A Reference Model for Free/Open Source Software (F/OSS) process management

Pawlak, Michel

UNIVERSITÉ
DE GENÈVE

FACULTÉ DES SCIENCES
ÉCONOMIQUES ET SOCIALES

A Reference Model for Free/Open Source Software (F/OSS) Process Management

THÈSE
Présentée à la Faculté des Sciences Économiques et Sociales
de l'Université de Genève par

Michel Pawlak

pour l'obtention du grade de
Docteur ès Sciences Économiques et Sociales
mention Systèmes d'Information

Membres du jury de Thèse

MM	Ciarán Bryce	docteur, directeur de recherche à l'INRIA, Rennes - France
	Francois Déchelle	professeur, ECP - Paris - France
	Dimitri Konstantas	professeur, directeur de thèse
	Michel Léonard	professeur, président du Jury

Thèse n^o 643
Genève, 2007

La Faculté des Sciences Économiques et Sociales, sur préavis du jury, a autorisé l'impression de la présente thèse, sans entendre, par là, émettre aucune opinion sur les propositions qui s'y trouvent énoncées et qui n'engagent que la responsabilité de leur auteur

Genève, le 24 septembre 2007
Le Doyen
Bernard Morard

Impression d'après le manuscrit de l'auteur.

Remerciements

Si de toutes les pages de cette thèse je ne devais en garder qu'une, ce serait celle-là, car elle résume en quelques remerciements l'aventure merveilleuse que j'ai vécue ces dernières années.

Tout d'abord, je remercie les membres de mon jury. Le professeur Dimitri Konstantas qui m'a permis de réaliser cette thèse au sein du groupe ASG et dont le soutien tout au long de mon parcours à l'Université de Genève, de ma license à l'obtention de ce doctorat et les nombreux conseils m'ont été extrèmement précieux. Le Docteur Ciarán Bryce pour son encadrement, pour nos discussions constructives et nos confrontations d'opinion qui ont forgé mon esprit critique et ont fait mûrir cette idée qui est devenue aujourd'hui une thèse. Le Docteur Francois Déchelle pour avoir accepté d'être mon jury externe, pour ses remarques pertinentes et enrichissantes, ainsi que pour son regard d'expert sur ce travail. Merci enfin au Professeur Michel Léonard, pour nos discussions toujours intéressantes et qui ont contribué au regard que j'ai aujourd'hui sur ce domaine passionnant que sont les Systèmes d'Information.

Je remercie également les membres du projet EDOS dans le cadre duquel ce travail a été réalisé et plus particulièrement Stéphane Laurière de Mandriva pour son avis d'expert, ses questions, critiques constructives et son enthousiasme : nos discussions ont été une véritable source d'inspiration pour moi.

J'adresse un grand merci à toute l'équipe présente et passée du groupe ASG. Vous, avec qui j'ai travaillé, fais chauffer mon cerveau, ai rigolé, voyagé, fais la fête et me suis arraché des cheveux : Manu, Jean-Henry, Giovanna, Jean-Marc, Katarzyna, Alfredo, Michel, Chrislain, Dimitris. Je garderai un excellent souvenir de ces années que nous avons passées ensemble.

Je tiens également à remercier Mme Monties pour avoir cru en moi et de m'avoir permis de terminer cette thèse dans d'excellentes conditions. Cette fois c'est fait !

Je n'oublie pas non plus mes amis proches, eux qui ont rendu ces années agréables que ce soit lors repas, journées, soirées et nuits passées à discuter, faire la fête et à ne pas se prendre au sérieux. Merci à vous: Dorothée, Carole, Marc, Jorge, Guillaume, Patrick, Cristina. Un merci tout particulier à Nicolas pour avoir débattu de ces idées et de bien d'autres avec moi, Alessandro, autre compagnon de route, pour ses encouragements et conseils avisés et Jimmy pour son amitié et son soutien de longue date.

Merci à ma grand-mère, Zofia qui habite malheureusement trop loin, ce qui ne l'empêche toutefois pas d'être à mes côtés. Ma marraine que j'adore, Ewa, pour son soutien, sa bonne humeur et son sourire. Mon oncle, Krzysztof, à qui je dois beaucoup. Enfin, mes cousins Nicolas et Michel: merci pour les rires, les fêtes et ce "manque de sérieux", qui font que les moments que nous passons ensemble sont si agréables.

Enfin, un grand merci à ceux qui me sont les plus proches, mes parents pour leur soutien inconditionnel tout au long de ma vie, leur confiance, leur amour, leur force, leur patience, leur sourires et leurs conseils avisés. Merci de m'avoir donné la chance de pouvoir en arriver là et merci d'avoir toujours été là pour moi. Je vous aime et j'ai énormément de chance de vous avoir.

Pour finir, merci à la femme que j'aime, toi Ludmila. Merci de m'avoir soutenu jusqu'au bout. Merci pour le bonheur que tu me fais vivre, tes petits plats succulents, tes gateaux, ta force, ta confiance, ta bonne humeur, merci de m'avoir redonné le sourire quand ca n'allait plus, de m'avoir toujours encouragé, d'avoir toujours cru en moi et de m'aimer. Tu es mon plus beau cadeau.

Acknowledgements

If I had to choose to keep a single page out of this document, it would be this one as it describes in some thank you's the story I lived these last years while working on this thesis.

First, I would like to thank the members of my thesis' commission. Professor Dimitri Konsantas who gave me the opportunity to do this thesis in the ASG group and whose continuous support during all the years I spent at the University of Geneva, from my bachelor's years until the end of my PhD, and his numerous advices have been precious to me. My advisor, Doctor Ciarán Bryce for all the constructive discussions we had, for the opinion confrontations we had, which forged my scientific spirit and helped grow this idea which finally became a PhD thesis. Doctor Francois Déchelle for having accepted to be the external member of the commission, for his enrichissing remarks and for giving me his expert's opinion which helped improving the quality of this work. Finally I would like to thank Professor Michel Léonard, for our always interesting discussions, which contributed to the viewpoint I have over this fascinating field Information Systems is.

I also thank the members of the EDOS project, in the scope of which this research work was done. A very special thank you to Stéphane Laurière from Mandriva for his expert opinion, his questions, constructive critics and his enthusiasm : our discussions have been a real source of inspiration for me.

I would like to thank all current and past members of the ASG group. You, whith whom I worked, made my brain overheat, laughed, traveled and had fun : Manu, Jean-Henry, Giovanna, Jean-Marc, Katarzyna, Alfredo, Michel, Chrislain, Dimitris. I will always keep an excellent souvenir about all these years.

I would also like to thank Miss Monties for having believed in me and for giving me the opportunity to finish this thesis in excellent conditions. This time it's done !

I don't forget my friends thanks to whom these years were fun, be it when going out, when talking or having fun. Thank to all of you: Dorothée, Carole, Marc, Jorge, Guillaume, Patrick, Cristina. A special thank you to Nicolas for having discussed with me the ideas presented in this PhD and many others, Alessandro, for having cheered me and for his useful advices and Jimmy for his longtime friendship and support.

