Semantic Linux: a fertile ground for the Semantic Web

Guillaume Barreau and Debora Abdalla

Universidade Federal da Bahia, Salvador, Brazil,
gui@im.ufba.br

Abstract. This paper discusses the application of ontology-based knowledge engineering to the domain of Linux. We show that both the Linux community and the knowledge domain have some very interesting properties which make it a very good testbed for many of the ideas developped in the field of knowledge management. Various applications are discussed which would all benefit from a comprehensive ontology of the domain.

1 Introduction

The emergence of the Linux operating system was a defining trend for the IT world in the 90s. During that decade, Linux was unexpectedly transmuted from an experimental project involving a few hard-core computer scientists to an operating system on which many depend for their daily operation. As an example Google, one of the pillars of the Internet today, runs its estimated 100.000 servers on Linux.

The amount of information available on Linux is substantial. Yet, the complete distributed way in which this information is generated means that this corpus varies in quality, language, style and target audience. It is therefore surprising that the question of Knowledge Management (KM), despite the attention it gets in other circles, has not yet been raised within the Linux community. Are the issues facing the Linux world in this respect any different from those facing other knowledge communities? What practical problems could KM tools and techniques solve for the Linux community? Are the standards emerging from the Semantic Web, RDF and OWL, of any practical relevance to the Linux community?

We are starting a 3 year project whose aim is to investigate those issues. Our aim is to build some prototypes that will show the benefits of a long-term investment in KM in the hope of influencing the Linux community towards some of the practices recommended for the Semantic Web. This article sets the ground for this research by examining in some detail the way knowledge is produced and consumed within the Linux universe. We then go on to discuss some work in progress on an ontology-based application in the area of package management. Finally, we move on to discuss some more ambitious applications which would be feasible if a comprehensive Knowledge Base about Linux was developed.

2 The Linux world

Although Linux has become a well established brand, many people are still unclear about what exactly lies behind the name. And for a good reason, since the answer is quite complex. The aim of this section is to clarify the kind of channels which exist for sharing Linux-related knowledge.

The Linux community is most probably the largest virtual community in existence today. And being both large and very active, it generates large amounts of information on a continual basis. This fact, we hope to show here, makes this community a prime target for the Semantic Web.

The success of Linux as a virtual community is due to the values it embodies: technical excellence, enthusiasm for technological progress and, above all, openness. These values, which are the defining values of the hacker ethic [1], are the same which were at the origins of the Internet. This is why Linux has acted as a focal point for many of the most experienced people on the Internet. Today, it keeps attracting many enthusiastic and curious minds from around the world. But while it is still innovating at a rapid rate, Linux is now taking on board a different community of users who is mostly interested in simplicity of use. Many governments and public institutions are now using it for their daily activities and they, of course, are coming with a different set of expectations than those of the early hackers.

Another defining element of this community, is that it is highly decentralised. Unlike the Microsoft Windows or Macintosh worlds, there is no central entity showing the way and setting the agenda. The fact that Linux could make it so far on such a distributed and unorthodox model is a lesson that will require more scholar analysis. But what has worked in one set of circumstances might not adapt gracefully to changing times. The mechanisms that enabled successful knowledge sharing between technically-minded pioneers might not serve so well the communication needs of a large base of users demanding simplicity. Mechanisms for ensuring that quality documentation, for instance, will have to be devised. It seems that Knowledge Management will be crucial to Linux's ability to become a product usable by the public at large.

2.1 The Kernel

Strictly speaking, Linux is the name of the kernel of a Unix-like operating system whose development was started by Linus Torvalds in 1991. Although the idea of collaborative work on open and freely available software had been formalised by Richard Stallman in the early 1980's, Linux was the first software project of that kind to capitalise on the potential of the Internet to let people across the world work together without having ever seen each other. It naturally became a reference for that kind of distributed development and has been emulated since by thousands of amateur programmers working on other software projects.

