On the Benefits for Industrials of Sponsoring Free Software Development

J-P. Rosen

Adalog, 19-21 rue du 8 mai 1945, 94110 ARCUEIL, France; Tel: +33 1 41 24 31 40; email: rosen@adalog.fr

Abstract

Adalog [1] has developed two tools recently, one for an industrial client[1] (AdaSubst/AdaDep), and one for Eurocontrol (AdaControl). Although the programs were custom-made after the requirements of the clients, in both cases, they allowed the tools to be released as free software after they were delivered to them. In this presentation, we describe the clients' needs, the tools that were produced, and more importantly our experience that releasing the tools as free software was indeed beneficial to the clients, to Adalog, and to the community at large.

Keywords: free-software, industrial experience, semantic tools, ASIS.

Introduction

When an industrial company develops a software tool, it usually keeps it for itself. The rationale is simple: if the company pays for the software, it owns the software. Why would a company pay for the benefits of others, by making it freely available?

First it should be noted that, contrary to a common misunderstanding, releasing a tool as free software does *not* mean that the company does not own it anymore: free software is not public domain software. The company holds the copyright, and can do whatever it wants with it, including reusing it in part or in whole for proprietary programs. Making software free *never* diminishes the rights of the owner, including the right of not making new releases free (unlike *users* of the software who must continue to distribute it freely, at least when the software is provided under the terms of the GPL).

However, releasing the tool freely outside the company implies that anybody can use it, including the company's competitors; this may create concerns. On the other hand, this also means that anybody can contribute to it and improve it. Therefore, taking the decision of releasing a program as free software is really a matter of balancing benefits and drawbacks.

In this paper, we describe two experiments where the releasing of paid developments as free software was beneficial to the industrials. We do not claim that this can be done in every case, but we argue that "paying for free software" can be cost effective for certain classes of tools.

[1] who didn't want to be disclosed

1 The case of AdaSubst/AdaDep

1.1 Context

An industrial client had developed over the years several big libraries dealing with its problem domains. Since this effort started long ago, the code and the structure of the libraries were still compatible with Ada83. And, as is often the case when a code has evolved over many years, it came to a point where a major restructuring was needed.

Axlog [2], Adalog's mother company, won the contract for reorganizing this software components base. This implied, among other things, breaking big packages into a hierarchy of child packages, and often changing the names of the provided services. Of course, such changes would break all existing code that used the libraries. Therefore, the contract stipulated that a tool should be provided to migrate code to the new library structure. The initial intent was to provide some kind of ad-hoc Python program to do this.

1.2 Solution

Adalog proposed to develop instead a general tool (AdaSubst), based on ASIS (*Ada Semantic Interface Specification* [3]), which would not be specific to this migration, but could be used for any similar needs. A dictionary file describes, for each entity, its old name and the place where it was declared, and its new mapping, i.e., its new name and the new package where it is now. Typical entries in the dictionary look like this:

```
Old_Pack => New_Pack
Old_Pack.Proc1 => New_Pack.Proc2
Pack1.Func{integer return integer} => New_Func
Big_Pack => Parent, Parent.Child1, Parent.Child2
all Print => Put
```

The first line means that the package "Old_Pack" is now called "New_Pack"; the second line means that the procedure "Proc1" in package "Old_Pack" has been renamed to "Proc2" in package "New_Pack". The third line is an example of dealing with overloaded declarations: only the function "Func" that takes an Integer parameter and returns an Integer value is changed into "New_Func". In the case of the fourth line, a package has been split into a parent package and two child units. The last line means that all procedures named "Print" are now called "Put", irrespectively of where they are located. Note that if a package name changes, it is not necessary to specify the transformation for all its elements, as long as the names are not changed; only changed elements need to be described.

The tool makes all the necessary transformations on the code, taking all Ada rules into account; use clauses are properly modified, overloading is taken into account; when a name changes in a generic, the change is propagated to all uses in all instances, etc. The only case that is not fully automated is for elements declared in a package that has been split (like "Big_Pack" above). "With" and "use" clauses are transformed to name all new packages, but for an element given in prefixed notation, it is not possible to know in which unit it resides know. In this case, the transformation prefixes the name by the various possible packages, separated by '?'. Since this does not compile, it is easy to edit the construct to choose the appropriate package manually. In short, the tool goes far beyond what could be done by text substitution, even with sophisticated pattern matching tools such as those provided by Python.