Thank you to my grandmother, Zofia, who unfortunately lives too far, but always is by my side. My aunt, Ewa, for her support, her joy, and her smile. My uncle, Krzysztof, to whom I owe so much. And my cousins, Nicolas and Michel: thank you for the laughs, the parties and for not being serious. This makes these moments we spend toghether so pleasant.

The biggest thank you is for the closest persons to me, my parents, for their unconditionnal support all over the path of my life, for always believing in me, for their love, their strength, their patience, their smiles, and their precious advices. Thank you for having given me the opportunity to reach this point and for always beeing by my side. I love you and I am extremely lucky to have you as parents.

Last but not least, I thank Ludmila, the woman I love. Thank you for your support, for the happiness you make me live, the excellent meals you prepare for me, your cakes, your strength, for believing in me, for your joy. Thank you for making me smile, for bringing me up when I was down, for having always believed in me and for loving me. You are the most precious treasure I have.

À mes parents, À Ludmila...

To my parents, To Ludmila...

Résumé

Le logiciel libre (F/OSS) constitue une part importante des logiciels utilisés de nos jours. Ce type de logiciels est produit par une communauté auto-organisée n'ayant aucun point de contrôle centralisé. Une des caractéristiques principales du F/OSS est la mise à disposition des sources du contenu produit (du code, de la documentation, etc.) Tous les utilisateurs de contenu F/OSS re coivent un accès aux sources, l'autorisation de les modifier et de redistribuer le contenu selon les conditions définies par la licence employée. Tous les projets F/OSS doivent gérer différents aspects tels que la production du contenu, la gestion des tests, la correction des bugs, la distribution, la documentation, mais également la gestion de la communauté, etc. Nous appelons le processus F/OSS le processus englobant tous les processus liés à la gestion de ces aspects.

La nature distribuée de l'environnement F/OSS ainsi que la dépendance à la participation volontaire d'une communauté sont des contraintes peu communes aux environements traditionnels. En effet, le succès de n'importe quel projet F/OSS dépend directement de sa capacité de stimuler la participation de la communauté et de sa capacité à rassembler des compétences. La dynamique élevée d'un tel environnement pose également des problèmes. Le contenu développé évolue rapidement, les projets apparaissent et disparaissent vite, les gens joignent les communautés et les quittent ce qui rend difficile la capture de connaissances. Les projet étendent leurs activités, doivent les adapter et les intégrer avec celles existantes. Enfin, vu que les membres participent en tant que volontaires, les projets ne peuvent pas les forcer à contribuer.

Actuellement, les chefs de projet F/OSS manquent de structures et d'outils adaptés à un tel contexte. Ceci les empêche d'avoir une vue d'ensemble du processus F/OSS. En effet, les outils existant se focalisent sur divers aspects du processus tels que la gestion des tests, des bugs ou de la distribution de contenu, mais ne couvrent que rarement plusieurs de ces aspects. Un certain effort a été fourni afin d'intégrer certains d'entre eux dans un outil ou un modèle commun, toutefois il n'y a aucune solution ni modèle fournissant une vue globale de l'environnement F/OSS. Une telle solution devrait permettre la description des activités d'un projet, la déclaration des ressources utilisées, des processus impliqués, la description des rôles et l'attribution de tâches dans un environnement distribué. Elle devrait également fournir aux projets F/OSS des moyens de mesurer le processus F/OSS et de l'analyser. Elle devrait fournir aussi un cadre permettant aux projets F/OSS d'évoluer. La situation actuelle pose des problèmes aussi bien organisationnels, qu'opérationnels qui ont un impact négatif direct sur l'efficacité des projets F/OSS qu'ils soient grands ou petits.

Afin de répondre à ce problème, cette Thèse propose un modèle de référence pour processus F/OSS (PRM). Ce modèle captures les éléments clé liés à cet environnement, à savoir les activités, les rôles et les ressources. Le modèle permet de travailler au niveau de la cohérence et de l'efficacité du processus dans son ensemble en tenant compte des particularités de cet environnement. Le PRM travaille sur trois niveaux différents: le niveau descriptif, le niveau d'exécution et celui d'analyse. Le niveau descriptif a pour but de décrire les projets avec leurs activités, les ressources que ces activités impliquent, les processus et rôles existants dans le projet. Il permet de combiner efficacement les processus ainsi que de décrire la communauté F/OSS, en termes de compétences et d'intérêts existants. Le niveau d'exécution, lui, définit les tâches devant être assignées aux membres de la communauté et permet la communication

inter-processus. Le niveau d'analyse, enfin, offre un moyen de mesurer le processus F/OSS en créant des métriques transversales aux projets et en donnant les moyens de les évaluer. La séparation de ces aspects procure aux chefs de projet un moyen de décrire clairement le fonctionnement interne de tout projet, tant du point de vue fonctionnel que de celui des charges et responsabilités à assigner ou de celui de l'analyse des résultats obtenus.

Le PRM force les projets à penser à ce qu'ils font et aux raisons pour lesquelles ils le font de cette manière. De plus, il fournit un moyen de décrire et d'expliquer comment un projet est géré. Les informations liées à la gestion de projet telles que les activités d'un projet, les processus, rôles, tâches , métriques, peuvent être partagées. Ces informations peuvent alors servir comme base de comparaison entre les différents processus utilisés par différents projets F/OSS. Une telle approche va au delà de la conception de base de F/OSS en ne proposant pas uniquement de partager du contenu mais également les méthodes de travail, ce qui est un pas en direction d'une gestion de projet elle-même F/OSS.

Le PRM est une base pour l'élaboration de Systèmes d'Information adaptés à l'environnement F/OSS. Bien que le PRM soit principalement con cu pour gérer et améliorer des processus distribués tels que le processus F/OSS, il peut toutefois être employé pour gérer n'importe quel type de projet.

Abstract

Free and Open Source Software (F/OSS) constitutes a large class of the software used today. It is produced by a self-organizing community with no centralized control. One of the key characteristic of F/OSS is that it guarantees users an access to the sources of produced content (code, documentation, etc.) Depending on the license under which the content is released, F/OSS content users receive the authorization to access these sources, modify them and redistribute the content under specific conditions. All F/OSS projects have to deal with different tasks such as the production of content, its testing, debugging, distribution, documentation, but also with the management of the community, etc. We call F/OSS process the umbrella process gathering all the processes related to the management of these aspects.

The distributed nature of the F/OSS environment and the dependency to a voluntary effort are particular constraints which are unusual to traditional project management. For instance, the success of any F/OSS project directly depends on its ability to foster community involvement and to gather competencies. Further, as community members are volunteers, projects cannot force people to contribute. The high dynamics of the environment also pose problems as content evolves fast, projects appear and also disappear, people join project communities and leave them, which makes knowledge volatile and hard to keep. Projects extend their activities and new ones need to be integrated with existing ones.

Currently, F/OSS project managers lack structures and tools for supporting such specificities which makes them unable to consider the F/OSS process as a whole. Indeed, tools exist for handling some of its aspects such as debugging, testing, distribution, but they are rarely covering these aspects globally. Some effort has been put for integrating them in a common tool or model, however there is no solution nor model providing a global view of the F/OSS environment. Such a solution should enable the description of the activities of a project, the declaration of the resources it uses, the declaration of involved processes, the description of roles and the assignment of tasks in a distributed environment. It should also provide F/OSS projects with support for measuring the F/OSS process, analyzing it, and making it evolve. This situation poses organizational, operational and efficiency problems for large-sized projects but also for small and emerging ones.

