# PloneGov as an Open Source Product Line

Gaëtan Delannay
Easi-Wal
Rue des Brigades d'Irlande 2
5100 Jambes, Belgium

Kim Mens
Département d'Ingénierie Informatique
Université catholique de Louvain
Place Sainte Barbe 2, B-1348 Louvain-la-Neuve, Belgium
kim.mens@uclouvain.be

Patrick Heymans, Pierre-Yves Schobbens, Jean-Marc Zeippen
Computer Science Department
University of Namur
5000 Namur, Belgium
{phe,pys,jmz}@info.fundp.ac.be

September 1, 2007

**Abstract**

PloneGov.org is a recent community aiming at providing open-source software to local and regional governments. It gathers IT specialists from local and regional governments from Belgium, France, Switzerland, Spain and Argentina, supported by IT SMEs. Open-source communities are not used to think of development in terms of software product lines (SPLs), relying instead on the flexibility provided by source code modifications. However, if variability is not planned and managed, there is a risk that the consistency of the software deteriorates with many adaptations to local uses. We present preliminary plans to counteract this risk, and questions that we will have to address.

## 1 Introduction

### 1.1 PloneGov

The open-source portal and content management system (CMS) Plone is flexible and well-adapted to non-expert citizens. It is built on Zope (Z Object Publishing Environment) , an application server that provides facilities for the fast but safe development of Web-centric applications. These characteristics match well with the need of local

authorities, and have thus attracted the interest of their IT specialists. In Belgium, and then France, several communes (local councils) and the Walloon Region gathered in CommunesPlone, a project to share applications built on top of Plone. Similar groups developed recently around the world, besides Belgium, in France, Switzerland, Spain and Argentina. They united during their first meeting on June 1st, 2007 in the PloneGov international project. The goals of this close collaboration is to develop in a cooperative manner, applications and websites suited for public organizations for their own use as well as for their citizens.

PloneGov expected benefits, as stated in [Par07], are:

- to develop modules to address users' needs;
- to promote collaborative websites (managed by communal services, associations and businesses);
- to improve independence from IT service providers,
- to promote both the autonomy and the proficiency of the internal IT staff;
- to use the most recent technologies while avoiding license fees;
- to ensure publication of sources with General Public License (GPL).

Benefits could also be stated in the following terms :

- The use of a proven platform allows to start with small projects giving a rapid return on investment.
- Common values: The public sector and the open source communities share some values, like collaboration, openness, preeminence of general interest.
- Such an approach offers better chances to small and medium enterprises (SMEs) to bring their expertise in e-gov applications developments. Indeed, the Plone approach combines software components of different sizes. IT SMEs are more likely to bid for small developments. SMEs can specialize in specific niches (library management, forms for the citizen, etc.) and rely on the community to ensure an integrated usability of their products.
- The community spirit of open source is important for IT specialists in small teams, as is often the case for local councils. Shared enthusiasm is a strong motivation.
- Reusing others' modules is not only spares on new development but also yields higher quality software.
- Many developments that are out of reach of the small teams of local councils are within reach of a group.

On June 13th, PloneGov received the "Lutece d'Or: Grand Prix du Jury", in recognition of its potential impact.

## 2 Questions

PloneGov is just starting, and many questions, including research questions, are still open.

While Plone itself and its core components can be developed somehow independently of the context of use, PloneGov intends to offer a proximity service that needs careful tailoring. It is thus much more difficult to find the right separation between generic components, that can be shared and thus benefit from a larger community, and specific, tailored ones. These last ones will not benefit from community effect and it is important to ensure that they can co-evolve with more basic components at reasonable costs, since they will be maintained by very small teams. The first project that currently meets this problem is the application "College" currently being redeveloped as "PloneMeeting". Its goal is to plan and record meetings and decisions of a legislative or executive body, be it at a local or regional level (for example, a city council, a regional government or parliament). Actually, it could probably also be used for non-profit associations, non-governmental organizations (NGO), etc.

Its goal is to plan and record meetings of a local Council, but it is planned to make it adaptable to meetings of other bodies, e.g. Parliaments. (Actually, it could probably also be used for non-profit associations, etc.)

We propose to tackle this problem in phases:

1. First, a number of selected cases are analyzed to extract the commonalities and the variation points. For PloneMeeting, the cases of the Walloon Government and of a Belgian commune have been selected. Expected causes of variability are:
   (a) Workflows have different levels of complexity or have specificities from one level of power to the other;
   (b) The local autonomy often leads to local customs even if the context is similar: e.g. numbering of the agenda points.
   (c) For later cases, the language is a variation point; Spanish and Basque variants are already foreseen.
   (d) The legislation is slightly different in each country.
2. Second, a survey of the possible means to express and control variability in Plone, their respective merits and their fitness to requirements found in the first phase will be undertaken. Note that Plone, Zope and its products contain a wide array of configuration possibilities: the Plone "skins" allow to tailor the appearance of a site, the internationalization is supported by language-specific files ("po"), each sub-tool possesses many switches. Plone 3 proposes the mechanism of interfaces and adaptors to make the software even more flexible. Means to express variability are thus many, but it is not easy to find the switches to tweak, and then to ensure their consistency (two tools may e.g. require incompatible switch settings).
3. The selected means for variability will be proposed to the community.

