Information Society in Extremadura". Design, within the Regional Strategy, of a Technological Framework, a Strategic Framework. The consolidation of this global project, both in the education context, supporting the creation of technologically-based businesses and promoting an ambitious plan for Technological Literacy, reached to the point in which, keeping a successful framework required to depend on an external factor, such as used software. This situation led to the creation of gnuLinEx: the need to have a software that allowed completing a project we could fully control; and this could only be made using free software. Therefore, gnuLinEx is not a product born by chance or spontaneously, but rather by the need to fulfill a double goal: on one hand, an educational goal to contribute to the development of the Red Tecnológica Educativa (Educational Technological Network), with a ratio of one computer for each two students in all the schools of the region; on the other hand, to an economic and social goal that consists on spreading free software in Extremadura, through the Plan de Alfabetización Tecnológica (Technological Literacy Plan). The availability of a fully functional software that can be copied and distributed legally, helps to overcome economic barriers, such as the high costs of software licences.

6.3 gnuLinEx

6.3.1 Educational System

In order to improve the quality of the Educational System, the Regional Ministry for Education, Science and Technology incorporated the Information Society into the Regional Educational System while the local Government implemented the Regional Intranet. The key lies in training, the elaboration of contents by teachers, the upgrading of new schools built by the Regional Government and the creation of our own free operating system. The training courses on ICT's have been carried out all through the region, and have been mostly a responsibility of the Teachers and Resources Centres. These courses have been running since 1999, and have reached 80% of teachers in the region, both physically present or online. Obviously, since gnuLinEx was introduced, training users became one of the main factors, so that they could get the basic knowledge to work with it and to look for its pedagogical possibilites: image processing, multimedia, etc. The usage of a completely free software, developed through the Internet by persons who share an enormous spirit of collaboration, has a remarkably high value for students. In other words: the idea of using a computer that works thanks to the existence of people who share their knowledge has a great educational value. Regarding the infrastructure, ever since the Regional Ministry for Education, Science and Tecnology took over the local education jurisdiction, the number of schools using the ITC's has been on the rise. Modern classrooms are bigger than the traditional ones, so that special desks can be used in order to achieve the ratio of one computer for every two students. The need to achieve a complete control of the whole array of computers the Educational Network involves, to look for a stable and powerful system and to have a kind of software that can be updated without having to rely on third parties, while lowering the costs to a minimun, made the Regional Ministry for Education, Science and Tecnology decide to use free software and to develop gnuLinEx, the only software installed in all the schools.

6.3.2 Public Administration

The Governmental Board of the Junta of Extremadura has approved an agreement that states that the electronic information generated and aimed at the interchange in all the departments and bodies that make up the Junta de Extremadura must use standard formats in a compulsory way.
- Open Document Format for office suite applications (OASIS Open Document Format, norm ISO/IEC DIS 26300), for information being created and undergoing administrative processes.
- Portable Document Format, PDF/A (ISO 19005-1:2005), for information where guaranteed unalterable visualization is required.

With this important agreement, the Junta de Extremadura is the first administration to adopt Open standards that all international organizations related to ITCs agree to refer to as the most important step to favour technological innovation, reduce the dependency of users, companies and Public Administrations on non-compatible, proprietary applications, and increment interoperability amongst systems and applications on a global level. The board has also approved that, as from now, all computer tools for personal productivity used by the staff of the Junta de Extremadura will be free office suite implementations that must support natively the established standards. Also, the designated operating system of obligatory use on all workstations of the staff of the Junta de Extremadura will be a derivative of gnuLinEx called LinEx SP. The board agreed on a gradual migration on said workstations of all administrative departments that make up the Junta de Extremadura. The deadline for the completion of the migration has been set one year from the ratification of the agreement. After one year, all the workstations of the staff of the Junta de Extremadura must be working fully and exclusively under gnuLinEx and any additional software in use must be open source software and be distributed under a free license.

6.3.3 Health and Care System

The Jara Project was aimed to link all the central services of the Health and Care System of Extremadura (SES), 10 hospitals, 104 health care centres and more than 300 centres in rural areas. This project had to build a system which will have to deal with more than 13,000 persons that work for the Health and Care System of Extremadura so it had to be adaptable, economical, secure and feasible. That is the reason why Linux Servers and gnuLinEx in the clients were chosen. This system is already working in a couple of Hospitals and will be soon spread to the rest of the Health and Care System of Extremadura.

