Topper Project*

Dario Rapisardi

27th February 2006

Abstract

The software and hardware dependencies involved in the Free Software domain is complex, given the diversity and development models of software developers and hardware vendors. The Topper Project aims to build an Expert System capable of gathering the diverse experiences in this field, and help to isolate compatibility problems.

Contents

*See Appendix A for a reference about the project's name.

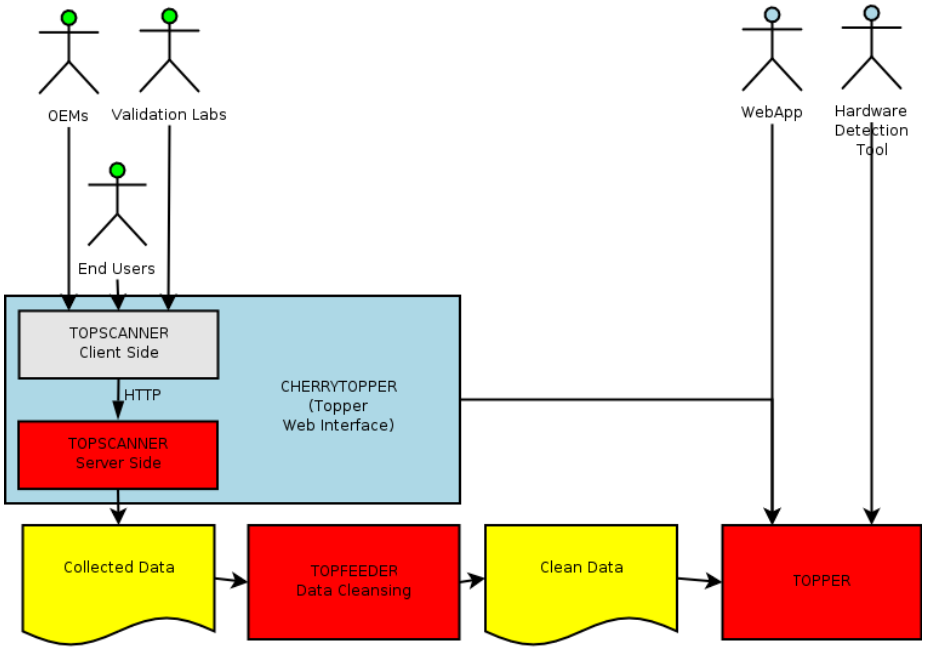
1 Introduction

At this very moment, a lot of information regarding hardware compatibility is being lost due to the lack of efficient information systems. Topper in itself is not a hardware detection tool, or a hardware database. It's an information agent part of a bigger system, capable of being used by third party tools such as HDT (Hardware Detection Tools) or WebApps.

2 Topper Internals

2.1 Mode of operation

We can identify the general mode of operation and the general functions of Topper as follows:



2.2 Data gathering

The data gathering process basically consists of an actor making some tests and delivering results. Three big groups of actors can be identified:

Actors
OEMs
End Users
Validation Laboratories

While OEMs and Validation Labs can provide factual, formal results to the tests, the end users can provide massive, spontaneous amounts of data. Because of this, all of the data must have the same, structured format, so Topper can properly understand it.

There are basically three types of data, from a Data Gathering point of view:

Hardware Data: This is information relevating to the devices present in a system, for example PCI devices.

Software Data: This is information about the software installed in a system, for example software packages.

Features: This are the results of the tests in itself. They involve several boolean values of the type “Working” or “Not Working”. Because of the subjective

> nature of this, the reasons behind the consideration of a feature "working" or not is outside the scope of Topper, and must be introduced by the Actors delivering the data. Nevertheless, errors and inconsistencies could be properly cleaned in a Machine Learning Process if desired. The Machine Learning process, however, is currently outside the scope of Topper.[1]

However, for Topper there's no difference between a Hardware Data, a Software Data or a Feature. For Topper, each of this things is a *Fact*, something present on a system. The summary of *Facts* define a *Behaviour*. When someone asks Topper about information, it asks for *Behaviours*, that will return sets of *Facts* that might produce them.

For simple data gathering and input, Topper provides a mechanism served by an application called *Topscanner*.

2.2.1 Topscanner

Topscanner is a set of tools aimed for easy data gathering. It's based on two tools, one for client side data gathering, and the other one as a server side application aimed for Training of the data system and data feeding.

The client side of the application collects every aspect of hardware-software related components present in a system, and creates a file called "topscanner.txt" with all the facts of the tested system.

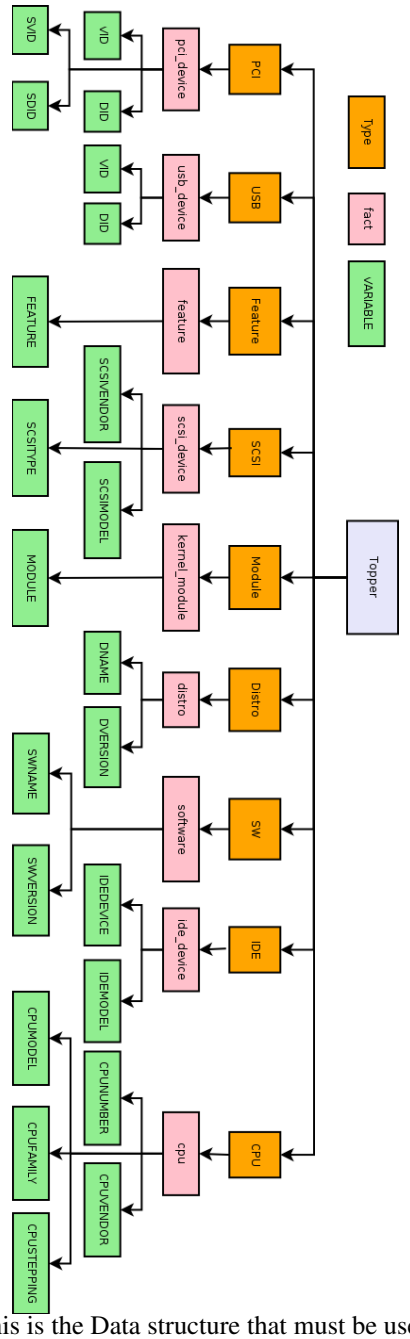
The server side is a Web frontend, where the Actor can upload the Facts file and then follow an on-line wizard about different behaviours (or features) of the system, based on the uploaded file. This way, End Users can train the Expert System and input data easily.

Full Facts files with all the information can still be provided the usual way (directly) to the system. Topscanner is just aimed for interactive training of the system in an easy way.

[1] Refer to Appendix B for more details about this topic.

2.3 Data structure

Topper' s data structure includes several types of buses and device types (facts) found in a system. The relationship between these facts can be seen as follows:



This is the Data structure that must be used to input Data in Topper. A Data Gathering tool (like *Topscanner)* should output this format.

2.3.1 PCI Bus

The pci bus information can be entered using the following fact format:

```
pcidevice(PCICLASS,VID,DID,SVID,SDID).
```

Being:

PCICLASS: The Class of the device in hexadecimal. Eg: 0x280 = Network Controller.

VID: Vendor ID in hexadecimal. Eg: 0x8086.

DID: Device ID in hexadecimal.

SVID: SubVendor ID in hexadecimal, if any. Otherwise, 0x0000.

SDID: SubDevice ID in hexadecimal, if any. Otherwise, 0x0000.

One entry per pci device must be introduced.

2.3.2 USB Bus

The usb bus information can be introduced using the following fact format:

```
usbdevice(VID,DID).
```

Being:

VID: Vendor ID in hexadecimal. For example: 0x8086.

DID: Device ID in hexacecimal.

One entry per usb device attached to the system must be introduced.

2.3.3 Software

Information about the software installed in a system can be introduced using the following fact format:

```
software(SWNAME,SWVERSION).
```

Being:

SWNAME: Software name, in string format. The recommended values are: “xorg”, “xfree86”, “linux”, “alsa”, “hotplug”, “udev”, “libc6”, “gcc”, “sane”, “cups”, “lpr”, “lprng”, “fglrx”, “nvidia”, “wireless-tools”, “lilo”, “grub”.