This Thesis tackles this problem by providing a process reference model (PRM) that captures the activities, roles and resources of the F/OSS process. The model allows reasoning about the coherency and efficiency of the process as a whole. The PRM works on three different levels: descriptive, execution and analysis. The descriptive level describes projects along with all their activities, the resources each activity involves, the processes existing within the project, the way these processes are organized and the different roles existing within the project. It also describes the F/OSS community in terms of competences and interests. The execution level handles tasks assignment to community members. It also enables inter-process communication. Finally, the analysis level provides a means for measuring the F/OSS process through transversal metrics which can be evaluated. The separation of these aspects provides project managers with the ability to clearly describe the internal details of each project, assign responsibilities to the community and analyze obtained results.

The PRM forces projects to think about what they do and about why they do it that way. Further it provides a means to describe and thus explain how a project is managed. Project management information such as project's activities, processes, roles, tasks, metrics, can be shared and put into repositories.

Such repositories could then serve for instance as a basis for comparing the processes used by various F/OSS projects. Such an approach goes beyond the usual F/OSS concept by not only sharing the content but also by sharing working methods and is a step toward F/OSS project management.

The PRM is meant to be the basis for the development of Information Systems adapted to the F/OSS environment. While being primarily designed for providing a means to handle and improve distributed processes such as the F/OSS one, it can however be used to manage any kind of project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Free and Open Source Software (F/OSS) is one of the great facts of software development of the past few years. In this model, a community of people with common interests collaborate to develop new ideas, models and, *in fine*, produce freely available software. The software is distributed with its source code which can be freely modified by any developer. The community is self-organizing – as opposed to a large software company, there may be no hierarchal control. Proprietary software is inherently *cathedral*-like, "'carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time"' [155]. While some F/OSS projects are also developed in the cathedral model, it has revealed, through Linus Torvalds' Linux project, another mode of development: the *bazaar* model.

In a F/OSS project, the interests of the community push design requirements and software licenses regulate intellectual property rights. A F/OSS community is responsible for all aspects of software development, from requirements to coding, testing, and even manual compilation and translation. F/OSS projects gave birth to ambitious developments such as Linux, Apache, PhP or MySQL but also open standards, frameworks and other models which are widely used among the community and enterprises. Further, large technology firms such as IBM and Sun have become major supporters of this phenomenon with projects like Eclipse [34] or OpenOffice.org ooo.

Nowadays, the F/OSS model is not only applied to software development. Indeed, this model provides enough flexibility to be adapted to different domains, fostering community effort in order achieve a common goal in a highly collaborative manner which is not seen in the closed source world. This includes for instance knowledge sharing [198, 200], health [127, 175, 186] or science [165].

1.1 Technical and Organizational Challenges

While providing flexibility, the bazaar model implies new constraints: information availability, information accessibility, information integrity, activity coordination and process management. These constraints are mainly due to the fragmented nature of the environment, the distribution and independence of actors and activities, the dependencies between projects and the highly dynamic environment of F/OSS. As of 30 October 2006, the standard Mandriva Linux distribution counts 10566 packages, with an average package size of 2717431 bytes. These package involve more than one hundred different licenses [112]. Gaim [71], one of the most active projects on SourceForge, had in August 2006 an average of 533 read transactions and 17 write transactions per day for an average of 101 files updated per day. These examples show the extreme diversity, dynamics and reactivity needs of such an environment. Due to the constraints implied by the F/OSS environment, F/OSS projects can encounter operational and organizational problems as they grow in size.

Potential technical problems, mainly concern information integrity, information validity or latency. Consider the following examples:

- **Dependency management** is 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 - and thus the complexity - of releases increases.

- **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.

- **Code Distribution.** The overhead of distributing software to a huge number of end-clients is another technical issue to be solved. 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. Errors or latency problems during code distribution can lead to inconsistencies in the software installed on an end-user machine.

Further, a growing F/OSS project community faces also 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, starting of new projects. 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, for instance, the following organizational requirements:

- **Locate competence in the community**. Starting and 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 collaborators. For instance, it can be hard to locate a community member who may be helpful for a certain task, e.g. to fix or develop a specialized package or organize a seminar on the project's software.

- **Effort allocation and reallocation**. There might be several contributors working on the same topic for the same code package, unaware of the efforts of each other (such as fixing a defect). Further, effort reallocation may be difficult, not knowing and not being able to predict the workload of contributors.

- **Information availability and sharing**. 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, defect information must be published for all concerned members to see.

- **Ensure coordination and information flow between activities**. Software artifacts and information produced by an activity must be made available to another activity in order to enable their smooth integration. For instance, coding activities need to be aware of information from testing; development activities need to be informed of information on community profiles produced by community management activities.

These technical and organizational issues impede process wide analysis. Indeed, the lack of information as well as the lack of control over available information makes process streamlining difficult and thus global F/OSS process improvement. We contend that these issues can only be satisfactorily addressed by considering the process as a whole, providing a way to represent and handle F/OSS related information and enabling process wide analysis.

1.2 Information Management Challenges

Increasing the size of a F/OSS project community brings great potential to a project – more ideas, code and developers. However, since there may be no hierarchal control or regulated coordination, projects can suffer from inefficiency and operational concerns as they become large. In some situation, needed information *may* be held by some contributor or *may* be available somewhere. However, as most of the contributors are free to leave at will, there is no guarantee that the information is really held, or will be available in the near future, nor that any contributor has ever kept this information. Similarly, there is no standardized way to collect or retrieve such information, nor guidelines indicating which information needs to be collected and made available by the tools being used.

Another concern is that F/OSS projects needs mechanisms for improved information availability, not just to address efficiency concerns, but also for correctness with respect to project artifacts and with respect to the way project processes should be run. Indeed, to be able capitalize on available information and achieve productive analysis, integrity rules should be enforced, and thus a way to declare them should be available. For instance, each package meta data must include complete dependency information and a complete list of patches; it should not be possible to create artifacts that do not have the complete set of meta-data or a corrupted one. Ensuring and maintaining the integrity throughout projects' processes is mandatory for long term efficiency.

Formally, F/OSS is an example of a virtual process. It involves a set of activities (e.g., coding, testing, community management, etc. in F/OSS), operations within these activities (e.g. test, enroll user, assign task, etc.), resources (e.g., packages, configurations, community), roles (e.g., developers, project committers) and sub-processes. The execution of this virtual process produces data. Such data includes the activity being involved, the operation being called or the role of the user who triggered the operation. Interpreting this data can be useful in order to detect process-wide issues. Indeed, once put in context, the data becomes information which potentially can enable process improvement. Possible improvements include modifying the process, by modifying the artifacts used by the process, modifying a sub part of the process, or reallocating roles given to actors.

The technical and organizational challenges mentioned above are often related to inadequate information flows between activities. Indeed information needed to take decisions, is not available or not accessible. In the case of F/OSS dependency management, there is insufficient 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.