A project of that complexity requires a lot of shared knowledge between the participants. But because Linux had the explicit objective of emulating the Unix operating system, the potential hurdle of agreeing on a design and a vocabulary was removed. Most things could be borrowed from Unix: the philosophy, the design, the vocabulary, the documentation format (man pages), etc. This was an important factor in the success of the project since it allowed people familiar with the Unix terminology to gather effectively around a commonly understood and agreed set of objectives.

Kernel development today is still the realm of a relatively small number of very talented and dedicated people distributed around the world. However, every release nowadays sets a chain reaction that propagates to every corner of the large virtual community which depends on this kernel.

2.2 The applications

The Linux kernel by itself doesn't do much. It is a foundation on which others can build useful applications for users. When Linux came out, most of these applications were those developed by the GNU project headed by Richard Stallman; Nowadays there is an extremely large number of groups developing software for Linux. Some of those groups are federated around a common philosophy or common design choices such as KDE and Gnome. Others have no affiliation.

The culture of sharing that prevails in the Linux community obviously favours code reuse. One area in which this reuse is most easily observed is that of code libraries. Rather than re-implementing over and over the same basic functionalities, applications tend to rely on specialised libraries. In addition to the advantage of not duplicating effort, this ensures that improvements in the library can quickly be passed on to those applications that rely on it. The coupling between a library and the applications that use it is guaranteed to evolve gracefully through a software contract in the form of an Application Program Interface (API).

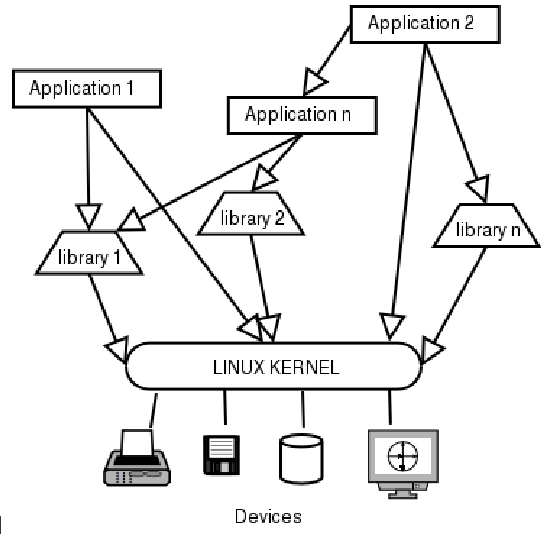
In some cases, an application might also depend on other applications as well as on libraries. For instance, applications designed to interface with the user through a browser will depend on having an http server to work with. Again, rather than rewriting code, application developers will often rely on some third-party software which provides the functionality in a reliable manner.

The reuse of libraries and other functions is not specific to Linux. However, due to the philosophy of sharing that underlies it, the web of dependencies between independently developed units of code is denser. Figure 1 shows the overall dependencies between the kernel, the libraries and the applications.

This Figure, however, is not representative in terms of scale. The number of applications and libraries available for Linux today is in the order of tens of thousands. Many of these have overlapping or even conflicting functionalities. So deciding what needs to be installed on a system at a given time is not a trivial task.

2.3 Linux distributions

The process of getting a system made of so many components to work on a machine whose physical properties are not known in advance is complex and can



[p]

Fig. 1. The functional dependencies between the various elements that make up a Linux system

be approached in a number of different ways. This is where Linux distributions fit in. They offer a practical solution to the problem of assembling (in software terms) a working system from that myriad of distributed components. They differ from each other in the following aspects:

- the tools they offer to guide the user through the installation of the system (kernel and applications). Some will simplify the process by assuming default values for many parameters, while others will give the user more control through a question-driven process.
- the set of applications and libraries that are distributed among all the available ones. When an application is not part of a distribution, the user of that distribution still has the option of trying to incorporate it without the support of the distribution; this is however usually a more complicated procedure which might involve compiling the source code.
- the tools they offer to maintain and configure the system. One important part of this process is the addition, removal and upgrade of libraries and applications in a way that reflects the user's changing needs and without duplicating shared components such as libraries.
- the way in which the files that make up an application are organised on the system. For instance, configuration files might go in one directory in one distribution and in a different one on another distribution.