6.4 Network in Extremadura

6.4.1 Regional Intranet

The Infodex project was also very valuable to identify the lack of good communications in Extremadura. Since Extremadura is a very wide and rural area, telecommunication companies did not want to deploy a good infraestructure to communicate the different villages of Extremadura because it was not profitable for them, like it was in Madrid, Barcelona or regions with a bigger density of population. To solve this issue, in December 2000, the regional goverment decided to build a corporative telecommunication network in order to connect more than 1,500 points of the regional administration, including hospitals, schools (more than 700 centres), libraries and other public buildings. The minimum speed of the Regional Intranet is 2Mbits/sec but now most of the links have a minimum bandwidth of 32Mbits/sec. By making a high bandwidth network available to all the schools of the region, the Regional Goverment has guaranteed that these infrastructures reach every municipality of the region, even the smallest villages. These infrastructures wouldn't have arrived there so early by the means of the market itself. The most outstanding objective reached through the deployment of the Board of Extremadura's Intranet is the integration of all corporate communications of the Board of Extremadura in a single contract with a consequent improvement of service quality and costs. It also favour the development of Information Society in Extremadura, constitute a true corporate Intranet, improve the quality of administrative services, develop the telecommunication infrastructures in Extremadura and exploit the real bidirectional connectivity at all seats of the Board of Extremadura in the region.

6.4.2 Broadband Internet access for 100% of population

Extremadura is the first region to offer broadband Internet access to 100% of the community, since June 2006. This fact is possible thanks to the contract signed between the Board of Extremadura and Telefónica. Telefónica, being the awarded operator, has expanded coverage of ADSL service to 333 villages taking the obligation to invest all the necessary in order to deploy the Network (20 millon euros). A significant fact is that almost 300 of these villages have less than 1,000 inhabitants. Thus connectivity will be possible from locations not situated in urban areas: rural tourism, cooperatives, service stations, plant nurseries, control stations, etc. Likewise, homes situated outside the urban areas in isolated groupings of houses will also gain Internet access.

6.5 Why Cáceres?

6.5.1 Overview of Cáceres

Extremadura is an autonomous communities in Spain of western Spain. It includes the provinces of Spain of Cáceres (province) and Badajoz (province). Extremadura borders Portugal to the west, and it is an important area for wildlife, particularly with the major reserve at Monfragüe, which has recently been recognised as National Park, or the project of International Tagus River Natural Park (Parque Natural Rio Tajo internacional).

Cáceres is the capital of Cáceres Province, there have been settlements near Cáceres since prehistoric times. Evidence of this can be found in the caves of Maltravieso and El Conejar. The city was founded by the Ancient Rome in 25 BC.

The old town or Ciudad Monumental still has its ancient walls; this part of town is also well known for its multitude of storks' nests. The walls contain a perfect Medieval town setting with no outward signs of modernity for this reason many films have been shot here. The Universidad de Extremadura, and two astronomical observatory are in Cáceres.

Caceres was declared a World Heritage City by UNESCO in 1986 because of the city's blend of Roman, Islamic, Northern Gothic and Italian Renaissance styles, fruit of the many battles fought here throughout history. An amazing 30 towers from the Muslim period still stand in Caceres, of which the Torre del Bujaco is the most famous.

6.5.2 History

The origins of Caceres go back to prehistoric time, as evidenced by the paintings in the Cuevas de Maltravieso (Maltravieso Caves) which date back from the late Paleolithic period. The best way to savour this unique city is to leave your car in one of the carparks located outside the historical centre, and then to stroll around the historical quarters at your leisure to admire the numerous buildings, towers, palaces and plazas most very well conserved or recently restored, dating from the Middle Ages and Rennaisance. Visitors will be able to see remnants from Medieval times, Roman occupation, Moor occupation and Jewish influence. Caceres has four main areas to be explored; the historical quarter, the Jewish quarter, the modern center, and the outskirts.

As mentioned above, the first evidence of humans living in Caceres is from the Late Paleolithic era, around 25,000 B.C. Caceres started to gain importance as a strategic city under Roman occupation, and remains found in the city suggest that it was a thriving center as early as 25 B.C. Some remains of the first wall built around the city by the Romans in III and IV A.D. still exist, including one entrance, Arco del Cristo.