SWVERSION: Version of the software installed in string format. For example, “2.6.11-2”, for a “linux” software installed.

One entry per software installed from the recommended list must be introduced.

2.3.4 Distribution

Information about the GNU/Linux distribution used in the tested machine can be introduced using the following fact format:

```
distribution(DNAME,DVERSION).
```

Being:

DNAME: Distribution name in string format. For example “gnuLinEx”, “guadalinex”, “debian”, “xandros”, “linspire”.

DVERSION: Version of the distribution used for the test. For example “2004r1” for gnuLinEx, “2004” for Guadalinex, etc.

Only one fact about the distribution is allowed.

2.3.5 Kernel modules

Information about the kernel modules loaded at test time can be introduced using the following format:

```
kmodule(KERNELMODULE).
```

Being:

KERNELMODULE: The name of the loaded kernel module at test time, without extension. For example, “id-cdrom”, “snd-hda-intel”, etc.

One fact per kernel module loaded must be introduced.

2.3.6 IDE Devices

Information about the IDE devices installed in the tested system can be informed, if any, using the following format:

```
ide(IDEDEVICE,IDEMODEL).
```

Being:

IDEDEVICE: IDE device name, as recognized by the Linux kernel. For example, “hda”.

IDEMODEL: IDE device model, as found in the /proc/ide/$IDEDEVICE/model file, for example “IC25N030ATMR04-0”.

One fact per IDE device present in the system must be introduced, if any.

2.3.7 SCSI Devices

Information about the SCSI devices installed in the system can be informed, if any, using the following format:

```
scsi(SCSIVENDOR,SCSIMODEL,SCSITYPE).
```

Being:

SCSIVENDOR: SCSI device vendor, as found in /proc/scsi/device_info. Eg: “SONY”.

SCSIMODEL: SCSI device model, as found in /proc/scsi/device_info. Eg: “CD-ROM CDU-8001”.

SCSITYPE: SCSI device type, in hexadecimal. Eg: 0x4.

One fact per SCSI device present in the system must be introduced, if any.

2.3.8 Features

The collective behaviour of the hardware and software installed in a system defines different types of features. This behaviours are boolean values, meaning “Working” or “Not working”. In essence, this are the results of the tests, are outside the scope of Topper and must be defined by the procedures involved in the testing procedures.

The features can be introduced in using the following format:

```
feature(FEATURE).
```

The variable FEATURE can have several values. The following features can be found in a system:

bluetooth: System has a bluetooth port and is working.

boot: If system boots.

cardbus: System has a cardbus controller, and is working.

digitizer-pen: Digitizer Pen is working.

display: Video is working.

display-dri: X environment is working, with Direct Rendering support.

display-3party: X environment is working, with 3rd party drivers.

display-3d: X environment is working, with OpenGL acceleration.

display-x: X environment is working.

display-xrandr-resize: X environment is working, with XRandR support, and resizing works.

display-xrandr-rotate: X environment is working, with XRandR support, and rotating works.

floppy: Floppy drive is working.

gameport: Game Port (joystick) is working.

hd: Hard disk is working.

ide: System has an IDE controller, working.

ide-dma: System has an IDE controller, and DMA is working.

ide-pata: System has an IDE Parallel ATA controller, working.

ide-sata: System has an IDE Serial ATA controller, working.

ieee1394: System has a iee1394 port and is working.

irda: System has an infrared port, and is working.

keyboard: Keyboard is working.

mass-storage: System has a generic mass storage device, and is working.

modem: System has a modem and is working.

mouse: Mouse is working.

multimedia-audio: System has a sound card, working.

multimedia-audio-hda: System has a sound card, and High Definition Audio is working.

multimedia-audio-mic: System has a sound card, and micrphone plug is working.

multimedia-audio-midi: System has a sound card, and midi port is working.

multimedia-audio-multichannel: System has a sound card, and multichannel is working.

multimedia-video: System has a Video 4 Linux (V4L) device attached, and is working. a

networking-atm: System as an ATM interface, and is working.

networking-eth: System has a wired NIC and is working.

networking-fddi: System has a FDDI interface, and is working.

networking-isdn: System has an ISDN interface, working.

networking-tr: System has a Token Ring interface, and is working.

optical: System has optical drives attached and working.

parport: System has a parallel port and is working.

pcmcia: System has a pcmcia controller, and is working.

printer: System has a printer attached and is working.

raid: System has a RAID controller and is working.

scsi: System has a SCSI controller, working.

scanner: System has a scanner attached and is working.

serial: System has a serial port and is working.

serial-multiport: System has a multiport serial board, and is working.

usb: System has a working USB 1.1 compatible usb controller.

usb2: System has a working USB 2 compatible usb controller.

usb-storage: System has support for usb-storage devices.

wifi: System has a wireless 802.11x NIC and is working.

wifi-80211ab: System has a 802.11a/b capable, working, wireless interface.

wifi-80211g: System has a 802.11g capable, working, wireless interface.

wifi-adhoc: System has a working wireless NIC, with Ad-Hoc mode support.

wifi-firmware: System has a working wireless 802.11X NIC, with additional firmware installed.

wifi-managed: System has a working wireless interface, with Infraestructure mode support.

wifi-master: System has a working wireless interface, with Master mode support.

wifi-monitor: System has a working wireless interface, with Monitor mode support.

wifi-wep: System has a WEP capable and working wireless network interface.

One fact per working feature must be introduced.

2.3.9 CPUs

Information about the CPUs installed in the tested system can be informed using the following format:

```
cpu(CPUNUMBER,CPUVENDOR,CPUFAMILY,CPUMODEL,CPUSTEPPING).
```

Being:

CPUNUMBER: The number of the installed CPU, being 0 the first CPU, 1 the second one, etc.

CPUVENDOR: The Vendor ID of the CPU as */proc/cpuinfo*. For example, *GenuineIntel.*

CPUFAMILY: CPU Family number. Eg: 6.

CPUMODEL: CPU Model number. Eg: 8.

CPUSTEPPING: CPU Stepping number. Eg: 6.