The last two aspects will be described in more details in Section 3.

A distribution can be seen as a a portal to the Linux world. For many users, this will be their only point of contact with the Linux universe. As such it partitions the Linux community into sub-communities with boundaries whose permeability will depend on the user experience. Indeed, advanced users can normally translate information that has been phrased in a way that is specific to a distribution to their own distribution.

2.4 The overall picture

In addition to the kernel developers, applications developers and distributions which have already been described, there are other important players within the Linux knowledge economy. Entities such as Sourceforge [1] and Savannah [2] have been crucial in lowering the barriers to distributed software development and facilitating for users and distributions the process of locating useful applications. Both these sites support the development of open-source software by providing reliable web-based collaboration software. The Linux Documentation Project [3] fulfils the important role of trying to provide a unique point of access to documentation on a wide range of topics. Yet another category includes sites who provide news on Linux and related topics such as Linux Weekly News [4] and Slashdot [5].

We have thus a very rich and diverse community with participants varying in commitment, experience, areas and levels of expertise, motivation (financial, philanthropic, educational), and language to list a few. Given the current trend, this diversity is likely to increase even more in the future.

The option now exists for someone to be a Linux user without having to navigate this information ocean. They should be able to get hold of a disk, follow some simple instructions, answer a few basic questions, and end up with a system running the universal applications (browser, word processor, email client, etc.). Excluding that category, everything indicates that a large number of people including kernel and application developers, technical writers, usability experts will need to manage their way through an ever-increasing pool of information in constant flux.

People in the Linux community will need tools to help them manage the knowledge they produce more effectively. One obvious use case is the retrieval of relevant information at the right time at a level of detail that is suited to the seeker. This kind of problem is what drives much of the Semantic Web effort. Thus it seems that an ontology, i.e. an explicit shared conceptualisation of the Linux universe [2], could greatly help the organisation and reuse of the knowledge thus produced.

In its entirety, Linux is quite a vast domain of knowledge to formalise. But there is a very favourable factor. Given that most of this knowledge is about software, a domain in which there is not much for subjective interpretation,

[1] www.sourceforge.org
[2] www.savannah.org
[3] www.tldp.org
[4] www.lwn.net
[5] www.slashdot.org

it ought to be relatively easy to formalise compared to other areas of human knowledge.

The following section will give a concrete example of how an ontology of a sub-domain of Linux could improve the ability of the community to handle complexity. The section after will provide a broader view of the kinds of applications which would become possible if a full ontology of Linux was developed.

3 An Ontology-based solution to Package management in Linux

3.1 The problem of package management

For the purpose of installing, removing or upgrading software, distributions wrap sets of files which are functionally related into units called packages. Typically, a package is a wrapper around an application or a library but it can also be the documentation files on a given topic.

Applications and libraries for Linux are not normally developed for a specific distribution. So they must remain quite neutral on a number of issues on which distributions differ. As a result, an extra effort is needed to bridge the gap between the distribution-neutral version of the application and the distribution-specific packaged version of that same application.

Because we are dealing with open-source software, one possibility is to distribute an application in its source form. The source code can then be compiled on the target system following a recipe that is summarised in a Makefile. This type of package management has been used by the Gentoo distribution. The drawback is that the compilation process, which could be executed only once, has to be repeated on every single machine where the software is installed.

By far the most common approach is to package a compiled version of the application along with all the necessary auxiliary files. The two most common packaging formats of this kind are those originally developed by the Red Hat and the Debian distribution. The problems that these methods are addressing are the same, mainly:

– supporting the intelligent handling of *dependencies* between packages: if package X only works in the *presence* of package Y, then package Y must be installed prior to installing X
– supporting the intelligent handling of *conflicts* between packages: if package X only works in the *absence* of package Y, then package Y must be removed prior to installing X
– each distribution has a particular way of organising the files that make up an application according to their function. For instance, a distribution might want to keep all the configuration files for package X under the directory */etc/X*. Files must therefore be identified in order to be copied at the right place and proper references to them must be passed to all other files which need to know where these files are kept.
– supporting the discovery of packages by users: users must be able to discover the names of packages when they need them.