To tackle this issue by gathering information, making it available and ensuring its integrity, fundamentally, the bazaar model needs to be extended with *observers*. These observers act as the eyes and ears of the bazaar; they do not regulate the bazaar but keep note of all that happens. By maintaining information on the state of the bazaar, they can be queried to ensure that activities are operating correctly and to obtain information directly without latency.

The huge amount of data which can potentially be collected, its complexity and volatility, makes impossible to assign the role of observers to human people. However, endorsing such a task is the role of Information Systems.

1.3 Toward an Information System for F/OSS

Information systems have been defined as the entire infrastructure, organization, personnel, and components for the collection, processing, storage, transmission, display, dissemination, and disposition of information [132]. The purpose of Information Systems is to provide a means – be they organizational, community management, procedures, or computer systems – for handling, exploiting and managing

information in a given context. In the case of F/OSS this context is defined by involved activities and resources.

Currently, F/OSS projects, to support their development, make use of collections of tools dedicated to one or many aspects of the F/OSS process. For instance, SourceForge [170] or GForge [72] provide an interface to a set of key tools for producing content, managing project's developers, communicating with the community, dealing with defects, allocating tasks and distributing content. Other tools focus on specific activities such as defect management, testing management or content management.

Nevertheless, while these tools are integrated, and while hardcoding bridges is always possible, none of them integrates all aspects of the F/OSS process, respects a common communication model or eases the integration of new features with existing ones. Indeed, these tools focus on some activities of the project, and present selected information to users who are not able to fully customize this information, nor link data in order to produced meaningful information. For instance, while the community is a key resource of F/OSS which is bound to most of the Activities, be it development, testing or defect management, little emphasis is put on this aspect. Project community is often managed through mailing lists or in the best cases, through wikis and content management systems. Even then, available information is sparse or no control over it is provided. Needless to say that when it comes to integrating more specific activities such as product support or e-learning one can hardly find adapted solutions.

Further, the flexibility of the bazaar model, does not impose on projects a single way to handle F/OSS process. Project managers are free to decide how they lead the project, the underlying processes and which information they expose. On the one hand, this flexibility is wanted and leaves each project to define its own way of functioning. On the other hand, projects and software have to be able to communicate in the scope of the F/OSS process. Due to the lack of structures it may be difficult to change a tool supporting a specific activity due to the incompatibilities with the replacing tool and integration with other used tools. Such interoperability and integration issue are especially related to the lack of available information, data format, activity and process integration.

In order to be able to improve project processes, it should be possible to correlate information manipulated and held by the different tools and integrate it in a common view. While F/OSS Activities are tightly coupled and interacting, related tools remain fragmented.

For these reasons, we argue that existing tools supporting F/OSS process cannot be considered as Information Systems adapted to this kind of environment with no central point of control. Usual available approaches for designing Information Systems do not fit the constraints of F/OSS environments. An adapted information system should enable the build of F/OSS project dashboards providing an overview of any aspect of the F/OSS process. It should also provide a means for modeling the F/OSS process, integrating projects, different activities and resources.

As the tools issued from F/OSS projects are more and more used within enterprises such issues become central. Indeed, long considered as opposed, the F/OSS and the enterprise environment are currently mixing. Enterprises make investments into Open Source development and an increasing amount of Open Source Software are used by enterprises. But enterprises are not only funding F/OSS an increasing number of them build their core business around F/OSS. This is the case for instance of Linux distributions such as Mandriva Linux [110] or Caixa Magica [17].

While small projects can benefit from an Information System aimed at F/OSS and providing efficient way to handle underlying processes, such support for F/OSS process management becomes mandatory in an enterprise context. Indeed the latter have obligations, need to meet deadlines, increase productivity, lower costs, optimize available resources, know when and where they should reallocating them, be able to easily move from a tool to another one and have control over their processes.

Thus, having control over the F/OSS Process Management appears to be central in the Enterprise context, especially when this process becomes part of the core business of enterprises and when enterprises have to face constraints and issues similar to the ones implied by the F/OSS environment. Existing project management frameworks and approaches [146, 201, 152] tend to be limited to some aspects of

the F/OSS process such as the development, distribution or testing. They often ignore the organizational aspects of the process. They also ignore process interactions, leaving open the question of process integration. We contend that F/OSS processes can only be improved by addressing process issues in a process-wide manner.

1.4 Focus of Work

Until now, Information Systems design has always been based on the core idea that the information to be manipulated is well known and that there is control over it. However, the F/OSS Process is a combination of multiple activities which are often held by different entities. Associated responsibilities are completely distributed and, unlike Information Systems as they are currently known in enterprises, there is no centralization of any kind, and thus no common ground of understanding. As a result, in the F/OSS environment, information is not explicitly held or are difficult to obtain. Information linking and information retrieval which may appear trivial in the Enterprise context become in the F/OSS context a headache.

The aim of this thesis is to define a generic model, the F/OSS Process Reference Model (FOSS-PRM or PRM), providing the key elements and formal means for building Information Systems able to tackle the issues specific to the F/OSS environment. Such a model has to consider the specific requirements and constraints that environment entails. These requirements and constraints are manifold. A first step of this research work highlights these by analyzing the F/OSS model, its evolution and the hot challenges it is currently facing. The PRM provides a means for formalizing the F/OSS process on three distinct aspects: a descriptive level, execution level and analysis level.

Descriptive level This level aims at describing the F/OSS process and making available related information. Projects can define involved resources, involved activities (such as production management, testing management, defect management, distribution management, community management, role management, project management and metrics management), how it is possible to interact with these activities, to describe the internal processes linking these activities, the different processes to be assigned to the members of the community and how the roles are organized within the project. The process model permits different F/OSS projects to be compared based on the resulting description.

Execution level The description provided by the descriptive level helps organizing the Process, comparing F/OSS projects and linking them. The execution level provides a way to execute the processes built on the descriptive level. The execution is ensured through the use of tasks and events. Tasks are assigned to community members to make them responsible of executing a process. The Event scheme of the PRM can be used to synchronize processes or to enable inter-process communication.

Analysis level The third and last level relates to the analysis of process execution and stored information. Information held by the PRM can be used and accessed by community members needing it. This information can be used to feed F/OSS dashboards for instance. In order to enable fine-grained analysis, the PRM provides a Metric element. A Metric is the description of a measurement to be done using information provided by project activities. It can be evaluated to generate needed information. Further, it allows to achieve transversal measurements involving multiple activities.

1.5 Contribution

The contribution of this Thesis is summarized as follows:

Understanding of F/OSS Challenges. The first part of this Thesis analyzes the F/OSS environment. We highlight the evolution this environment has been subject to during the past decades, from the very first idea of *Open Source* to the current integration of F/OSS with the Enterprise environment. Main problem areas are outlined and related requirements are extracted. Based on this analysis, we define how to structure F/OSS information in order to be able to streamline and manage the F/OSS process, considering the particular constraints implied by the F/OSS environment.

A Model for F/OSS. We propose then a model for F/OSS that formalizes the associated virtual process. The result is a Process Reference Model for F/OSS (FOSS-PRM, or in short PRM) that permits the abstraction of bazaar observers to be implemented. The PRM is a reference model linking user applications to implementations of different activities such as testing, defect management, distribution tools, etc. This model sets up the core of what a F/OSS-oriented Information System (FIS) should support. It provides a flexible way to structure information through Artifacts and manipulate them through basic primitives. The model enables activity management. This includes content and community management, rights, role, metrics, process and task management and defines the operations which have to be offered by FISs. Processes can be built on top of these activities and be measured even if no central point of control is available. The model is extensible as both new Artifacts and Activities can be declared.