2.3.10 Example

An example from a test result can look like this, for a i855 system running juegaLinEx LINEXTREMIX-1:

```
pcidevice(0x1180,0x0552,0x0000,0x0000).
pcidevice(0x8086,0x24c7,0x0000,0x0000).
pcidevice(0x8086,0x3582,0x0000,0x0000).
pcidevice(0x8086,0x3582,0x0000,0x0000).
pcidevice(0x8086,0x24ca,0x0000,0x0000).
pcidevice(0x8086,0x24cc,0x0000,0x0000).
pcidevice(0x8086,0x24c6,0x0000,0x0000).
pcidevice(0x8086,0x24c5,0x0000,0x0000).
pcidevice(0x8086,0x4220,0x0000,0x0000).
pcidevice(0x8086,0x24c3,0x0000,0x0000).
pcidevice(0x8086,0x2448,0x0000,0x0000).
pcidevice(0x8086,0x24c2,0x0000,0x0000).
pcidevice(0x8086,0x24c4,0x0000,0x0000).
pcidevice(0x8086,0x3585,0x0000,0x0000).
pcidevice(0x8086,0x3580,0x0000,0x0000).
pcidevice(0x8086,0x3584,0x0000,0x0000).
pcidevice(0x8086,0x103d,0x0000,0x0000).
pcidevice(0x1180,0x0476,0x0000,0x0000).
pcidevice(0x8086,0x24cd,0x0000,0x0000).
pcidevice(0x1180,0x0476,0x0000,0x0000).
usbdevice(0x045e,0x0040).
software(gcc,'4:3.3.5-1').
software(wireless-tools,'27-1').
software(grub,'0.95+cvs20040624-12').
software(linux,'2.6.7').
software(alsa-base,'1.0.9b-4linex0').
software(hotplug,'0.0.20040329-16').
software(xserver-xfree86,'4.3.0.dfsg.1-8').
software(libsane,'1.0.15-2').
software(libc6,'2.3.2.ds1-18').
software(cupsys,'1.1.20final+rc1-10').
distribution(debian,gnulinex2004r1).
kmodule(i830).
kmodule(snd_mixer_oss).
kmodule(ds).
kmodule(lp).
kmodule(thermal).
kmodule(fan).
kmodule(button).
kmodule(processor).
kmodule(ac).
kmodule(battery).
kmodule(af_packet).
kmodule(arc4).
kmodule(ieee80211_crypt_wep).
kmodule(crc32).
kmodule(p80211).
kmodule(eepro100).
kmodule(ipw2200).
kmodule(firmware_class).
kmodule(ieee80211).
kmodule(ieee80211_crypt).
kmodule(snd_intel8x0m).
kmodule(slamr).
kmodule(pciehp).
kmodule(shpchp).
kmodule(pci_hotplug).
kmodule(usbhid).
kmodule(uhci_hcd).
kmodule(intel_agp).
kmodule(eth1394).
kmodule(mousedev).
kmodule(joydev).
kmodule(tsdev).
kmodule(evdev).
kmodule(e100).
kmodule(mii).
kmodule(ohci1394).
```

```
kmodule(ieee1394).
kmodule(yenta_socket).
kmodule(pcmcia_core).
kmodule(snd_intel8x0).
kmodule(snd_ac97_codec).
kmodule(snd_pcm).
kmodule(snd_timer).
kmodule(snd).
kmodule(soundcore).
kmodule(snd_page_alloc).
kmodule(ehci_hcd).
kmodule(usbcore).
kmodule(supermount).
kmodule(dm_mod).
kmodule(ide_cd).
kmodule(cdrom).
kmodule(parport_pc).
kmodule(parport).
kmodule(rtc).
kmodule(xfs).
kmodule(jfs).
kmodule(reiserfs).
kmodule(isofs).
kmodule(ext2).
kmodule(ext3).
kmodule(jbd).
kmodule(mbcache).
kmodule(ide_disk).
kmodule(ide_generic).
kmodule(piix).
kmodule(ide_core).
kmodule(unix).
kmodule(sata_via).
kmodule(sata_sx4).
kmodule(sata_svw).
kmodule(sata_sis).
kmodule(sata_sil).
kmodule(sata_promise).
kmodule(ata_piix).
kmodule(libata).
kmodule(scsi_mod).
ide(hdc,dv-w28e).
ide(hda,ic25n030atmr04-0).
cpu('0','GenuineIntel','6','13','6').
feature(audio).
feature(hd).
feature(boot).
feature(video).
feature(mouse).
feature(usb-storage).
feature(iee1394).
feature(printer).
feature(wi-fi).
```

2.4 Data Cleansing

Topper sources of data can be wide and diverse. For that reason, a good mechanism to clean the data introduced to the system must be applied. Basically, this mechanism will involve several steps, like looking for dirty data, duplicates, incomplete, redundant or irrelevant. As indicated in the Mode of operation chart, Actors introducing data can be diverse, such as OEMs making tests, Validations Labs and End Users. Classifying this sources of information by quality, the following levels can be defined:

- Highest quality: The validation tests made by Intel and/or validation/certification centers.
- Good quality: The information about hardware present in the kernel. This kind of information is not very useful in terms of Topper point of view, since it lacks of

different attributes, such as supported software versions, dependencies between user-space and kernel-space packages (eg. *udev, initramfs)*, and dependencies involved when some of the kernel modules are provided as separate packages.

- Poor quality: The data introduced by end users, meaning by *end users* Community and OEMs.

Given that Topper has to deal with all this kind of sources, the highest level of Data Cleansing should be applied. The good opposite about this scenario, is that big amounts of information can be handled with certain reliability.

For that purpose, six steps are involved:

- Syntax errors cleaning.
- Cleaning of empty attributes.
- Removing attributes with ilogical values.
- Elimination of non relevant attributes.
- Testing and evaluation.
- Selection of Data Views by attributes (attribute=X), creating subsets of data classificated by features.

2.4.1 Mode of operation

All of the tests must be done off-line. This means, in a batch process, applied to all the data collected every certain time. Since Topper's philosophy is to treat all data as equal, the problems involved with data testing and manipulation is similar to those found in Data Mining and Machine Learning situations. In machine learning, one should not learn and test classifiers on the same data set. For that reason, the data should be splitted and used some of them for training, and the rest for testing. The good thing about this mechanism is that once that some data introduced by the Highest Quality sources (eg. Intel), it can be used to train the system and test the other set of data.

A description of the steps involved in the Data Cleansing process follows.

Syntax error cleaning

This process just parses the input tests and look for syntax errors that would conflict with the normal operation of the testing suites.

Cleaning of empty attributes

Here's a snippet of an example about how to remove tests with missing values (a little bit reduced for space purposes) with Orange:

```
import orange
data = orange.ExampleTable("topscanner.txt")
boot, hd, ide, pci, cpu = data.domain.variables
data2 = orange.Preprocessor_dropMissing(data)
```

Removing attributes with ilogical values

As the previous example, now let's see how to remove tests with ilogical values with Orange:

```
import orange
data = orange.ExampleTable("topscanner.txt")
boot, hd, ide, pci, cpu = data.domain.variables
pp = orange.Preprocessor_drop()
pp.values[boot] = "no"
data2 = pp(data)
```

Elimination of non relevant attributes

Most of the data collected by Actors might not be of interest to the application level. There's a way to reduce the size of the data base, eliminating non relevant attributes. This is done eliminating attributes recursively using Relief measure, until the estimate relevants of all attributes is beyond certain threshold. In the following example let's see a margin of *marg=0.01* as an example:

```
import orange, orngFSS
data = orange.ExampleTable("topscanner.txt")
marg = 0.0.1
ndata = orngFSS.filterRelief(data, margin=marg)
```

Testing and Evaluation

As said, one should not learn and test classifiers from the same data set. One could split the data in half, or use some of the highest quality data to train the system and cross validate against the new data. In the next example, let's see this scenario, applying a Bayes learner and a Classification Tree learner.

```
import orange, orgnTree
train_data = orange.ExampleTable("factslist.pro")
test_data = orange.ExampleTable("topscanner.txt")
bayes = orange.BayesLearner(train_data)
tree = orngTree.TreeLearner(train_data)
bayes.name = "bayes"
tree.name = "tree"
classifiers = [bayer, tree]
#Compute accuracies
correct = [0,0]*len(classifiers)
for ex in test_data:
  for i in range(len(classifiers)):
    if classifiers[i](ex) == ex.getclass():
      correct[i] += 1
  for i in range(len(correct)):
    correct[i] = correct[i]/len(test_data)
#Print accuracies
for i in range(len(classifiers)):
  print classifiers[i].name, acc[i]
```

Unfortunately, a good data set is needed to determine the efficiency of the Learners.

Selection of Data Views by attributes

The selection of Data Views is useful when large amounts of data are being used for several and different purposes. For example, from the same data set some application might be interested in seeing the relationship between Chipset families and Software packages, while other user might be interested in the relationship between Wi-Fi cards and software packages. In both cases, information such as loaded kernel modules and scsi drives can be discarded.

This can be easily done with Orange and selecting data tests by attributes. In the example, let's see how to make a subset from tests involvinf wi-fi, software and distribution:

```
import orange
data = orange.ExampleTable("topscanner.txt")
newData2 = data.select(['wi-fi', 'software', 'distribution'])
```

This allows the generation of smaller groups, allowing a faster processing of data at the application level.

2.4.2 Tools

Topper uses the Orange[2] data mining software as a backend for data cleansing purposes. Orange is a data mining software with a range of preprocessing, modelling and data exploration techniques. This components can be accesed through Python scripts.

2.5 Data Processing

Collected data might be introduced to Topper in the form of facts, with the format seen in the previous example. Once introduced, Topper assigns it a Unique Test Identifier and builds the relationships between the Data. The Unique Test Identifier it's just a unique string assigned by the Topper engine.

To introduce the data, the *parser* python module can be used, to input the data in a plain test format like seen in the previous section, like this:

```
import parser
factslists = parser.Parser()
```

Then, we can pass the plain text file to the *Parser* directly:

```
file = "topscanner.txt"
factslists.addTest(file)
```

After this, the Parser will generate a new Unique Test Identifier and add the Test to the Database.