3.2 The Debian way of packaging

We will present here the basics of the Debian package management meta-data system as they are representative of the current way of addressing the problem. A typical package for an application will contain all the files needed for the application to be installed. The documentation files are also normally included as part of the package. In addition, every package contains a file which contains some basic meta-data about the contents of the package. This file for the Apache web server has the following contents:

```
Package: apache
Version: 1.3.26-0woody3
Section: web
Priority: optional
Architecture: i386
Depends: libc6 (>= 2.2.4-4), libdb2 (>= 2:2.7.7.0-7),
   libexpat1 (>= 1.95.2-6), mime-support,
   apache-common (>= 1.3.26-0), apache-common (<< 1.3.27-0),
   perl5 | perl, logrotate(>= 3.5.4-1), dpkg (>> 1.9.0)
Suggests: apache-doc
Conflicts: apache-modules,jserv (<= 1.1-3)
Replaces: apache-modules
Installed-Size: 752
Maintainer: Matthew Wilcox <willy@debian.org>
Description: Versatile, high-performance HTTP server
 The most popular server in the world, ...
```

Many of the elements above are self-explanatory. Packages in Debian are organised into a number of discrete sections; Apache belongs to the web section. We see that a number of dependencies are listed. For instance, Apache needs libc6 in a version at least as recent as 2.2.4-4. There are also a number of packages conflicting with this one. The maintainer is the name of the person who is responsible for packaging this application for Debian and a valid email address must be provided for that person. The description is a paragraph explaining the purpose of this application which has been truncated in this example.

In order to facilitate installation and removal of packages, Debian has developed tools which can parse such package files and extract from them all the information needed to automate this process.

3.3 An ontology-based solution

Access to a large pool of frequently updated and constantly improving software is one of the attractive features of the Linux operating system. It is therefore important to ensure that the process of finding, adding, and removing this software is as easy as possible. Most users, even beginners, have software needs that change with time.

Currently, every distribution addresses the problem of package management independently from the others. Distributions and their collaborators end up putting a lot of effort into the preparation of appropriate package files from the files and instructions produced by application developers. In the Debian distribution, the maintainer of the package is responsible for this task.

But the meta-data which is needed for package management is the same for every distribution. We are currently working on a representation of this meta-data which would be compliant with the W3C standards OWL and RDF [3, 4]. We are also writing tools which can interpret this OWL/RDF file and turn it automatically into a debian or red hat package. The meta-data shown in the apache example is not enough in order to be able to do that. We will need to make explicit some additional meta-data which is currently implicit. An example is the list of files which are used as configuration files. An even more appropriate way of handling configuration, would be for applications to specify, in RDF, the full list of configuration parameters as well as their default values. This way, the exact way in which these parameters are represented in a file would stop to matter.

Figure 2 shows some of the elements of the ontology we are creating to accommodate this meta-data.

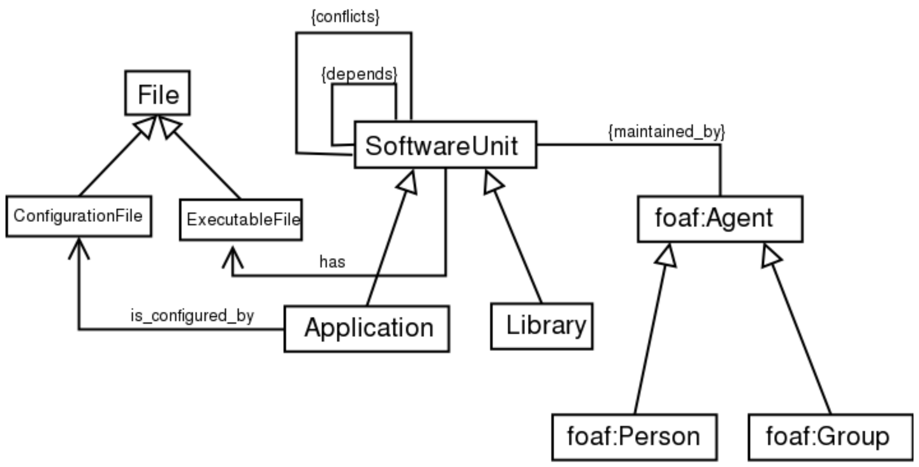


Fig. 2. The basic elements of a package management ontology. Notice the use of the FOAF vocabulary.