In addition, the migration itself required a detailed analysis of which elements from all "withed" packages were used. Adalog developed a companion tool (AdaDep) to ease this analysis. It gives, for a given unit, which elements from each withed unit is used and how many times, together with the nature of the element. For example, given:

```
package Pack is
   I : Integer;
   package Internal is
      V : Float;
   end Internal;
end Pack;

with Pack, Text_IO;
use Pack, Text_IO;
procedure Sample is
begin
   I := 1;
   Internal.V := 3.0;
   Put_Line (Integer'Image (I + Integer(Internal.V)));
end Sample;
```

Running AdaDep will produce:

```
SAMPLE (body) =>
=> from ADA.TEXT_IO
   PUT_LINE - A_Procedure_Declaration * 1
=> from PACK
   I - A_Variable_Declaration * 2
=> from PACK.INTERNAL
   V - A_Variable_Declaration * 2
=> from STANDARD
   INTEGER - An_Ordinary_Type_Declaration * 2
```

In agreement with the client, AdaSubst and AdaDep were released as free software. The client, who is not in the language tools business, had no interest in keeping them proprietary.

1.3 Lessons learned

In the end, the provided tool was far more powerful than initially required. Although the requirement was to simply minimize manual adjustments, it turned out that AdaSubst properly processed automatically several 100 000's SLOCs without *any* correction (except for ambiguities).

As for any other contract, the tool was delivered to the client with a warranty period. It happened that shortly after the end of this warranty period, the client reported a bug. Had the tool been developed under a conventional contract, we would have asked for a contract extension to make a fix. However, since at that time the tool was free software, we reacted like any developer of free software: we said "thank you for reporting this", and fixed the problem. This little story shows that by allowing the tool to be released as free software, the client eventually got better (and free) support than under a regular contract.

Even after the end of the contract, the tool continued to evolve and improve, thanks to the community feed-back. The client now has a better and more general tool than if it had kept it proprietary.

The approach was also beneficial to Adalog: the tools are commonly used inside the company, and many parts of them could be reused in other developments. For example, Adalog helped one of its clients in a migration to a different target; representation clauses from the original system were no more appropriate. It was easy to adapt Adasubst to provide a new functionality that commented out all representation clauses from the original program.

2 The case of AdaControl

2.1 Context

Eurocontrol (*European Organisation for the Safety of Air Navigation*) is developing programs to manage air traffic all over Europe. These programs are not life-critical, in the sense that a failure would not cause planes to crash, however a break-down of the system would cause huge delays for all airplanes flying over Europe; the software is therefore highly business critical. The system is made of very big programs (over 1.1 MSLOC), developed and maintained by a big team. With a project of this scale, it is not possible to rely on individual discipline to make sure that programming rules are being followed; Eurocontrol needed a tool to enforce programming rules and search for occurrences of bad or arguable programming practices. Thanks to the cooperation with AdaCore, some of these checks were incorporated into the GNAT compiler. However, many rules were deemed too specific to be put in a compiler, and it was felt that an independent controller program, allowing parameterizable rules, was necessary.

There can be many such rules, and it was expected that new ones would appear as more experience was gained by using the tool (and this expectation was verified quite rapidly). Therefore, the contract called for a general framework, where rules could be added at will with minimum effort, with just a minimum number of rules to be implemented as part of the original contract, to serve as a proof of concept.

The bid was granted to Adalog. It is interesting to note that since AdaSubst was free software, Adalog could show it in its response to the bid, as a show-case of its know-how in ASIS development.

Like the first client, Eurocontrol is not in the business of providing tools. On the other hand, such a tool was deemed useful to the community at large. Moreover, since the tool is easily extendable, Eurocontrol felt that it would benefit from the contributions of other users. Therefore, it was decided right from the start that the program would eventually be released as free software.