The PRM forces projects to think about what they do and about why they do it that way. Further it provides a means to describe and thus explain how a project is managed. Project management information such as project's activities, processes, roles, tasks, metrics, can be shared and put into repositories. Such repositories could then serve for instance as a basis for comparing the processes used by various F/OSS distributors such as Debian [27], Mandriva Linux [110] or Ubuntu [188]. Such an approach goes beyond the usual F/OSS concept by not only sharing the content but also by sharing working methods and is a step toward F/OSS project management.

Model Application. FOSS-PRM is being defined in the context of the European Union project EDOS number FP6-IST-004312 (Environment for the development and Distribution of Open Source software.) The model has been extended to fit the particular constraints of the EDOS project (EDOS-PRM). As a result, the PRM is being applied in the following contexts: transversal process measurement for F/OSS decision making and distributed testing.

1.6 Thesis Overview

The structure of this thesis is the following. Part I presents the background underlying this work. Chapter 2 provides an overview of the F/OSS model, presenting its evolution as well as related problem areas, then the state of the art related to this research work is discussed chapter 3. The PRM model is described in Part II. Chapter 4 lists the requirements of the PRM model which is then presented in Chapter 5. Chapter 7 exemplifies how this approach can help structure and improve the F/OSS process. The usage of the model are then discussed in part III. Chapter 8 presents the implementation of the PRM as well as the environment in which the PRM model was evaluated. Model extensions for handling advanced metrics and testing are presented in chapter 9 and chapter 10. We finally conclude in chapter 11.

Part I

Background

Chapter 2

Free and Open Source Software

The *Free and Open Source Software* (F/OSS) model is a philosophy and methodology characterized by development and production practices providing access to the sources (usually source code, but also documentation, etc.) of produced content and by the authorization provided to the users of these sources to modify and distribute them under specific conditions.

Two different movements have put forward this philosophy: the Open Source Software (OSS) [138] and Free Software (FS) [62, 64]. Despite the fact that OSS and FS are two separate approaches distinguished by the philosophical reasons leading to access provision to source code, i.e. a development methodology in the case of OSS and a social movement in the case of FS [174], their aim is similar: giving access to source code. For this reason in this thesis we make no distinction between them nor with other approaches. Instead, we group them all under the umbrella name F/OSS.

F/OSS has concretized ideas which modified the whole software production environment and which are part of current software panorama. These ideas included aspects such as making software available to anybody, enabling software upgrades retrieval whenever it was needed. Another aspect was to enable software versions switch when changing hardware architecture. Other aspects were focusing on enabling software improvement by users to make it more powerful, to adapt the behavior of the software to users' needs, correct bugs, or even enabling software maintainance even if the entity or company that produced it went out of business. More than being wished features, these ideas were considered as users' rights for the F/OSS movement and set the background for the so called *software freedom*.

Thus the F/OSS philosophy was mainly driven by the will of making software widely available, in order to enable its fast improvement, to foster innovation as well as to provide means to enable increased adaptability in the highly changing software and hardware environment. To achieve this goal the core idea was to put together competencies, knowledge and create communities around common projects.

However, as nobody would like to put lots of work into a program, then see anybody sell improvements done without giving anything back, some limits had to be fixed. Indeed in order to foster the involvement of contributors and make them comfortable with sharing their knowledge and the software they produce, the following set of rights were defined [151]:

- The right to use the program without any restriction on the domain of use.
- The right to make copies of the program, and distribute those copies.
- The right to have access to the software's source code.
- The right to make improvements to the program.

These rights form the basics of F/OSS as they provide the liberty wanted by F/OSS community members: they keep all contributors at the same level relative to each other, ensuring thus continuity in ideas development. Further refinement of these rights to make fit F/OSS to specific needs have been

done using different licenses, [137, 65] documents containing the rights and obligations software users accept, matching whether FS [63] or OSS [139] requirements. F/OSS aims at sharing information and does not put limits on content usage, as long as the key rights of giving access to content sources is ensured.

As collaborative work and information, not to say knowledge, sharing are appealing for information production in current global context, F/OSS philosophy has been the inspiration for increased liberty in other fields and has been thus adapted then applied to other domains such as open formats such as the office documents format [133] created and used by the OpenOffice.org project [143], content management such as open encyclopedy [198], dictionary [200], global business listings [205, 108] (all supported by the mediawiki [114] tool provided by the wikimedia foundation [197]). Other application fields include pharmaceutical development [127, 175] having led to the creation of the Tropical Disease Initiative [186], and more globally science through the Science Commons project [165], which aims at serving the advancement of science by removing unnecessary legal and technical barriers to scientific collaboration and innovation. Support for government with open voting [140] or even beverages formulaes [142, 191] are other domains where this philosophy has been applied..

While this adaptivity, the growing proportion of F/OSS content having key positions in the software landscape and the appearance of business models involving F/OSS are warrants of its success, questions about the correspondence of existing tools to the particular environment F/OSS is or to the new challenges it has to face have to be raised. Thus, this chapter sets up the background of this thesis, introducing the F/OSS model and related process, highlighting the constraints implied by this environment as well as the new stakes to be tackled, and defining how the F/OSS Process can be improved in order to put it in line with todays F/OSS requirements.

The structure of this chapter is the following. Section 2.1 provides a description of the F/OSS environment, providing a brief history, taxonomy and highlighting its evolution. In section 2.2 the issues related to the management of the F/OSS process are discussed. Then in section 2.3 we describe major stakes the F/OSS model is facing and which need to be tackled in order to improve the underlying process. Finally we provide a set of constraints involved by the F/OSS environment in section 2.4

2.1 F/OSS Environment

This section provides means to understand the F/OSS environment and the stakes which are involved having led to the research described in this thesis. A taxonomy including common artifacts used within the F/OSS model is first provided. Then the articulation of these artifacts in the scope of the F/OSS process is described. The evolution this model is currently undergoing is then analyzed.

2.1.1 History and Philosophy

Writing about F/OSS not specifying the origin of this philosophy would leave many questions open about the evolution this environment follows. Therefore, we explore briefly in this section the history of this movement, highlighting different aspects of the underlying philosophy.

Sowing the seeds of independence

The history of Free and Open Source Software started with the birth and evolution of the UNIX operating system. In 1969-1970, Kenneth Thompson, Dennis Ritchie, and others at AT&T Bell Labs began developing a small operating system named Unix on a PDP-7. In 1972-1973 the system was rewritten in the programming language C, an unusual step that was visionary: due to this decision, Unix was the first widely-used operating system that could switch from and outlive its original hardware.

While AT&T continued developing Unix under the names “System III” and later “System V”, the accademic community, led by Berkeley, developed a UNIX variant called the Berkley Software Distribution (BSD). In the late 1980’s through early 1990’s the confrontation between these two major strains was at its top. After many years each variant adopted many of the key features of the other. Commercially, System V won the “standards wars” by getting most of its interfaces into the formal standards, and most hardware vendors switched to it. The BSD branch did not die, but instead became widely used for research, for PC hardware, and for single-purpose servers such as web servers.