Once processed by Topper, the previous example will look like this inside Topper's database:

```
has_tested('test10571').
has_pci_device('test10571', pcidevice('0x1180','0x0552','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24c7','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x3582','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x3582','0x0000','0x0000')).
```

[2]http://www.ailab.si/orange

```
has_pci_device('test10571', pcidevice('0x8086','0x24ca','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24cc','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24c6','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24c5','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x4220','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24c3','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x2448','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24c2','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24c4','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x3585','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x3580','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x3584','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x103d','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x1180','0x0476','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x8086','0x24cd','0x0000','0x0000')).
has_pci_device('test10571', pcidevice('0x1180','0x0476','0x0000','0x0000')).
has_usb_device('test10571', usbdevice('0x045e','0x0040')).
has_software('test10571' ,software('gcc','4:3.3.5-1')).
has_software('test10571' ,software('wireless-tools','27-1')).
has_software('test10571' ,software('grub','0.95+cvs20040624-12')).
has_software('test10571' ,software('linux','2.6.7')).
has_software('test10571' ,software('alsa-base','1.0.9b-4linex0')).
has_software('test10571' ,software('hotplug','0.0.20040329-16')).
has_software('test10571' ,software('xserver-xfree86','4.3.0.dfsg.1-8')).
has_software('test10571' ,software('libsane','1.0.15-2')).
has_software('test10571' ,software('libc6','2.3.2.ds1-18')).
has_software('test10571' ,software('cupsys','1.1.20final+rc1-10')).
is_distro('test10571', distribution('debian','gnulinex2004r1')).
has_kmodule('test10571', kmodule('snd_pcm_oss')).
has_kmodule('test10571', kmodule('i830')).
has_kmodule('test10571', kmodule('snd_mixer_oss')).
has_kmodule('test10571', kmodule('ds')).
has_kmodule('test10571', kmodule('lp')).
has_kmodule('test10571', kmodule('thermal')).
has_kmodule('test10571', kmodule('fan')).
has_kmodule('test10571', kmodule('button')).
has_kmodule('test10571', kmodule('processor')).
has_kmodule('test10571', kmodule('ac')).
has_kmodule('test10571', kmodule('battery')).
has_kmodule('test10571', kmodule('af_packet')).
has_kmodule('test10571', kmodule('arc4')).
has_kmodule('test10571', kmodule('ieee80211_crypt_wep')).
has_kmodule('test10571', kmodule('crc32')).
has_kmodule('test10571', kmodule('p80211')).
has_kmodule('test10571', kmodule('eepro100')).
has_kmodule('test10571', kmodule('ipw2200')).
has_kmodule('test10571', kmodule('firmware_class')).
has_kmodule('test10571', kmodule('ieee80211')).
has_kmodule('test10571', kmodule('ieee80211_crypt')).
has_kmodule('test10571', kmodule('snd_intel8x0m')).
has_kmodule('test10571', kmodule('slamr')).
has_kmodule('test10571', kmodule('pciehp')).
has_kmodule('test10571', kmodule('shpchp')).
has_kmodule('test10571', kmodule('pci_hotplug')).
has_kmodule('test10571', kmodule('usbhid')).
has_kmodule('test10571', kmodule('uhci_hcd')).
has_kmodule('test10571', kmodule('intel_agp')).
has_kmodule('test10571', kmodule('eth1394')).
has_kmodule('test10571', kmodule('mousedev')).
has_kmodule('test10571', kmodule('joydev')).
has_kmodule('test10571', kmodule('tsdev')).
has_kmodule('test10571', kmodule('evdev')).
has_kmodule('test10571', kmodule('e100')).
has_kmodule('test10571', kmodule('mii')).
has_kmodule('test10571', kmodule('ohci1394')).
has_kmodule('test10571', kmodule('ieee1394')).
has_kmodule('test10571', kmodule('yenta_socket')).
has_kmodule('test10571', kmodule('pcmcia_core')).
has_kmodule('test10571', kmodule('snd_intel8x0')).
has_kmodule('test10571', kmodule('snd_ac97_codec')).
has_kmodule('test10571', kmodule('snd_pcm')).
has_kmodule('test10571', kmodule('snd_timer')).
has_kmodule('test10571', kmodule('snd')).
has_kmodule('test10571', kmodule('soundcore')).
```

```
has_kmodule('test10571', kmodule('snd_page_alloc')).
has_kmodule('test10571', kmodule('ehci_hcd')).
has_kmodule('test10571', kmodule('usbcore')).
has_kmodule('test10571', kmodule('supermount')).
has_kmodule('test10571', kmodule('dm_mod')).
has_kmodule('test10571', kmodule('ide_cd')).
has_kmodule('test10571', kmodule('cdrom')).
has_kmodule('test10571', kmodule('parport_pc')).
has_kmodule('test10571', kmodule('parport')).
has_kmodule('test10571', kmodule('rtc')).
has_kmodule('test10571', kmodule('xfs')).
has_kmodule('test10571', kmodule('jfs')).
has_kmodule('test10571', kmodule('reiserfs')).
has_kmodule('test10571', kmodule('isofs')).
has_kmodule('test10571', kmodule('ext2')).
has_kmodule('test10571', kmodule('ext3')).
has_kmodule('test10571', kmodule('jbd')).
has_kmodule('test10571', kmodule('mbcache')).
has_kmodule('test10571', kmodule('ide_disk')).
has_kmodule('test10571', kmodule('ide_generic')).
has_kmodule('test10571', kmodule('piix')).
has_kmodule('test10571', kmodule('ide_core')).
has_kmodule('test10571', kmodule('unix')).
has_kmodule('test10571', kmodule('sata_via')).
has_kmodule('test10571', kmodule('sata_sx4')).
has_kmodule('test10571', kmodule('sata_svw')).
has_kmodule('test10571', kmodule('sata_sis')).
has_kmodule('test10571', kmodule('sata_sil')).
has_kmodule('test10571', kmodule('sata_promise')).
has_kmodule('test10571', kmodule('ata_piix')).
has_kmodule('test10571', kmodule('libata')).
has_kmodule('test10571', kmodule('scsi_mod')).
has_ide_device('test10571', ide('hdc','dv-w28e')).
has_ide_device('test10571', ide('hda','ic25n030atmr04-0')).
has_cpu('test10571', cpu('0','GenuineIntel','6','13','6')).
has_feature('test10571', feature('boot')).
has_feature('test10571', feature('hd')).
has_feature('test10571', feature('audio')).
has_feature('test10571', feature('video')).
has_feature('test10571', feature('accel')).
has_feature('test10571', feature('mouse')).
has_feature('test10571', feature('keyboard')).
```

As we can see in the example, it's assumed that a set of facts is what defines the global behaviour, with no particular precedence.

Let's take a look at each of this rules:

2.5.1 has_pci_device

The *has_pci_device(TEST, PCI)* rule accepts a *TEST* and a *PCI* elements, being *PCI* of the form: *pcidevice(PCICLASS, VID, DID, SVID, SDID):*

```
has_pci_device('test_identifier', pcidevice('pciclass','vid','did','svid','sdid')).
```

2.5.2 has_usb_device

The *has_usb_device(TEST, USB)* rule accepts a *TEST* and a *USB* elements, being *USB* of the form: *usbdevice(VID, DID)*:

```
has_usb_device('test_identifier', usbdevice('vid','did')).
```

2.5.3 has_software

The *has_software(TEST, SOFTWARE)* rule accepts a *TEST* and a *SOFTWARE* elements, being *SOFTWARE* of the form: *software(SWNAME, SWVERSION)*:

```
has_software('test_identifier', software('swname','swversion')).
```

2.5.4 is_distro

The *is_distro(TEST, DISTRIBUTION)* rule accepts a *TEST* and a *DISTRIBUTION* elements, being *DISTRIBUTION* of the form: *distribution(DNAME, DVERSION)*:

```
is_distro('test_identifier', distribution('dname','dversion')).
```