2.2 Solution

Like AdaSubst, AdaControl is based on ASIS. Actually, it is a perfect example of the kind of application that ASIS was intended for.

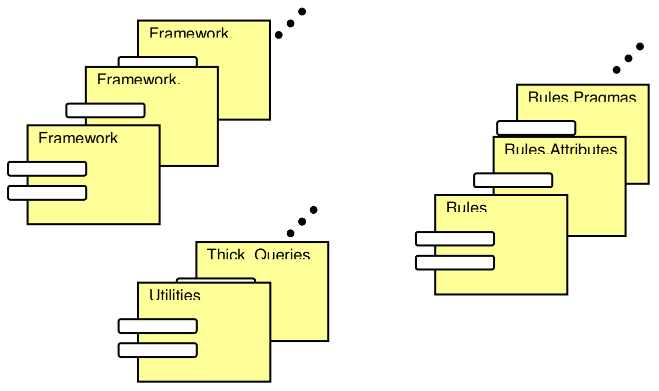
The structure of AdaControl has been designed to make the addition of new rules as simple as possible. It provides a general framework that hides all the internal machinery and offers a number of utilities that make the writing of rules easier: various services are provided to deal with complex issues like overloading, scope management, etc. Rules are plugged in a special module, and rule writers have to care only with the ASIS requests necessary to the rule.

Actually, AdaControl implements a full language to describe the checks that are to be performed. Utilities are provided to the rules for easy parsing of the rules' parameters. There is an interpreter for this language, allowing AdaControl to be used interactively as well as in batch mode. Rules just register themselves to the interpreter, thus adding new "verbs" to the command language, without needing to change the interpreter itself.

An important feature of AdaControl is that rules can be locally disabled by means of special comments in the code. This allows for local derogations to a rule, which is very important since there are almost always cases where general rules are not applied for good reasons. The mechanism for this is hidden in the module that reports errors, therefore the writer of a rule does not have to care about it: it is fully automatic.

Finally, the framework provides facilities for debugging rules. This is a great help since, given the complex structures used by ASIS, it is difficult to understand the origin of a problem under a debugger.

The overall structure of AdaControl is thus made of three well identified and separated parts: the framework itself (specific to AdaControl), general ASIS utilities (useful for any ASIS application), and the rules, as pictured below:



Important modules from AdaSubst were reused in AdaControl; this raised no copyright issue, since both programs were free software. In proprietary development, it is often the case that similar modules must be developed again, since it is not possible to provide a client with a module developed for another client!

2.3 Lessons learned

In addition to the framework, the initial bid required the implementation of only four rules. Later, an extension to the contract supported the development of three more rules. But since Adalog had similar needs for controlling its own programs, we developed other rules for our own benefit. The result was that the tool was delivered with more rules than contractually required, and the number of rules continued to grow after the end of the contract. At the time of writing (version 1.4), 25 rules are implemented (each with various parameters that allow them to check many things). It is expected that the number of rules will continue to grow as the tool gets more and more used.

As mentioned above, several modules were reused from Adasubst, especially those dealing with command line options and the way of specifying which units are to be processed (including integration with GNAT's ".adp" project files). On top of the usual benefits of reuse (no need to rewrite, test, debug), this brought two benefits that are rarely mentioned:

- Uniformity. Since the modules are the same, the user instructions for using Adasubst and AdaControl are the same.
- Reuse of documentation. Similarly, part of the user documentation for AdaControl was reused from the documentation from AdaSubst.

Thanks to the continued cooperation with Eurocontrol, all the rules were checked against Eurocontrol's software, thus providing an extensive test bench that would not have been available if Adalog had developed the product in-house.

2.4 The consortium effect

Since its initial release, the tool has raised interest in several other companies, which are willing to sponsor further development, including the development of more rules. At this point, the story of AdaControl seems to open the way to a new model of commercial free-software: cooperative development. The situation is that several companies, from totally different markets, needed a tool; none of them was willing to pay for the full development, and their interests were too different to even think of gathering them all in a consortium, just for the sake of developing the tool. This is however exactly what happened in practice: one company put the initial stake, other companies contribute in proportion of their particular needs, and in the end everybody benefits from a much more sophisticated tool than could have been developed (custom or in-house) by each of the companies separately.