In 1984 Richard Stallman’s Free Software Foundation (FSF) [63] began the GNU project [60], a project to create a free version of the Unix operating system. By free, Stallman meant software that could be freely used, read, modified, and redistributed. In the mean time, Bill Jolitz’s operating system, 386BSD, departed from BSD as a return to UNIX origins but in modern form. It was first released inside the University of California and the US Department of Energy in 1989. Major parts were released in the Networking II (Net/2) release that was labeled by University of Californa as "freely redistributable". These two parallel trends represent the first steps toward F/OSS.

While the FSF was promoting complete freedom, it had in the 1990’s [61] trouble developing a free operating system kernel, the major component needed to provided a free operating system. In 1991 this issue was solved when Linus Torvalds began developing an operating system kernel, which he named “Linux” [185]. When in 1992 Linux became free, its combination with GNU components, BSD components and MIT’s X-windows software resulted in a complete operating system.

In the early part of 1993, the last 3 coordinators of 386BSD patchkit: Nate Williams, Rod Grimes and Jordan Hubbard, created the FreeBSD Project [67] in reaction to the abandon of 386BSD development. Around 1994, Novell and U.C. Berkeley settled their long-running lawsuit over the legal status of the Berkeley Net/2 tape. Indeed, a large part of the latter were property of Novell, who had in turn acquired it from AT&T some time previously. As FreeBSD was using Net/2, it had to get free from this commercial part. FreeBSD community then had to literally re-invent itself from a completely new and rather incomplete set of 4.4BSD due to large chunks of code required for actually constructing a bootable running system having been removed due to various legal requirements. It took the project until November of 1994 to make this transition, which represented the access to freedom for this operating system.

The Open Source Definition [138] started life as a policy document of the Debian GNU/Linux Distribution, which was built entirely of free software. However, since numerous licenses purported to be free, Debian had some problem defining what was exactly free and had to make its free software policy clear. These problems were addressed by proposing a Debian Social Contract and the Debian Free Software Guidelines in July 1997. The Social Contract documented Debian’s intent to compose their system entirely of free software, and the Free Software Guidelines made it possible to classify software into free and non-free easily, by comparing the software license to the guidelines.

At the beginning of 1997, Eric Raymond was concerned that business people were put off by Richard Stallman’s freedom approach, which was very popular among the more liberal programmers. He felt this was hampering the development of Linux in the business world while it flourished in research. In 1998, Netscape announced that it planned to release Navigator, its browser, as an Open Source project, and Raymond was invited to help them plan this action. Raymond used this opportunity to sell the free software idea strictly on pragmatic, business-case grounds, the ones that motivated Netscape. This created a precious window of time to market the free software concept to business people and to teach the corporate world about the superiority of an open development process. Companies that use open source software have the advantage of its very rapid development, often by several collaborating companies, and much of it contributed by individuals who simply need an improvement to serve their own needs.

While the Debian Free Software Guidelines were the right document to define Open Source, they needed a more general name and the removal of Debian-specific references. Bruce Perens edited the Guidelines to form the Open Source Definition a bill of rights for the computer user. It defines certain

rights that a software license must grant the user to be certified as Open Source. Then a certification mark, a special form of trademark meant to be applied to other people's products, was registered. Eric Raymond and Bruce Perens have since then formed the Open Source Initiative (OSI) [139], a non-for-profit organization, exclusively for managing the Open Source campaign and its certification mark which is governed by a six-person board chosen from well-known free software contributors. F/OSS as we know it was born.

As of 1^{st} February 2007, there are 140,417 registered projects and 1,498,326 registered users on SourceForge.net [170] the world's largest Open Source software development web site. Table 2.1 lists some of the most important open source projects available today and Table 2.2 lists some of the available Linux and BSD distributions.

Project Name	Description of produced software
Ajax	Interactive Web Applications
Apache	HTTP Server
Azureus	BitTorrent client
Eclipse	Extensible Integrated Development Environment
Firefox	Web browser
Gaim	Messenging
GCC	C Compiler
Hibernate	Persistence and query service
Jboss	Application server
KOffice	Office Suite
MySQL	Relational Database
Openoffice	Office Suite
PHP	Hypertext Preprocessor
The Gimp	Photo retouching, image composition and image authoring
Thunderbird	Mailer
Tomcat	Servlet container

Table 2.1: Examples of F/OSS Projects.

Different approaches to a common philosophy

The concept of F/OSS is thus an old one. It is a philosophy with different arguments promoted by different approaches focusing on different aspects of freedom and each setting a different threshold to the limits of this freedom. The reader should keep in mind that when computers first reached universities, they were research tools. Software was freely passed around, and programmers were paid for the act of programming, not for the programs themselves. Only later on, when computers reached the business world in the 70's-80's, did programmers begin to support themselves by restricting the rights to their software and charging fees for each copy. As defined earlier, the core idea of all Free and Open Source Software approaches remains to gather people around a common project, to make it grow, share knowledge, then share the benefits of the resulting cooperation by giving access to the code. The goal is the same, only the constraints and available means to reach it differ. As an example, here follow the description of three of these approaches.

FSF approach. Free Software as a political idea has been popularized by Richard Stallman since 1984, when he formed the Free Software Foundation (FSF) and its GNU Project. The moto of FSF is that people should have more freedom, and should appreciate their freedom. In order to reach such

freedom, a set of rights Stallman felt all users should have was defined, and codified in the GNU General Public License or GPL. It gives users four main rights or freedoms: the freedom to run a program for any purpose, to study how a program works and adapt it to user's needs, to redistribute copies to help the community and the freedom to improve the program and release them to the public. Stallman developed initial works of free software such as the GNU C Compiler, and GNU Emacs. His work inspired many others to contribute free software under the GPL.

FreeBSD approach. The goals of the FreeBSD Project are to provide software that may be used for any purpose and without strings attached. For FreeBSD, the first and foremost mission of F/OSS is to provide code to any and all comers, and for whatever purpose, so that the code gets the widest possible use and provides the widest possible benefit. Thus the FreeBSD License permits the redistribution and use in source and binary forms with and without modification as long as the source code retains the license itself and a disclaimer and as long as redistributions reproduce these elements in the documentation and/or other materials provided with the distribution. The FreeBSD license helps developers avoid the additional complexities involved by code released under the GNU General Public License (GPL) or Library General Public License (LGPL) in a commercial use context.

OSI approach. The purpose of the Open Source Initiative (OSI) [139] is to inform the commercial world that open source is a rapid evolutionary process which produces better software than the traditional closed model in which only a few programmers can see the source and everybody else must blindly believe in this closed work. Although it is not promoted with the same libertarian fervor, the Open Source Definition promoted by OSI includes many of Stallman's ideas, and can be considered a derivative of his work. While GNU's Freedom emphasizes the importance of the principles of liberty and freedom, OSI's Open Source is more pragmatic, emphasizing the technical merits of code developed in an open fashion. To achieve this goal, the Open source Definition is a list of ten criteria, software licenses must comply with in order to be considered as Open Source licenses: free redistribution must be ensured; un-obfuscated source code must be provided; derived works must be allowed; integrity in the authors source code must be ensured; in order to get the maximum benefit of the process, no discrimination against people or groups can be allowed; no discrimination against fields or endeavor is allowed in order to prohibit traps that prevent open source to be used commercially; the distribution of license defines that the rights attached to the program must apply to to whom the program is distributed; license must not be specific to a product; it cannot restrict other software, it must be technology neutral [138]. Multiple licenses comply with this definition [137]

Mandriva Linux	RedHat Linux
Fedora Linux	Suse Linux
Debian Linux	Gentoo Linux
Ubuntu Linux	NetBSD
FreeBSD	

Table 2.2: Examples of F/OSS Linux / BSD Distributions.