3.4 Advantages of the approach

Once we have a prototype in place, we intend to advertise this approach in the hope of convincing application and library developers to make available an RDF file that would contain the application meta-data in a form compatible with this ontology. This file would contain all there is to know about that piece of software in a consistent form which can be manipulated by RDF/OWL compliant tools. Distributions should be able to specify their way of dealing with packages in terms of the vocabulary defined by this ontology. As a result, the differences between distributions would themselves be more easily scrutable.

We are not proposing to generate additional meta-data about applications. All that we are proposing to represent is information that could be discovered by someone who would carefully explore the web site of each application. We are proposing to make this meta-data explicit and keep it in a standard form. The advantages are many. One of them is that RDF has already defined some vocabularies which could be interesting for our purposes. When talking about people (such as the developers of a particular application for example), we could make use of the FOAF [5] vocabulary whose purpose is to express relationships between people and the projects they are working on.

Another advantage of the OWL/RDF format is that it can be queried quite easily [6] and supports inferencing. Currently, the classification schemes of packages is quite basic and doesn't let a user locate the application he needs easily. The OWL formalism would of course provide a solid foundation for classifying packages in a semantically sound way. With the right facets in place, we could expect to be formulate very specific queries such as: "Which are the applications which can edit gantt charts and save them in an Xml format?".

4 Other possible applications

4.1 Improved Information Search

One of the most obvious applications of more formal semantics is an improved retrieval ability. The current way of searching for Linux information is limited especially for the novice. There are at least two way in which ontology could improve the search process.

One way is to use index documents based on such ontologies. This will only become practical when it is supported by tools which let the author easily specify the topics of her document by reference to existing ontologies. It will also become a practical solution when the process can be fully automated through the use of advanced Natural Language Processing techniques as suggested by Craven et al.[7].

A more basic but yet useful approach is to use an ontology to help the novice find good keywords to represent their query in a traditional way. The user would have a first attempt at formulating their query in terms of keywords. An ontology-based system would then try, through some kind of simple dialog with the user, to map those keywords into a query formulated with the controlled vocabulary of the ontology. The result of this modelling of the user's query would then be translated back into a search string which could be expected to return quality results when submitted to a traditional search engine.

4.2 Self-healing Computers

Knowledge based systems have been quite successfully applied in the area of medicine [8, 9]. It has been possible with such systems to formalise the knowledge of a human expert and reproduce the reasoning that takes place when she finds the most likely causes of some symptoms and recommends a possible cure.

Linux systems, like any kind of system, sometimes stop functionning properly due to software misconfiguration. The reasons are varied: accidental removal of important files, network intrusion, file system corruption caused by power failures, buggy applications ... When some serious problem happens with a system, beginners will need access to an expert or resort to the radical solution of replacing large parts of the system with an "out of the box" configuration. The latter is very costly in time and sometimes in data. An expert, on the other hand, will try to formulate more hypothesis regarding the cause of the problem until he can find one which successfully explains the problem and all its symptoms. The analogy with the medical field should be obvious: the computer is ill and the expert will find the causes of the "disease".

So, taking advantage of the experience gained in the area of medical KBS, we should be able to build a system with enough formalized knowledge of Linux to detect abnormal system behaviour and recommend or even perform the required changes in the system's configuration. The kind of knowledge base that would support such a system would be identical to the one needed for intelligent search. This is after all the fundamental promise of ontologies: to represent knowledge in a way that is independent of the intended use [10].