3 Adalog's point of view

As explained above, releasing the tools as free software had a number of benefits for the client. But from a vendor's point of view, isn't it better to have a product that can be sold under a usual proprietary license?

First, it should be noted that developing a tool with the intent of selling it requires an important upfront investment. Such tools require many months of work, before even knowing if the tool will raise interest on the market place. By having the development paid under contract, Adalog could minimize the risk, and by having the tool released as free software, Adalog continued to have the opportunity of turning the development into a commercial product that can be offered to others than the initial customer.

Building a successful product for a client is always good for a company, but only the client knows about the quality of the work. If the product is released as free software, then anybody can assert the quality of the product. This makes good publicity for the company… and also attracts the sympathy of the community at large. It demonstrates Adalog's know-how in the development of custom language tools and its ability in ASIS development. The tools now form a suite of Ada semantic utilities, and we hope that they will attract new clients who need other similar tools (and will hopefully accept that they be released as free software too).

Since we consider these tools as fully commercial, Adalog is selling support contracts for them, and develops (paying) improvements on demand for clients with special needs. All this means more business opportunities.

There is also a "business attracts business" effect. Adalog has developed a custom analysis tool for a client, based on the same technology as AdaControl. The availability of AdaControl not only demonstrated the ability of Adalog for designing semantic tools, but also gave the client the idea of having a tool made to his own needs.

Finally, a side benefit is that the availability of these programs on Adalog's web site [4] attracts many people to visit us. Adalog uses a Web measurement service [5] to measure the popularity of its site; among 269 sites in the '"programming languages" category, Adalog ranks 9th, which is a good indication of its own popularity… as well as of the interest for Ada.

4 Difficulties

Sponsoring free software may create some difficulties, because it goes against a number of established practices. For example, all standard contracts stipulate that the product becomes the property of the client. This in itself does not preclude the software from being free, but in practice, for free software to grow and flourish, it is necessary to have a well identified, centralized entity to which contributions can be sent. To most users, this will be the name that appears in the copyright notice. It is therefore more convenient if the company that made the initial development keeps the copyright (possibly shared with the client, as was done with Eurcontrol). This must be negociated with the client.

The legal department of the client company may also on occasions be unaware of what free software really means, and raise concerns. It is then necessary to either educate the lawyers, or find an arrangement that does not raise issues of intellectual property. For example, it is possible to have a contract by which the provider must "provide a tool" (including a free one) to the client, and not state contractually that the tool is actually developed for the client. And of course, there is the issue of finding who, in the client's company, is empowered to sign the letter allowing the product to be released as free software…

Finally, it is clear that there are many tools whose nature is not appropriate for this model of development; this can work only for tools that are general enough to not require any problem domain knowledge (which clearly belongs to the client), and be usable in different application fields.

Conclusion

The story of AdaSubst/AdaDep and Adacontrol is another example that it is possible to develop *commercial*, but *free* software. Of course, Adalog is not the first company to take this approach: obvious other examples are RedHat (and others) with Linux distributions, MySQL for databases, and AdaCore with the GNAT compiler. However, our approach is different by using a business model that allows sponsoring the developments by various, unrelated companies, thus building a "virtual consortium".

In conclusion, releasing these tools as free software was beneficial to the industrials, because they have tools that are more powerful than if they had kept them proprietary. Moreover, they benefit from continued maintenance and improvements. But it is also beneficial to Adalog, as a showcase of what the company can achieve, and because it generates more business through custom improvements and maintenance contracts. And last but not least, it is beneficial to the Ada community at large, since anybody can use the tool.

Web references

[1] http://www.adalog.fr:
Adalog's home page

[2] http://www.axlog.fr:
Axlog's home page

[3] http://www.sigada.org/wg/asiswg/asiswg.html:
ASIS working group.

[4] http://www.adalog.fr/compo2.htm:
Access to Adalog components, including AdaSubst, AdaDep, and AdaControl.

[5] http://www.weborama.fr:
Web measurement service.