After the end of the Roman Empire, the city was occupied Barbarians and Visigoths and entered a period of decline and decay until the Arabs conquered Caceres in the seven hundreds. The city spent the next few centuries mostly under Arab rule, although power did exchange from Moors to Christians several times. During this time, the Arabs rebuilt the city, including a wall and various towers, including the Torre de Bujaco, and palaces. Caceres was reconquered by the Christians in the 13th century. During this period the city had an important Jewish quarter: in the 15th Century when the total population was 2,000, nearly 140 Jewish families lived in Caceres. The Jewish population was expelled in 1492, but many remnants which are a result of the Jewish influence during this period are still be seen today in the Barrio San Antonio.

Caceres flourished during the Reconquest and the Discovery of America, as influential Spanish families and nobles built homes and small palaces here, and many members of families from Extremadura participated in voyages to America where they made their fortune. In the 19th Century Caceres became the capital of the province, marking a period of growth which was halted by the Spanish Civil War. The headquarters of the University and several regional government departments are to be found in Caceres which today has a population of 90,000 inhabitants.

6.5.3 Monuments

Cathedrals and Churches: Iglesia y Convento de San Pablo, XV Century church and convent; Convento de la Compañía de Jesus barroque style which today is used for art exhibitions; Iglesia yConcatedral de Santa María, cathedral built in XXIII, Gothic period; Iglesia de San Mateo, XV Century church built on the site of a former mosque; Iglesia de San Francisco Javier, Baroque period, XVIII century; Iglesia de San Juan, large magestic church which took five centuries to complete, from 13th to 15th Century; Ermita de San Antonio Iglesia de Santo Domingo; Ermita de la Paz; Iglesia de Santiago

The Wall: Torre de Bujaco XIIc; Arco de la Estrella XVIIIc; Torre de Sande, XIVc-XVc; Torre de los Púlpitos; Torre de la Hierba; Arco de Santa Ana; Torre del Horno; Torre del Postigo; Torre Redonda; Torre Desmochada; Arco del Cristo; Arco del Socorro

Palaces and stately homes: Palacio de los Golfines de Arriba; Palacio de los Golfines de Abajo - one of the most spectacular. The Reyes Católicos, Isabella and Ferdinand, lived here; Palacio del Comendador de Alcuescar; Palacio-Fortaleza de los Torreorgaz, today a Parador hotel, Palacio Episcopal; Palacio de Carvajal XVc; Palacio de Godoy; Palacio de Mayoralgo; Mansión de los Sande; Palacio de las Veletas; Palacio de los Cáceres-Ovando; Casa del Mono; Palacio de los Toledo-Moctezuma; Casa del Sol; Casa Mudejar; Casa de Carvajal y Ulloa.

6.5.4 Natural Parks and rural tourism

Monfrague Natural Park: 85 km. 17,852 hectares, the Parque Natural de Monfragüe contains the following villages: Torrejón el Rubio, Serradilla, Malpartida de Plasencia, Toril, Serrejón, Jaraicejo y Casas de Miravete. With one of the largest forests in Spain with over 1,400 different species of trees. A favourite with birdwatchers, the park has the world's largest colony of black vultures and imperial eagles, and is also home to colonies of black storks, eagle owls, black-shouldered kites, grassland birds including great bustards, sandgrouse...

Chapter 7

Abstracts

7.1 Debian Webservices Development

Frank Lichtenheld

Debian has a lot of useful Web Services that collect and present data about Debian Development. However, they are only loosely integrated, mostly only by including links to each other (e.g. PDO ¡-¿ PTS), or by cron'ed data retrival (e.g. BTS -¿ DDPO, BTS -¿ PTS). While Debian is probably not a community that would like to loose its diversity in Services Development (i.e. One Service To Rule Them All, aka launchpad), better integration between these services is certainly possible and might lead to more efficency and usability. Some ideas that could be discussed at the BoF: 1) central usermanagment (i.e. don't force people to tell every website again and again which packages and maintainer addresses they are interested in; OpenID?) and persistent configuration (i.e. allow people to configure services via cookies, server-side stored information, centrally server-side stored information); 2) Dynamic data retrival; 3) Look (i.e. creating images and CSS that can easily be reused by many services instead of all the more or less different instances we currently have); etc.

7.2 dak discussion / hacking session

Joerg Jaspert

A BoF for people interested in dak, the Debian Archive Kit. Main target is discussion where we want to go, what needs to be in it, what should it do, etc. Also, if time permits (DebCamp lasts a week), also some hacking session implementing some features.

7.3 Emdebian update

Neil Williams

Emdebian GUI configuration and touchscreen support, root filesystem installation methods, remaining issues in cross building Debian and extending the package set and device support.