2.5.5 has_kmodule

The *has_kmodule(TEST, KERNELMODULE)* rule accepts a *TEST* and a *KERNELMODULE* elements, being *KERNELMODULE* of the form: *kmodule(KERNELMODULE)*:

```
has_kmodule('test_identifier', kmodule('kernelmodule')).
```

2.5.6 has_ide_device

The *has_ide_device(TEST, IDE)* rule accepts a *TEST* and a *IDE* elements, being *IDE* of the form: *ide(IDEDEVICE, IDEMODEL)*:

```
has_ide_device('test_identifier', ide('idedevice','idemodel')).
```

2.5.7 has_feature

The *has_feature(TEST, FEATURE)* rule accepts a *TEST* and a *FEATURE* elements, being *FEATURE* of the form: *feature(FEATURE)*:

```
has_feature('test_identifier', feature('feature')).
```

2.5.8 has_cpu

The *has_cpu(TEST, CPU)* rule accepts a *TEST* and a *CPU* elements, being *CPU* of the form: *cpu(CPUNUMBER, CPUVENDOR, CPUFAMILY, CPUMODEL, CPUSTEPPING)*:

```
has_cpu('test_identifier', cpu('cpunumber', 'cpuvendor',
  'cpufamily', 'cpumodel', 'cpustepping')).
```

3 Interaction with Topper

As already noticed, Topper facts are written in Prolog. Besides Prolog's ability to create powerful expert systems, this language offers bindings for the most popular programming languages. As an intermediate tool aimed to be used as a helper for others, this seemed very appropiate.

While Topper facts are in Prolog, the main engine is written in Python, and can be accessed as a module to use it. Just need to import the proper module:

```
from topper import
```

Then, to make an instance of it, the main file with the facts database can be passed to the class. By default it uses *"factslist.pro"*:

```
topper = Topper()
topper.reloadFacts()
```

The *reloadFacts()* method makes topper re-read the facts file and load them in memory. Now the engine is running.

Topper has several predefined methods to get information, and one method for Free Queries, as described below:

3.1 Predefined Methods

The following are the predefined methods in Topper, to provide easy access to common data.

3.1.1 getAllPci()

The *topper.getAllPci()* method returns all the PCI devices Topper knows about, in a list. Each element of the list contains a tuple with five elements: *PCICLASS, VID, DID, SVID* and *SDID.*

3.1.2 getAllUsb()

The *topper.getAllUsb()* method returns all the USB devices Topper knows about, in a list. Each element of the list contains a tuple with two elements: *VID* and *DID*.

3.1.3 getAllSoftware()

The *topper.getAllSoftware()* method returns all the Software packages Topper knows about, in a list. Each element of the list contains a tuple with two elements: *SWNAME* and *SWVERSION*.

3.1.4 getAllDistros()

The *topper.getAllDistros()* method returns all the Distributions Topper knows about, in a list. Each element of the list contains a tuple with two elements: *DNAME* and *DVERSION*.

3.1.5 getAllKModules()

The *topper.getAllKModules()* method returns all the Kernel Modules Topper knows about, in a list. Each element of the list contains a *KERNELMODULE* element (a module name).

3.1.6 getAllFeatures()

The *topper.getAllFeatures()* method returns all the Features Topper knows about, in a list. Each element of the list contains a FEATURE element (a feature name).

3.1.7 getAllIde()

The *topper.getAllIde()* method returns all the IDE Devices Topper knows about, in a list. Each element of the list contains a tuple with two elements, *IDEDEVICE* and *IDEMODEL*.

3.1.8 getAllCpus()

The *topper.getAllCpu()* method returns all the CPUs Topper knows about, in a list. Each element of the list contains a tuple with five elements, *CPUNUMBER, CPUVENDOR, CPUFAMILY, CPUMODEL* and *CPUSTEPPING*.

3.1.9 getTestFromPci(vid, did, svid, sdid)

The *topper.getTestFromPci(vid = "0x0000", did = "0x0000", svid = "0x0000", sdid = "0x0000")* method returns all the Tests that match with the given PCI device. The return format is a list with the Tests names (tests unique identifiers).

3.1.10 getTestFromUsb(vid, did)

The *topper.getTestFromUsb(vid = "0x0000", did = "0x0000")* method returns all the Tests that match with the given USB device. The return format is a list with the Tests names (tests unique identifiers).

3.1.11 getTestFromSoftware(swname, swversion)

The *topper.getTestFromSoftware(swname = "", swversion = "")* method returns all the Tests that match with the given software package. The return format is a list with the Tests names (tests unique identifiers).

3.1.12 getTestFromDistro(dname, dversion)

The *topper.getTestFromDistro(dname = "", dversion = "")* method returns all the Tests that match with the given distribution name and version. The return format is a list with the Tests names (tests unique identifiers).

3.1.13 getTestFromKModule(kmod)

The *topper.getTestFromKModule(kmod = "")* method returns all the Tests that match with the given kernel module name. The return format is a list with the Tests names (tests unique identifiers).

3.1.14 getTestFromIde(idedevice, idemodel)

The *topper.getTestFromIde(idedevice = "", idemodel = "")* method returns all the Tests that match with the given IDE device and model. The return format is a list with the Tests names (tests unique identifiers).

3.1.15 getTestFromFeature(feat)

The *topper.getTestFromFeature(feat = "")* method returns all the Tests that match the given feature. The return format is a list with the Tests names (tests unique identifiers).

3.1.16 getTestFromCpu(cpunumber, cpuvendor, cpufamily, cpumodel, cpustepping)

The *topper.getTestFromCpu(cpunumber = "", cpuvendor = "", cpufamily = "", cpumodel = "", cpustepping = "")* method returns all the Tests that match the given CPU. The return format is a list with the Test names (tests unique identifiers).

3.1.17 getPciFromTest(test)

The *topper.getPciFromTest(test = "")* method returns all the PCI devices that match the given Test unique identifier, in a list. Each element of the list is a tuple with five elements: *PCICLASS, VID, DID, SVID* and *SDID*.

3.1.18 getUsbFromTest(test)

The *topper.getUsbFromTest(test = "")* method returns all the USB devices that match the given Test unique identifier, in a list. Each element of the list is a tuple with two elements: *VID* and *DID*.

3.1.19 getSoftwareFromTest(test)

The *topper.getSoftwareFromTest(test = "")* method returns all the Software packages that match the given Test unique identifier, in a list. Each element of the list is a tuple with two elements: *SWNAME* and *SWVERSION*.

3.1.20 getDistroFromTest(test)

The *topper.getDistroFromTest(test = "")* method returns all the Distributions that match the given Test unique identifier, in a list. Each element of the list is a tuple with two elements: *DNAME* and *DVERSION*.

3.1.21 getKModuleFromTest(test)

The *topper.getKModuleFromTest(test = "")* method returns all the Kernel Modules that match the given Test unique identifier, in a list. Each element of the list contains a *KERNELMODULE* element.

3.1.22 getIdeFromTest(test)

The *topper.getIdeFroMTest(test = "")* method returns all the Ide Devices that match the given Test unique identifier, in a list. Each element of the list contains a tuple with two elements: *IDEDEVICE* and *IDEMODEL*.

3.1.23 getFeatureFromTest(test)

The *topper.getFeatureFromTest(test = "")* method returns all the Features that match the given Test unique identifier, in a list. Each element of the list contains a *FEATURE* element.

3.1.24 getCpuFromTest(test)

The *topper.getCpuFromTest(test = "")* method returns all the CPUs that match the given Test unique identifier, in a list. Each element of the list is a tuple with five elements: *CPUNUMBER, CPUVENDOR, CPUFAMILY, CPUMODEL* and *CPUSTEPPING*.

3.1.25 getAllFromTest(test)

The *topper.getAllFromTest(test = "")* method returns every data the Test has, in a Dictionary. The keys of the dictionary are:

pcidevice has a list with tuples with the PCI devices, as returned by *getPciFromTest(test)*.

usbdevice has a list with tuples with the USB devices, as returned by *getUsbFromTest(test)*.

software has a list with tuples with the Software packages, as returned by *getSoftwareFromTest(test)*.

distribution has a list with tuples with the Distributions, as returned by *getDistroFromTest(test)*.

kmodule has a list with *KERNELMODULE* elements, as returned by *getKModuleFromTest(test)*.

ide has a list with tuples with the Ide devices, as returned by *getIdeFromTest(test)*.

feature has a list with *FEATURE* elements, as returned by *getFeatureFromTest(test)*.

cpu has a list with tuples with the CPU elements, as returned by *getCpuFromTest(test)*.