2.1.2 From garage to Enterprises and Public Administrations

Since its creation the F/OSS environment faced many changes. Long seen as a development model opposed to usual Enterprise development models, it was considered in the Enterprise environment as a threat for proprietary software. Slowly, with the release of Netscape source code under an Open Source license, the border separating F/OSS from Enterprise environments blurred, then faded out, and now

synergies progressively replace previous oppositions. For instance shared source licenses such as Microsoft's Shared Source Initiative [121] or Sun Microsystems's Community Source License [179] are a step toward giving access to the sources to end users and providing them with benefits not available in traditional proprietary software licenses, while defining restrictions when needed. This license family defines restrictions which can range from the most restrictive such that it could be only be viewed and to the least restrictive that it could be viewed, modified, or redistributed the source code for either commercial or non-commercial purposes [199]. Now this evolution is supported by public administrations which played and continue to play a central role by promoting, producing and using F/OSS content.

Indeed, nowadays, there is a concrete political awakening that Enterprise environment and F/OSS environment are closely related. Several European Projects explore related issues. FLOSS project [54] provided interesting results through surveys concerning the use of F/OSS in firms and public administrations, the motivations and policy implications of such use, as well as on the policy in the Public Sector within the European Union [55]. These results mandated for government support for Enterprise / F/OSS integration. Ongoing CALIBRE project [18] aims at establishing a European Open Source industry forum to foster the transfer of F/OSS best practices to European industry. The Consortium for Open Source in the Public Administration (COSPA project) [22] aims at analyzing the effects of the introduction of Open Data Standards an F/OSS in European Public Administrations. Existing projects not only analyze the impact of Open Source on the enterprise or try to integrate it with the Enterprise, but they also seek to improve F/OSS in order to make it better fit Enterprise needs. For instance, the EDOS Project [1] aims at structuring the F/OSS distribution process, make it more efficient, integrate different activities in a common view and cut down release cycle time.

European and International Public Administrations rely more and more on F/OSS. For instance, in France, public sector institutions increasingly use F/OSS solutions for their IT systems since 1998. Concerned sectors include for instance the Ministry of Defense, the Ministry of Justice, and the national crime register, Ministry of Economy, Finance or Industry. Countries like Malaysia have a Open Source Awareness Programme for educating and assist Public Sector users in adopting and implementing F/OSS solutions [109]. The Open Source Observatory of the IDABC [74] (Interoperable Delivery of European e-Government Services to public Administrations, Businesses and Citizens) offers a good overview about OSS-related government activities in Europe and abroad.

A large number of F/OSS projects currently widely used were initially developed by well known companies and have industrial partners. To give a few examples, Mozilla [126] was a fork of Netscape and its current partners are IBM, Sun Microsystems, Hewlett Packard or Red Hat. The Eclipse platform [34] was originally released into Open Source by IBM and is now a Foundation having support of 50 member companies hosting 4 major F/OSS projects including 19 sub projects. Project JXTA [101] started as a research project incubated at Sun Microsystems to address peer-to-peer space. Up to day the JXTA holds 117 projects in 6 categories and about 27 enterprises and 29 universities have contributed to the project. The office suite OpenOffice.org [143] has been released into F/OSS by Sun Microsystems who is still the primary contributor.

Further, Enterprises release into F/OSS their proprietary products and focus on service providing or on specialized products development. Eclipse is a good example of a successful synergy between the Enterprise and F/OSS worlds. IBM takes advantage of the inputs from the community which helps with the improvement of the F/OSS platform and sells its product WSAD built on top of Eclipse. In the mean time, the community provides new plug-ins, other companies also provide plug-ins and tools which integrate with the platform and IBM products creating thus a win-win situation where everybody takes advantage of the base platform: IBM, the F/OSS community and other companies.

From the license perspective, in order to protect this model and the community in cases where the F/OSS environment merges with the commercial environment, the Eclipse project has been released under the terms and conditions of the Common Public License (CPL) [23]. Indeed, this license has special entries concerning grant of rights and commercial distribution. It specifies that *each Contributor hereby*

grants Recipient a non-exclusive, worldwide, royalty-free patent license in order to protect contributors from being suited for patent infringement when using code under the CPL. The CPL specifies too that only Commercial Contributors, i.e. contributors reselling products released under the CPL, are liable and responsible for potential damages caused by the product. In case of damage caused by a Commercial product using non commercial code, only the Commercial Contributor has thus to pay those damages.

F/OSS and enterprise as well as public integration have matured during the last few years and are now converging. The F/OSS environment now starts supporting the Enterprise environment with various projects. For instance Open For Business (OFBiz) aims at providing applications and tools made to easily and efficiently develop and maintain enterprise applications [7]. The growing availability of Open Source products such as enterprise resource planing (ERPs) and consumer resource management (CRMs) [171, 129, 147, 21] is another sign of the Enterprise - F/OSS integration.

F/OSS has now evolved and the *garage era* has been left behind. Therefor, new constraints coming from the Enterprise world have to be faced. Among other constraints, these include delays imperatives, return on investment imperatives, collaborator management issues, management efficiency, continuous measurements and analysis, etc. Therefor, F/OSS process management has to be adapted to this new reality. To tackle these new issues efficient project and process management is mandatory. This implies the existence of means to achieve F/OSS information management.

Nevertheless, common approaches to build Information Systems do not fit the particular constraints the F/OSS environment imposes. Next section explores these constraints and highlights the reasons that make available means to build Information Systems unadapted to such an environment.

2.1.3 Taxonomy

When describing the F/OSS environment, multiple terms can be used. These terms represent the key elements involved in the environment and are often called F/OSS artifacts. This section provides a taxonomy of the main F/OSS terms, which will be used in this thesis.

Project F/OSS project is a gathering of people, or F/OSS Community around an idea. Indeed, a F/OSS project aims at providing F/OSS content developed by a community of contributors to a community of users. Each project defines the type of content, which is produced, the schedule associated to this production and the different activities which are involved. Further, each project manages people contributing to these activities and specifies the roles these contributors may have in the scope of these activities. Finally, roles are associated to the different contributors.

Resource F/OSS Projects involve multiple resources which can be classified as two fundamental types: community, which represents the man force at project disposal, and content. By content, we mean anything produced by F/OSS projects. This can be source code, binary code or documentation. There is no real difference from the process viewpoint between documentation and code. For instance, some of the OpenOffice developers produce natural language translations of user manuals rather than code; their content gets packaged, tested and linked to others in the same way that code from developers gets treated. All units of content within a distribution need to be consistent with one another, and this is also true for documentation.