7.4 Virtualisation in Debian

Jan Lübbe

There are now many virtualization and emulation packages available on Debian. This talk will give an overview over the different approaches, their pros/cons and current states. I will also show where each of them is heading and what that means for Debian.

7.5 Debian on the Neo1973/Freerunner

Jan Lübbe

I'll show how to install Debian on the Neo1973/Freerunner.

7.6 Debian Edu 100% in main

Holger Levsen

This talk will briefly explain what Debian Edu is, how we develop our distribution, how we differ from Debian, how we work on bridging the gaps that still exist and what our plans for the future are.

7.7 SPI BOF

Bdale Garbee

An opportunity to meet board members of Software in the Public Interest who are present at Debconf, and informally discuss the relationship between Debian and SPI, and the future of SPI.

7.8 Managing 666 packages, or how to tame the beast

Martín Ferrari

PET (Package Entropy Tracker) is a collection of scripts that gather information about your (or your group's) packages, based on the SVN repository, but reaching many external sources. It allows you to see in a bird's eye view the health of hundreds of packages, instantly realizing where work is needed.

7.9 dh_make_webapp: yeah right!

Andrew McMillan

Any developers of web applications seem to live in a world of their own. They pull libraries from here there and everywhere written in multiple languages, with varying licenses. They expect the database to work the way their database does, they depend on specific versions of Java, PHP, Python, MySQL or other software and the debugging details seem to go into a black hole. In this talk I will review some of the functionality in and around Debian which can help work around these issues, and I will try and produce a checklist for developers to consider when trying to see if their software is able to be packaged easily.

7.10 Locating bugs to kill with SOAP

Don Armstrong

A tutorial on using the SOAP interface to the BTS as well as other methods of tracking, organizing, modifying, and killing bugs in the interest of developer sanity.

7.11 Ruby packaging in Debian

Lucas Nussbaum

The various Ruby-related teams are facing various challenges: - how are we going to deal with the transition from Ruby 1.8 to Ruby 1.9 ? - what should we do with jruby ?˚- What should we do with gems ?

Ideally, we will have answers to all those questions at the end of this BOF :-)

7.12 Organizing better in-person meetings

Lucas Nussbaum, César Gómez Martín

In-person meetings, such as the ones organized by Extremadura, are a great opportunity for Debian. However, many participants to the various meetings held in the past feel that the way they were organized is suboptimal. For example, some participants prefer to use those meetings for discussions rather than usual Debian work, but then it's difficult to get some other people involved in the discussions. Ideally, the outcome of this BOF will be an HOWTO about how to organize successful meetings. With requirements, tips and tricks to get people involved, etc.

7.13 Debian Derivers Roundtable

Andreas Tille

The Debian-Derivers round-table will bring together representatives of organizations involved in producing Debian derived distributions to discuss the political, organizational, and social barriers to collaboration with Debian and with each other. The round table will include representatives of Canonical and Ubuntu, Skolelinux/Debian-Edu and a representative from the CDD community (e.g., Enrico Zini, Andreas Tille, etc). If available, it may also include representatives from any number of Spanish distributions distributions who may be in attendance (e.g., Guadalinex, Llurex, LinEx), Userlinux, and others. A complete roster will be created once conference attendees have been settled.

7.14 Debian and LiMux

Florian Maier

A meeting with two developers (and aspiring DD's, also ;-) of the Munich's LiMux team to share and discuss some details of our implementation of an open source desktop based on the fantastic Debian distribution. Discussion of possible synergy effects (it would be nice to get some kind of joint effort going). Everybody interested is welcome!

7.15 Debian-Science

Andreas Tille

The huge pool of packaged software inside Debian has the consequence that it also contains a large number of software which is used in day to day work of scientists. But the pure fact that packages are available is not enough to attract scientist who frequently tend to so called "easy to install" distributions and just are not aware which profit they might gain from Debian. Debian-Science has the goal to make Debian really attractive for scientists.

7.16 Synfig - Animation in the free world

Paul Wise

Short presentation about the Synfig animation studio, some of the animations produced by it, call for developers.

7.17 Packaging with version control systems

Martin F. Krafft

Version control systems are becoming more popular for package maintenance. In this talk, I present an overview of current practices and recent developments. I also report on the work of the vcs-pkg.org effort, which tries to identify a workflow for package maintenance which could yield better cross-distro collaboration.