3.2 Free Queries

Topper also has methods that allows to make free queries to the Prolog engine, either from Facts or from Rules.

3.2.1 From Data Structure

This method is *topper.query(args)*, being *args* a list with the facts to be queried with the *AND* operator. Each type of facts is explained in the *Data Structure* Section. The facts not mentioned are treated as *Don't Care* variables.

For example:

```
import topper
topper = Topper()
topper.reloadFacts()
args = [
       "pcidevice('0x1180','0x0552','0x0000','0x0000')",
       "feature('mouse')",
       "usbdevice('0x045e','0x0040')",
       "software('gcc','4:3.3.5-1')",
       "software('linux','2.6.7')",
       "distribution('debian','gnulinex2004r1')",
       "kmodule('snd_pcm_oss')",
       "ide('hdc','dv-w28e')",
       "feature('keyboard')"
       ]
topper.query(args)
```

This will return a list with the matching *TESTS* for the given facts.

3.2.2 From Facts

To make direct questions in the form of Facts, the *pylogquery.run(rules)* method is required, being *rules* a chunk of rules to be queried to the Prolog engine directly. *pylogquery* will run each rule inside of a *"run(TEST):-"* program. Each type of rule is explained in the *Data Processing* Section. For example:

```
import pylogquery
rules = """
has_tested(TEST),
has_pci_device(TEST, pcidevice('vid','did','svid','sdid')),
has_usb_device(TEST, usbdevice('vid','did')),
has_feature(TEST, feature('feature'),
has_software(TEST, software('software'),
is_distro(TEST, distribution('dname','dversion'),
has_kmodule(TEST, kmodule('kernelmodule'),
has_ide_device(TEST, ide('idedevice','idemodel').
"""
pylogquery.run(rules)
```

This will return a lists with the matching *TESTS*. Right now, only *run()* is understood by *pylogquery*.

4 Topper uses

Topper has several possible uses. Please refer to the *Proof of Concepts* Appendix to see one way to gather, feed and consult with Topper from a python+GTK application. Another possible uses are from WebApps, or directly from HDT (Hardware Detection Tools).

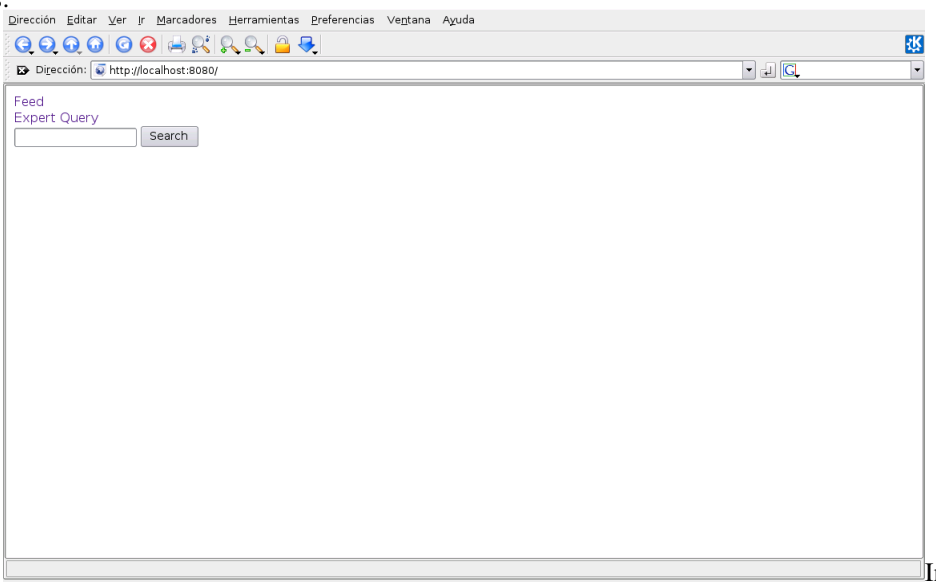
4.1 Cherrytopper

Disclaimer: Cherrytopper is a work in progress. All features and screenshots here are changing periodically.

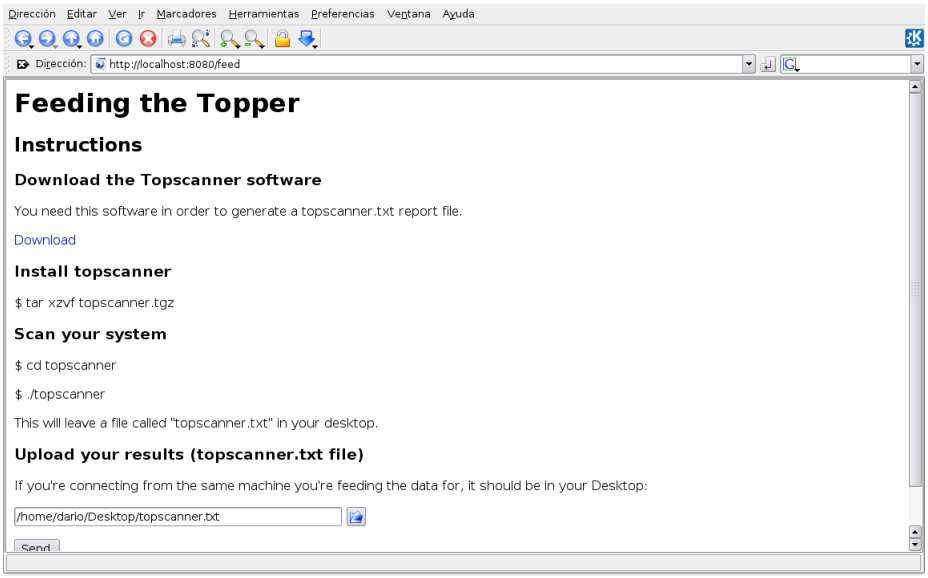
Cherrytopper is a Web Interface to the *Topscanner* and *Topper* libraries. It has between its main goals:

- Provide a simple interface to *Topscanner*, asking feature questions based on the hardware found by the user.
- Provide a Query mechanism to access the *Topper* libraries and hardware database.
- Provide a quick search engine to look thru the Tests database, and look for Topper's information regarding those tests.

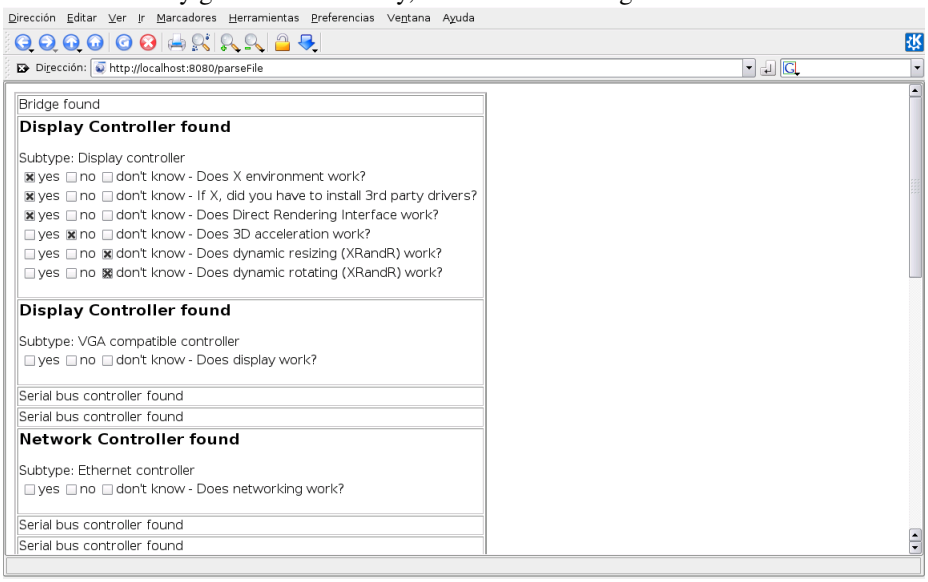
The main screen of Cherrytopper, including the above functionalities, could look like this:



In the above screen, three basic features are displayed. Simple enough, the options are "feeding", "expert query" or a simple search field. Pre-defined queries (like "sound card queries") could be defined (they would be just a simplification of the expert query).