In medicine, getting data that can support the decision process is often difficult and costly since it involves performing some more or less invasive clinical analysis. With computers, in contrast, there is often an excess of data in the form of log files. Additionally, testing a potential cure is often as simple as editing a few files and doesn't have the risks associated when dealing with human life. As a result, a KBS for the diagnosis of computer faults should be easier to implement than a medical one.

In another respect, however, medicine is simpler than operating systems. The underlying laws of the human body are more or less fixed and don't change with time. Only our understanding of them evolves slowly. But a universal computer is by definition a machine where every behaviour can be changed by software. In the case of a Linux system, tens of thousands of programmers across the world are constantly working to change those laws. This is normally for the better, but the potential for unforeseen side effects is large. So every upgrade of the software on a machine would potentially require a change to the Linux knowledge base.

5 Conclusion

The Linux community is the largest virtual community today. As a result, it publishes on the web a lot of information in very diverse forms: mailing lists archives, technical documentation, news, bug reports, ... We have tried to show in this paper that this Linux-related cross-section of the web has some very interesting features from the point of view of the semantic web effort which are not easily found in any other domain-related cross-section of the web.

First in terms of size and diversity. It is large enough that it has most of the properties and problems of the web as a whole. For instance, the limitations of search based on character patterns are certainly heavily felt when searching the Web for information about Linux. Yet, this cross-section is orders of magnitude smaller than the web as a whole which makes it a interesting intermediary step for validating the ideas of the Semantic Web.

Secondly, knowledge about Linux is knowledge about software which is formal by design. Hence formalising that kind of knowledge ought to be easier than in most other domains of knowledge.

Thirdly, people who make up the Linux community are likely to be more sympathetic than others to the aims of the semantic web and the extra rigour it demands when publishing content. Being computer literate, they can understand better than anyone what is needed to help computers make sense of content. It is unclear whether enough of them can be convinced of the worthiness of the project to reach the critical mass that is needed for the project to take off. But if that category of people cannot be convinced of the worthiness of this experiment, we doubt that any other knowledge community can be.

Finally, as we have sketched along this article, it is quite easy to imagine many applications that would make use of this formalised Linux knowledge if it was available. Some of them could greatly increase the quality of the operating system. One criticism often heard about the Semantic Web is that there is no "killer" application. That would not be the case with Linux for the additional reason that this community has the ability to build its own tools. So if a comprehensive knowledge base of Linux was available, we could expect a flowering of interesting applications: the Linux hackers could be showing the way for the Semantic Web. Given that the ideals that gave birth to the Internet and the web have found their continuity in the open-source community [11], this should not be such a surprise after all.

References

1. Himanen, P.: The Hacker Ethic. Random House (2001)
2. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5** (1993) 199–220
3. Manola, F., Miller, E.: Rdf primer. http://www.w3.org/RDF/ (2004)
4. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: Owl web ontology language reference. http://www.w3.org/TR/owl-ref/ (2004)
5. Blag, J.: Foaf vocabulary specification. http://www.foaf.org (2004)
6. Broekstra, J., Kampman, A., van Harmalen, F.: Sesame: A generic architecture for storing and querying rdf and rdf schema. In Davies, J., Fensel, D., Harmelen, F.V., eds.: Towards the Semantic Web: ontology-driven knowledge management. John Wiley & Sons (2003)

7. Craven, M., DiPasquo, D., Freitag, D., McCallum, A.K., Mitchell, T.M., Nigam, K., Slattery, S.: Learning to construct knowledge bases from the World Wide Web. Artificial Intelligence **118** (2000) 69–113
8. Heijst, G.V.: The Role of Ontologies in Knowledge Engineering. PhD thesis, Universiteit van Amsterdam (1995)
9. Gangemi, A., Pisanelli, D., Steve, G.: Ontology integration: Experiences with medical terminologies (1998)
10. Sowa, J.: Knowledge Representation. Random House (2001)
11. Lessig, L.: Code and other laws of cyberspace. Basic Books (1999)