7.18 Debian and Ubuntu

Mark Shuttleworth

Perspectives on collaboration - an analysis of current patterns of collaboration between Debian and one of its largest derivatives, Ubuntu, as well as proposals for additional pathways and processes for better collaboration in general between Debian and its derivatives.

7.19 Herding Wild Cats

Bdale Garbee

A brief history of the Debian project and description of how the key elements of the project evolved and work today, from the perspective of someone who has been involved nearly since the beginning.

7.20 LaTeX Beamer Debian Theme BOF

Andreas Tille

The BOF with the same title was intended to find a consensus about some kind of corporate design for talks using LaTeX Beamer. While there were some specifications done nothing happened regarding implementation. We should meet again and try to work on something.

7.21 Quality Assurance in lenny+1

Lucas Nussbaum

What worked well in Quality Assurance for lenny+1? What didn't? What should we do for lenny+1?

The goal of this BOF is to put everybody interested in QA in the same room, to discuss: - improvements in the way we work: can we be more efficient? What needs to be changed? - which archive-wide tests/mass bug filings are people interested in doing during the lenny+1 release cycle? - MIA and Bapase, or "how to keep track of the dark corners of Debian?" - orphaned packages, and WNPP in general

A short introduction on those topics will be given, so people not familiar with the works of the QA team can participate in the discussions.

7.22 Healthy CDDs

Andreas Tille

The talk will give an overview of the status of the Debian-Med project and how it could work as an example for other CDDs. Considering that specific things about medical software is of quite low interest for the DebConf audience the main focus of the talk will be the way from a one-man idea to a fully grown team that is working contiuosely to enhance Debian for a specific user group. Experiences are shared how good tools and reasonable managemend can help to attract people-users and developers.

7.23 Debian-Med BOF

Andreas Tille

This should be an open discussion for all people interested in free software in health care.

7.24 Method diffusion in large volunteer projects

Martin F. Krafft

This presentation is about ongoing research on innovation diffusion in the Debian project. The goal is to determine the conditions under which volunteers adopt new approaches to everyday challenges and order them into a framework, which can be used prescriptively to help improve the diffusion of certain tools and foster the competition among contenders.

7.25 Best practises in team-maintaining packages

Gregor Herrmann

Team maintenance for (groups of) packages is en vogue; dozens or perhaps hundreds of packaging teams care for a subset of the Debian archive in a collaborative style.

This BOF offers the opportunity for members of different packaging teams to exchange their experiences, share their success and problem stories, and in general learn from each other.

7.26 netconf

Martin F. Krafft

This presentation introduces netconf, a network configuration management system designed with modern network infrastructures and the needs of roaming users in mind. The talk describes netconf's architecture and reports on the progress to date.

7.27 Bringing closer Debian and Rails

Gunnar Wolf

Ruby on Rails has become a very popular framework for Web-based applications. And, even though Rails itself is neatly packaged and integrated in Debian, supporting Rails applications (specially in a large-scale provider) can prove rather difficult. Besides the core application, we face problems such as handling plugins, concurrent versions, and the like. In this BoF session we will try and study the different problems we face, and come up with adequate solutions.

I'm only familiar with Ruby on Rails - but it might be interesting to have the opinion of people working with other similar-minded frameworks.

7.28 Internationalization in Debian

Christian Perrier

This keynote lecture will attempt to give the overall picture of the situation of internationalization and localization in the project. Topics: - i18n team, l10n teams - situation for various regions of the world - relations/non relations with derived distributions - situation in various areas covered by i18n (WWW, packages, documentation...) - i18n infrastructure: where are we?

7.29 Redesigning DEHS (a.k.a. changing the watch files atmosphere)

Raphael Stephane Geissert

DEHS being more than four years old has only been modified to satisfy most of the current needs. Rewriting DEHS in perl with a new design, doing things at the right place (i.e. extraction of watch files), and more should be discussed and input, and code, from several people is needed.

7.30 Debian-HPC: making Debian "the" distro for clusters and supercomputers

Fabricio Cannini Flores

Enrico Zini's talk about world domination was missing an important step: clusters and supercomputers.

7.31 Bits from NMs and users

Paul Wise

Short presentation of the results of these two surveys, and a discussion about what the results mean for Debian.

http://lists.debian.org/debian-mentors/2008/03/msg00030.html

http://lists.debian.org/debian-user/2008/03/msg02475.html

A followup discussion on Ubuntu's strategies for attracting and training new developers and what Debian can learn from that. Also a discussion about how users transition from being purely users to helping develop Debian.