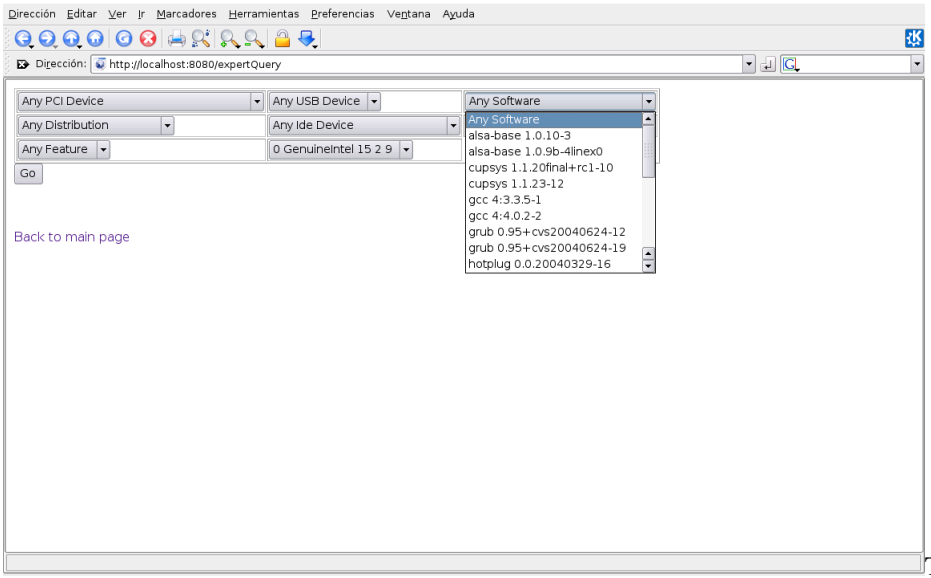


The above image shows the feed screen. Pretty rough right now. The user downloads the scanner, and uploads the results, which are processed on-line and a questionnaire is automatically generated on the fly, as shown in next figure:

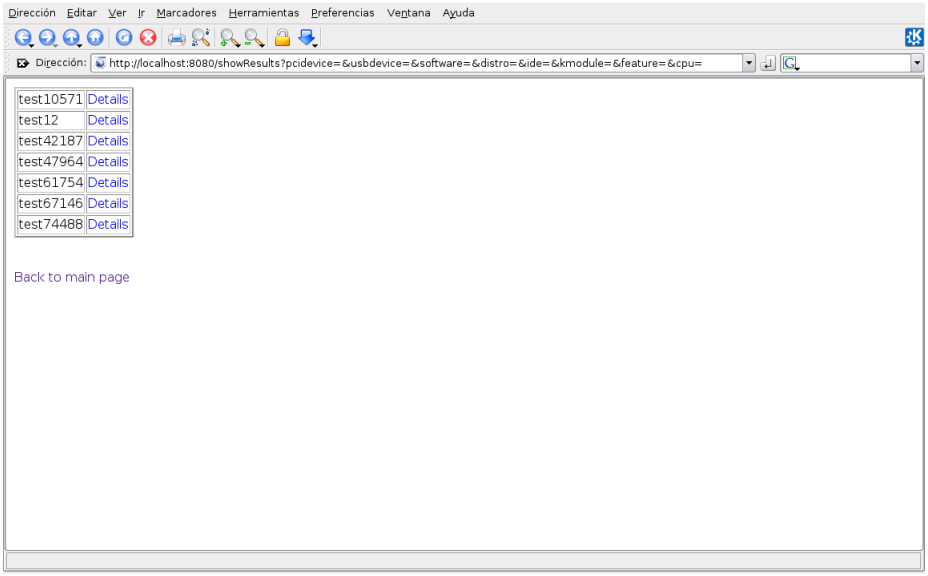


After the user completes the questionnaire, the information is queued up for quality Testing, and then it's merged into Topper's database.

Next figure show the Expert Query menu, where the user can select any kind of constrains to narrow the search. In the example, the user don't care about PCI devices, Distribution, etc., but wants information about successful tests on Intel 15-2-9 processors, with some specific software version:



The results of the Query returns a set of Tests (at least in Expert Mode). This are the tests that succesfully satisfies the constrains entered by the user. Each test can be seen inside to get further details, like all the testing environment, best knowm method document, user's comments, etc.



4.2 Topper from WebApps

Topper can be used from Web Applications to offer technical support to end users. For example, a rule like:

```
from topper import *
topper = Topper()
topper.reloadFacts()
args = [
       "feature('sound')",
       "distribution('Debian','3.1')"
```

```
        ]
topper.query(args)
```

Should answer the hardware and software combinations that allows to have sound support in the given distribution.

Since the rules will depend of the scope of the application, they won' t be detailed here. Some of them can be already predefined in Topper, while the others in the application itself.

4.3 Topper from Hardware Detection Tools

Most of the Hardware Detection Tools (HDT) available in Linux distributions consist of a static hardware database included with the distribution, and some software loading kernel modules or configuring the system according the the hardware found, in a per-component basis.

Debian's HDT, *discover*, for example, delivers static hardware information in XML form in the *discover-data* package, and uses a shell script, *discover-modprobe*, to load kernel modules according to the pair of Vendor ID/Product ID hardware found in the system. Since *discover* is only a way of retrieving XML data, it can be extended, for example with a *discover-apt* [3] application. Even while *discover* provides a way to extend the hardware database[4], it doesn't have a track of the behaviour of different hardware combinations. This results in the necessity of human interaction to provide sensible defaults for certain hardware[5], like blacklisting.

Concluding, since *discover* knows how to install software, how to load kernel modules, and what hardware is in the system, it can perfectly ask Topper for clever information regarding this subjects.

[3] *discover-apt* is currently deployed and used in the *juegaLinEx* distribution, with the *discover-linex-tools* package, in order to install non-Debian software packages, like graphic drivers.

[4] For example, the *linux-sound-base* package provides an extension of the sound card database.

[5] For example with the blacklisting of the *i8xx_tco* kernel module, which caused hangups in some platforms.

5 Scope

Even when Topper in itself is not a solution for the complex dependencies between software and hardware components, it can help in diverse areas, like User Support, Software Installation and Software Configuration. It can even help in Software development, helping the developer to identify the conflictive pieces of software that deny a proper system behaviour.

Topper does not provide a tool or a process. Topper provides *Information* given a set of *Data.*

5.1 Bottom Scope

At the bottom, the scope of Topper begins with the processing of *Facts* previously gathered by some other tool, or with *Topscanner*. Refer to the *Data Structure* or *Data Gathering* Sections for more information on this.

If not using *Topscanner,* this Facts must be introduced in plain text format to the *parser* class:

```
import parser
factslists = parser.Parser()
```

Then, we can pass the plain text file to the *Parser* directly:

```
file = "topscanner.txt"
factslists.addTest(file)
```

After this, the Parser will generate a new Unique Test Identifier and add the Test to the Database.

It's *not* Topper's duty to collect this data. However, see the *Proofs of Concept* Appendix for examples, or *Data Gathering* Section about *Topscanner.*

5.2 Upper Scope

The Upper Scope of Topper is defined by the delivery of the requested information, via the *Predefined Queries* or the *Free Queries*, as shown in the *Interaction with Topper* Section above.

It's *not* Topper's duty to provide and end user tool to consult the Data, even when it can be done in several ways. See the *Proof of Concepts* Appendix for examples.

6 Conclusion

Given the complexity of today's software and hardware dependencies in the Free Software environment, and the complexity of the actors involved in development, to offer and use an active hardware and software dependency tool is quite difficult. For this reason, a passive approach is used, in the hope of a better understanding of the main causes of today's situation.