4. When agreed, the tasks for development are distributed. Clearly, the task of the last tailoring must be performed close to the final users. A specific role is to maintain and oversee the consistency of the product line in the long term. The task of overseeing the product line will require the development of specific tools. We plan to reuse an existing notation for variability. From this notation, a tool (wizard) requiring no programming knowledge can be automatically derived. Even if variability management tools now exist [Bat04, CPRS04, vdML03], they need to interface with Plone/Zope/Python and to take into account the flexibility typical of open source projects. Also, the PloneGov products and their local variants must be kept in line with the evolution of Plone, which is currently making the transition to Plone/Zope 3.

5. We believe that there is a need for several levels of variability: The model above will deal with non-programmer customization, but a wider flexibility is available to the programmer. While non-programmers typically select in a closed list of predefined choices, programmers can provide methods, adapters, etc. that give rise to an infinite variability that still needs to be controlled.

6. The problem is made more complex in the open source context: all users can use the PloneGov modules without control, and are often initially not conscious of the usefulness of variant management to allow their specialization of the modules to evolve together with the main product. In the limit, this can lead to a forking of the project, loosing the economies provided by a shared development. Dealing with this problem is not purely technical: it requires to build a team spirit among the users of the products, to teach them the intended way to specialize the product and the interest of avoiding direct changes anywhere in the source code. Conversely, the project leader should introduce enough flexibility to accommodate the needs of most users in a generic way.

7. The PloneGov community often follows the successful approach of agile methodologies, characterized by:

   - strong involvement of the end users;
   - peer coding;
   - small, focused teams
   - rapid delivery of incremental versions of the software;
   - systematic testing;
   - frequent builds;
   - regular software refactoring, that can change the architecture

   This approach, however, seems difficult to reconcile with the management of software product lines, where variability is ideally identified as early as possible, i.e. during the requirements phase. A compromise that we would like to test is to integrate small teams that are in charge of the development of variants. Due to their different background, they would rapidly spot cases where too many assumptions are built in the generic product. A mechanism for variability

can then be proposed by the team leader, and rapidly tried out by each team for their variant. The variation points should be clearly identified and their assumptions documented. First examples of their use will rapidly be made available by the small teams. Generic unit tests for the variation points can detect broken assumptions.

8. A problem often met by agile development is to ensure a memory of the project. The problem is alleviated in Plone, where usually the UML Class Model creates the Plone data structures using ArchGenXML. We plan to also use State Diagrams to describe the associated workflows (as perceived by the objects). The data and state models is thus naturally updated. This is a simple form of Model-Driven Engineering (MDE). However, many subtleties also need documentation. Our proposal above will provide a specific documentation for the variation points, that would be gathered in a tutorial for customizers. Documentation for the product maintainer is also important in the long run.

9. Further than the consistency of the variant of a single Plone Product (e.g. PloneMeeting), PloneGov intends to provide a coordinated set of products for local governments and administrations, all built on Plone. This allows the products to interact, since their data will be available to all products. For instance, the product for the meetings of local council (PloneMeeting) could share data with a product for planning and tracking road works: the works have to be proposed to the council, and the implementation of the council decisions can be tracked and reported to the council. It is hoped that most such interactions are indeed positive. They will have to be managed, since they assume some sharing between products. However, the history in other domains (like telecoms) shows a risk that negative interferences also occur. A process to detect and solve feature interferences will thus probably be needed. It will in turn impact on the variability management, since these interferences and wanted interactions might only show up in particular instances of the products. It is known that such interferences are difficult to handle by an open source community, since contributors are often interested only in a few combination of products and will not take the trouble to check the whole product line for interactions.

# 3 Conclusion

We do not examine here how to integrate open source software in a product line [LM06], but how to integrate product line concept in an open source community. This paper brings no answer, but rather a set of questions that we believe will become important and are often overlooked by open source communities, and a plan to address them in time for the development of the PloneGov project. Today, to the best of our knowledge, these question are largely unanswered, neither in the academic literature nor in the experience of practitioners. To validate the proposed answers, we will need to collect estimates of the trade-offs of our approach: what effort should be invested in the initial planning of the product line; how much supplementary effort is then required during development to comply with the variability standards; at what time does it pay

off; what are the final benefits. We think that other open source projects will also face these questions, and we hope that exchanges of experience will allow progress on these questions.

# References


[Bat04] Don S. Batory. Feature-oriented programming and the AHEAD tool suite. In *ICSE*, pages 702–703. IEEE Computer Society, 2004.

[CPRS04] Vaclav Cechticky, Alessandro Pasetti, O. Rohlik, and Walter Schaufelberger. XML-based feature modelling. In *ICSR*, volume 3107 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2004.

[LM06] J.J. Lee and D. Muthig. Feature-based determination of product line asset types. in-house, cots, or open source? In F. van der Linden, editor, *1st International Workshop on Open Source Software and Product Lines, OSSPL 2006 : 22 August 2006 - Baltimore, Maryland*, 2006.

[Par07] ZEA Partners. Présentation du projet plonegov au ń lutèce d'or ż, June 2007.

[vdML03] Thomas von der Maßen and Horst Lichter. Requiline: A requirements engineering tool for software product lines. In Frank van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, LNCS 3014, Siena, Italy, November 2003. Springer Verlag.