

A Reference Model for F/OSS Process Management

Michel Pawlak & Ciarán Bryce
Object System Group
University of Geneva
Geneva, Switzerland
{pawlak,bryce}@cui.unige.ch

Abstract

Free and Open Source Software (F/OSS) constitutes a large class of the software used today. A self-organizing community with no centralized control produces it. This poses operational and efficiency problems for large-sized projects such as most of the major F/OSS projects. This presentation will describe a process reference model (PRM) and model that captures the activities, roles and resources of the process, that allows reasoning about process coherency and efficiency.

1 F/OSS Process Management

As users of open source projects know, several technical problems appear as the project grows in size. One is dependency management (the problem of identifying and locating the set of packages that need to be installed – or removed – when a given package is installed). This problem increases as the frequency of releases increases. Another operational problem is testing: many F/OSS software errors are configuration errors that cannot be detected by the distributor given the multitude of configurations that he needs to test. These errors are only detected once the software is deployed on the clients. A third operational problem is code distribution, and in particular, the overhead of distributing software to a huge number of end-clients. As the number of users grow, then the latency involved in downloading software from one of a set of mirror servers increases, especially when releases are frequent, as does the effort needed to keep the mirrors up-to-date.

A growing F/OSS project community also faces organizational challenges. F/OSS is more than the simple production, testing and deployment of software. It also involves activities such as community management, organization of seminars, production of manuals, etc. A community member may even decide to start a new activity related to the project, and this can be as varied as starting a sub-project, fund-raising or server maintenance. The plethora of activities poses the following organizational requirements:

- Possibility of locating competence in the community. Managing an activity entails harnessing the competence of community members. Competent and potentially interested members must be located. Apart from news-groups and mailing lists, there is no way of actively locating potential activity collaborators.
- Information about activities and participants need to be made available to the community. For instance, a user seeking to install a package must be immediately informed of any detected configuration error. Similarly, coding activities must be aware of information from testing; development activities need to be informed of information on community profiles produced by community management activities.

The technical issues mentioned above are also related to inadequate information flow between activities. In the case of F/OSS dependency management, there is inefficient information flow between the development

of packages activity and the download and installation activities. In the case of configuration defects, clearly there is a need for a testing and defect reporting activity that is closely intertwined with the download activity. To optimize downloading, it is important for a client to precisely specify the packages or type of software he requires so that superfluous packages are not transferred.

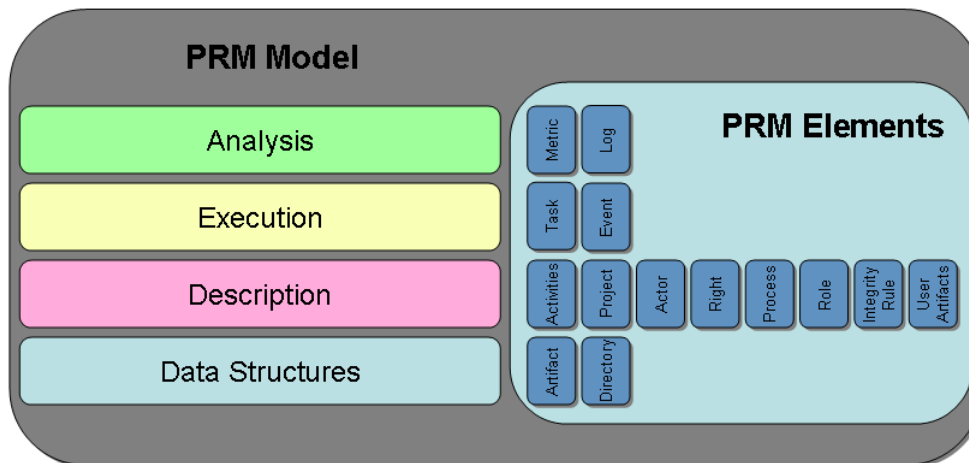


Figure 1: PRM Overview

To address these issues, we have developed a process model for F/OSS that encapsulates all activities and artifacts. The main elements of the model are presented in (the right-hand side of) Figure 1.

- The core data types of the model represent each F/OSS *artifact*; a type is defined for each entity managed within a project, e.g., packages, test reports, community members, projects, etc. An artifact definition specifies *attribute* artifacts to capture the relationship between artifacts, e.g., a package artifacts must have a valid software license as an attribute.
- The description layer of the model defines artifacts related to management. An *activity* for instance formalizes a set of operations with a well-defined purpose, e.g., testing operations, community management operations, documentation translation procedures, etc. A *process* is a set of computational steps, based on activity operations. It encapsulates the actions that some member of a F/OSS project is responsible for. A process is a first-class entity in our model – it can be declared and instantiated. An *Actor* is any member of the F/OSS project community. Actors execute processes, but for security and management reasons, the actors permitted to execute each process is controlled. Thus, each actor is assigned a set of *Roles* that determine his *Rights* to artifacts in the scope of the project.
- The instantiation of a process is known as a *task*. A running task has an actor and role associated. *Events* are asynchronous communication objects for processes. For instance, a bug fixing process that specializes in kernel bugs is programmed to react to events representing kernel defect reports; these events would be generated by a kernel test process.
- The final layer of the model concerns measurement. One of the core requirements of any process is performance feedback. These can be varied in nature; in F/OSS, examples can include the number of bugs per month, the number of developers specialized in cryptography, the number of code submissions, etc. The model thus permits *metrics* to be dynamically declared and bound to artifacts, and a *log* to be maintained.

We contend that such a model addresses the efficiency concerns of the F/OSS process since it has the following flexibility:

1. *Flexibility* of look-up. An end-user is able to locate code units based on properties such as functionality, platform requirements or license. Similarly, a community member can locate other members by specifying competences or topics of interest. This is possible since all relevant properties are expressible as attributes of the model's artifacts.
2. *Integrity* of artifacts. The model not only defines F/OSS attributes, but operations to create and manipulate artifacts. These operations ensure that the precise set of attributes are bound to an artifact. Thus, a code package cannot exist without the exact set of dependency information and a license attribute.
3. *Cross-activity coordination*. Attributes clarifies interaction between activities. For instance, defect reporting entails binding defect report attributes to code artifacts; these in turn can be localized by participants of a debugging activity based on the report attributes.

The presentation will outline the model in more detail. We will describe the formal model and its Java-based implementation, and work through a small case-study that exploits the model.

Acknowledgments This work is being conducted in the context of the EU 6th Framework project EDOS (grant number FP6-IST-004312).