7 Machine Learning

One of the possibilities in this subject is to implement a Machine Learning mechanism, to let the system identify which combinations of hardware, software and features would actually work, without the need of human interaction.

Refer to Appendix B to see an example.

8 Appendix A - The “Topper” Name

Topper is the name of one of the minor characters in the Dilbert strip by Scott Adams, who made his first appearance in 2001. He’s well known for always having a comment that “tops” anything that anyone always made. Since the Topper Project is meant to deliver an expert system aimed to answer questions regarding hardware and software dependecies issues, it seemed to be the right name.

Topper can be seen in January 2001 Dilbert strips, appearing again in August 2005.

9 Appendix B - Data Mining with Machine Learning

In this section we will see an example using the Open Source Machine Learning software Weka[6]. For this purpose, a false and small data set is provided for experiment purposes, as follows:

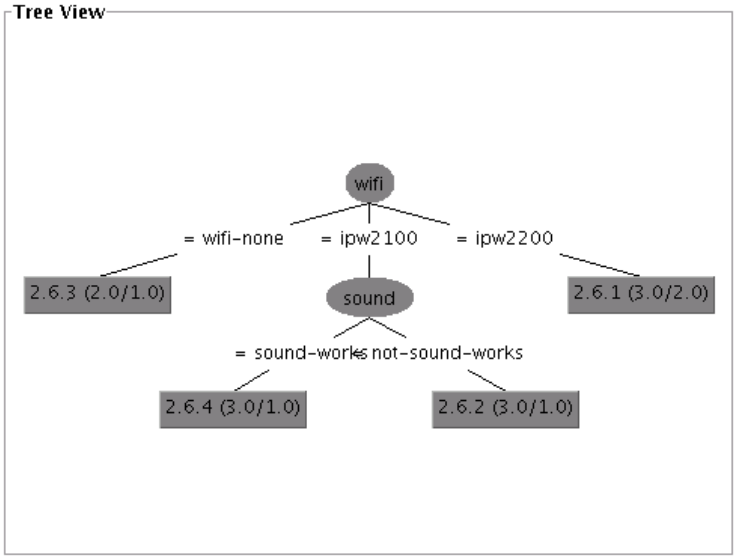
```
%
@relation topper
@attribute kernel { 2.6.1, 2.6.2, 2.6.3, 2.6.4, 2.6.5, 2.6.6, 2.6.7, 2.6.8, 2.6.9,
  2.6.10, 2.6.11, 2.6.12}
@attribute xorg { 6.8, 6.9, 7 }
@attribute suspend-to-ram { suspends-to-ram, not-suspends-to-ram }
@attribute sound { sound-works, not-sound-works }
@attribute video { video-works, not-video-works }
@attribute wifi { wifi-none, ipw2100, ipw2200 }
@attribute wireless-capabilities { wifi-works, not-wifi-works }
@attribute host-bridge { 0x3580, 0x3581 }
@attribute uhci-controller { 0x24c2, 0x24c4 }
@data
2.6.1,6.9,suspends-to-ram,not-sound-works,video-works, ipw2200,not-wifi-works, 0x3580, 0x24c4
2.6.4,6.8,not-suspends-to-ram,sound-works,video-works, ipw2100,wifi-works, 0x3581, 0x24c4
2.6.3,6.9,suspends-to-ram,sound-works,not-video-works, wifi-none,not-wifi-works, 0x3581, 0x24c4
2.6.2,7,not-suspends-to-ram,not-sound-works,video-works, ipw2100,not-wifi-works, 0x3580, 0x24c2
2.6.6,7,suspends-to-ram,sound-works,not-video-works, ipw2200,not-wifi-works, 0x3581, 0x24c2
2.6.2,6.8,not-suspends-to-ram,not-sound-works,video-works, ipw2100,wifi-works, 0x3580, 0x24c2
2.6.4,6.8,suspends-to-ram,not-sound-works,not-video-works, wifi-none,not-wifi-works, 0x3580, 0x24c2
2.6.4,6.9,not-suspends-to-ram,sound-works,not-video-works, ipw2100,wifi-works, 0x3581, 0x24c4
2.6.6,6.9,not-suspends-to-ram,sound-works,video-works, ipw2100,wifi-works, 0x3581, 0x24c4
2.6.6,7,suspends-to-ram,not-sound-works,video-works, ipw2100,not-wifi-works, 0x3580, 0x24c2
2.6.9,7,not-suspends-to-ram,sound-works,video-works, ipw2200,not-wifi-works, 0x3580, 0x24c2
%
%
%
```

As seen in the example, there is information regarding software as kernel versions and Xorg versions, regarding hardware as Wi-Fi, USB controllers and Host Bridge chipsets, and features like “sound works” or “suspend to ram works”.

Once the data is collected, the system can analyze it using diverse tree algorithms. In this example, the j43[7] tree is used to analyze the relationship between Wi-Fi chipsets, kernel versions and sound capabilities from the given tests.

[6] http://www.cs.waikato.ac.nz/ml/weka/

[7] Chunsheng Li, Li Liu, and Qingfeng Song, A Practical Framework for Agent-based Hybrid Intelligent Systems, Asian Journal of Information Technology, 3 (2): 107-114, 2004, Grace Publications Network.



In this example, looks like kernel version 2.6.2 makes sound to do not work using a ipw2100 wi-fi chipset.

Concluding, the data gathered by the Actors can be used to help determine strange beheviours when certain hardware/software circunstances are presented.

10 Appendix C - Proof of Concepts

In http://rapisardi.org/download/topper/ there're different Proof of Concepts, in video.

http://rapisardi.org/download/topper/discovery.htm shows a possible way to gather Data.

http://rapisardi.org/download/topper/feed.htm shows how to introduce Data into Topper.

http://rapisardi.org/download/topper/consult.htm shows a possible way to gather information from Topper.

11 Changelog

11/07/2005:

- Initial Version.

11/20/2005:

- Added data structure.
- More details about data gathering.
- More details in the Scope.
- Modified way of treating collected data.

11/21/2005:

- Fixed some typos.

11/27/2005:

- Replaced “module” for “kmodule” in fact name to avoid conflicts.

12/06/2005:

- Updated data structure example.
- Added Topper’s predefined methods.
- Updated collected data example.
- Merged previous “Topper” subsection into “Interaction with Topper” section.
- Added “Free Queries” subsections.
- “Facts” Section removed.
- Added “Upper Scope” and “Bottom Scope”.
- Fixed fact name “distro” for “distribution”, as implemented.
- Improved “Data Gathering” Section.

01/30/2006:

- Added CPU information to the data structure.
- Updated data structure chart.
- Move Data Cleansing to 2.4.
- Data Processing now 2.5.
- Added Data Cleansing subsubsection.
- Added Topscanner data gathering process.

01/31/2006:

- Changed orientation of Data Structure chart.

02/02/2006:

- Changed example of Data Cleansing.
- Be more precise with “highest quality of data”.

02/06/2006:

- Add CPU info support.
- Add CPU related methods.

02/12/2006:

- Add Class info support to PCI data.

02/27/2006:

- Order feature list alphabetically.
- New features: *pcmcia, cardbus, serial-multiport, display-x, display-dri, display-3party, display-3d, display-xrandr-resize, display-xrandr-rotate, digitizer-pen, gameport, scsi, ide, ide-dma, ide-pata, ide-sata, floppy, raid, mass-storage, multimedia-video, multimedia-audio, multimedia-audio-hda, multimedia-audio-mic, multimedia-audio-midi, networking-eth, networking-atm, networking-tr, networking-isdn, networking-fddi, wifi-firmware, wifi-80211ab, wifi-80211g, wifi-managed, wifi-adhoc, wifi-master, wifi-monitor, wifi-wep, usb, usb2, irda.*
- Change feature names: *display* instead of *video*, *wifi* instead of *wi-fi, multimedia-audio** instead of *audio.*
- Remove *accel*, *audio*, *network* and *webcam* features (replaced by other ones).
- Better formatting of this Changelog.
- Add *Cherrytopper* to the Mode of Operation diagram.
- Fix Subsection types in Section 4, *Topper Uses.*
- Add Cherrytopper subsection in *Topper Uses.*