7.32 Multi winner voting in Debian

Manoj Srivastava

This will be a quick update of devotee functionality update to handle clone proof STV voting for multiple winners, which degenerates to our current process for single winner votes. This also serves as a bits from the secretary talk.

7.33 Debian technical policy update

Manoj Srivastava

This talk delivers a progress report of the policy process in Debian, tocuhing the new version control, BTS usetags and user categories, and the progress made in rewriting policy in docbook XML. This can be shorter than an hour if needed, though I can certainly find material to fill the hour.

7.34 Debian Wiki

Franklin Piat

During this work session will focus on two aspects, which should be completed by Lenny's release : Choose a license and get a new site layout.

7.35 Debian: casos de éxito en implementaciones empresariales

José Miguel Parrella Romero

En esta breve presentación se introducirá al participante en las aplicaciones empresariales de Debian, el sistema operativo libre y universal, y como puede una organización de cualquier objeto (pública, privada, académica, defensa, ONG) aprovecharse del modelo de desarrollo colaborativo que caracteriza a la distribución de software libre más grande del Mundo. Se visitarán casos de estudio en América Latina y otras regiones, delineando una estrategia de acercamiento para los participantes que deseen implementar Debian en sus organizaciones. La charla durará entre 45 y 60 minutos, y será realizada en castellano.

In this brief presentation participants will be introduced to enterprise applications of Debian, the free and universal operating system, and how an organization of any type (public, private, academic, defense, NGO) might take profit from the collaborative development approach featured by the biggest free software distribution in the world. Study cases from Latin America and other regions will be visited, scaffolding an approach strategy for those participants willing to deploy Debian in their institutions. The talk will take between 45 and 60 minutes and will be given in spanish.

7.36 Internacionalización en Debian

César Gómez Martín

Debian contiene numerosas aplicaciones software que tienen la característica de procesar datos en forma de texto, la mayoría de estas aplicaciones asumen que el texto está escrito en inglés (ASCII). El problema aparece cuando la gente cuyo idioma principal no es inglés intenta usar Debian. Aunque la mayoría del software que contiene Debian maneja ISO-8859-1 además de ASCII, algunas de ellas no pueden manejar caracteres de idiomas como chino, japonés o coreano.

Es absurdo pensar que una persona que quiere utilizar un ordenador debe aprender antes a hablar inglés, ahí es donde entra en juego la internacionalización de Debian.

7.37 Proyecto gnuLinEx

César Gómez Martín

En 1997 Extremadura comenzó a trabajar en la Estrategia Regional de la Sociedad de la Información en Extremadura (Proyecto Europeo Infodex). En 1998 el Presidente de Extremadura propuso una estrategia para el modelo de desarrollo regional apoyándose en las posibilidades que ofrecen las Tecnologías de la Información y las Comunicaciones. Las acciones estratégicas eran: 34 centros de alfabetización tecnológica, 1 PC para cada 2 estudiantes de secundaria en 2005, 60 nuevas empresas creadas en un vivero de empresas y una feria tecnológica anual.

Para lograr estos propósitos Extremadura necesitaba adaptabilidad, viabilidad, seguridad y acceso universal de los ciudadanos a las herramientas. Esta es la razón por la que se escogió desarrollar un sistema operativo libre llamado gnuLinEx basado en Debian.

7.38 ¿Qué es el Software Libre?

Gunnar Wolf

Hay una duda muy común al contemplar nuestro trabajo, que mucha gente puede dar por obvia: ¿Qué es el software libre? ¿Qué hay detrás de nuestro movimiento? ¿Qué implicaciones tiene?

Posiblemente ya has asistido a pláticas que te presentan las famosas "cuatro libertades" y te presentan al Software Libre desde un punto de vista técnico. En esta plática, más bien, intento responder a estas dudas dando un énfasis diferente: elaboro sobre del Conocimiento Libre, y en el software como una expresión del conocimiento, de la ciencia, del desarrollo histórico de la humanidad.

7.39 Hosting Caseros

Sebastian Montini

Durante esta presentación se explicará cómo configurar un servidor web con bases de datos y un gestor de contenidos utilizando Software Libre (CMS:

Joomla, Wordpress, Drupal o Wiki). Además, se explicará la configuración de los servicios de dns dinámicos para poner en marcha un servidor web con una PC y conexión a internet hogareña.