Activity Each F/OSS project involves different activities focusing on specific domains management. These domains include for instance the management of the community, production of content, content testing, defect management, content distribution, measurements, etc.

Role F/OSS roles represent the obligations and rights community members have as F/OSS project contributors. Depending on the role a community member has, he will be, for instance, able to run tests,

submit patches, commit modifications made to project's code base, enroll other community members, or mentor them.

Process Each F/OSS project has a set of processes specifying what has to be done in which order within the project. A process involves a community of actors who consume and produce resources, a multitude of activities and tools. A process can define for instance the different steps, which are involved in the production of a piece of content, from its development to its testing, defect correction and then distribution. Processes orchestrate thus activities. Each process is bound to a role authorized to execute it. We call F/OSS process the umbrella process gathering all processes of a F/OSS project.

2.2 Understanding F/OSS process

Since the end of the nineties, F/OSS has become a key feature of the software industry. As defined previously, in this model, content is produced by a self-organizing community of actors. Members of the community, take on the different tasks. For instance *developers* produce content; *testers* test it toward different use cases and *distributors*, package software and make it available for community download. The lack of hierarchal control between the different activities has led to the F/OSS development model being termed as having a *bazaar* organization compared to the controlled *cathedral* model of proprietary software development projects [155].

The process underlying F/OSS is a combination of different subprocesses corresponding to the different activities it involves. The bazaar model leaves open the way these activities are led. The following set of processes is usually considered as the core of F/OSS.

The development process provides the content produced by F/OSS projects. A documentation process provides information and explanation about the produced content. A testing process is meant to detect existing issues in the produced content, and to assert that – ideally – no errors remain when distribution of the content starts. Debugging helps correcting detected errors. An integration process may be needed in the case of content taken from external projects have to be adapted to the produced content. Packaging is the process preparing the content to be distributed, putting it in different formats (packages, ISO images, etc.) Then the distribution process aims at disseminating the content over the community using different means such as mirror servers or bittorrent. Figure 2.1 shows the combination of the different processes involved in the distribution of a Linux distribution.

A common mistake is to consider these processes as being completely isolated while they have to be considered as a whole. Indeed, efficient improvement of one process cannot be always ensured by simply improving it locally. For instance improving the distribution process to enable a more intuitive content retrieval allowing users to search content depending on the availability of selected features or allowing them to search for content through a request made in natural language cannot be done by only focusing on the distribution process. Such a search capability requires expressiveness and more details about the produced content than what may be available, which can only be added by content developers, packagers or even by other actors which would be responsible for this task in the scope of an additional process.

Further, local improvements are not a viable solution to transversal issues. Indeed, all processes described before are linked in a way or another. Content cannot be tested as long as it has not been produced; it cannot be packaged as long as a given stability level has been reached; it cannot be distributed as long as it has not been packaged, and so on. Thus, the different activities influence each other through information flows existing between them. This implies that transversal improvements can hardly be done through local actions. For instance if the improvement goal is to minimize the time needed between each release of a F/OSS project, the information dependences chain existing between the processes imply that the issue has to be tackled from a transversal perspective considering all activities and processes part of the process to be improved.

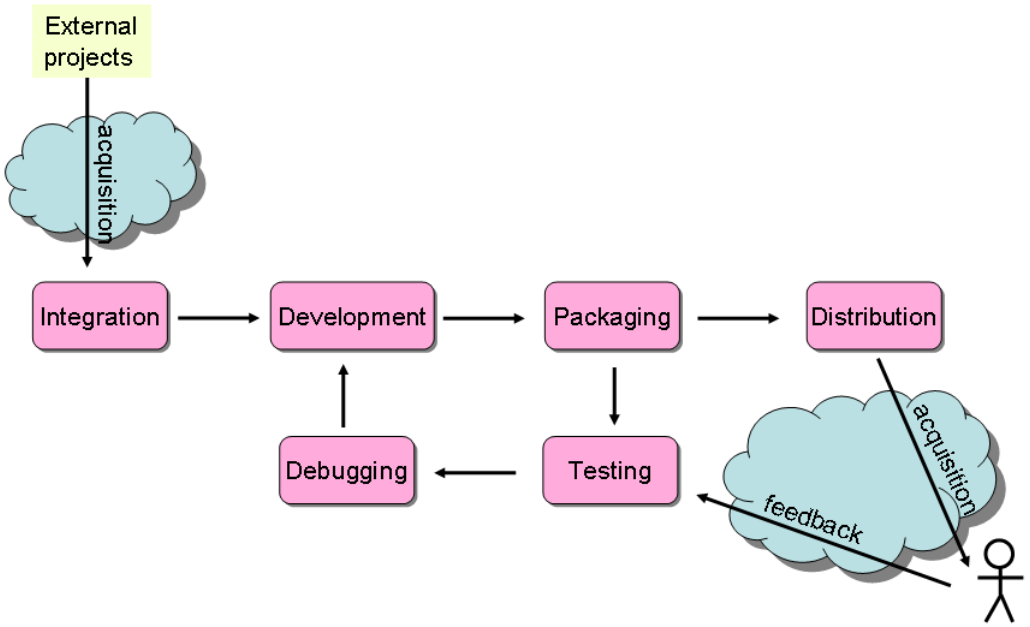

Figure 2.1: Example of production and distribution process

However, considering processes in a transversal manner may still not be sufficient to improve efficiently the F/OSS process if only production processes are considered. Indeed support processes also have to be considered but are usually neglected be it for for a lack of interest in them, the lack of gratification working on them provides, or a lack of knowledge about their importance for the F/OSS process. More pragmaticaly, nobody has the time nor wants to achieve the tasks underlying them as they do not seem useful. These processes include efficient community management, enabling the handling of actors as resources which can be affected to different tasks depending on their knowledge, experience and interests; efficient role management, providing an overview of who is in charge of what task within a project; activity management, in order to know what are the different activities involved in a project; measurement and metrics management, in order to have a numeric representation which can be used to analyze the health of a project; or process management to define what has to be done within a project, what are the recurring tasks, and to be able to detect tasks to be affected to community members.

Despite the lack of interest in them, these support processes are central for multiple reasons. They handle information common to the different production processes and as such are a glue between them. Further, they can provide information which can help analyze and understand facts. For instance, while a huge number of defects indicates that something went wrong during the development phase of the content, it does not indicate the underlying reasons. However, knowing that the developers had twice the work they usually have or that a new not experimented developer was part of the development team, can explain encountered problems, and thus help in their tackling.

The efficient handling of support processes is thus mandatory to enable global F/OSS process improvement. While the bazaar organization leaves open the way the processes and activities are organized, it does not mean that it has not to be organized at all. In order to achieve improvements throughout the whole F/OSS process, means to handle and analyze it are mandatory. As the processes are interwoven the knowledge about resources used, facts, etc. should be globally available to be able to analyze the F/OSS process. Finally, information flows and dependencies existing between the processes need to be highlighted and as the bazaar model is open, the way the F/OSS process is handled should also be open, and thus reusable by different projects.

Currently, F/OSS is still considered as an aggregation of processes not considering thoroughly their integration. The main issue hampering a more efficient process management enabling F/OSS process improvement is that there is no guideline indicating how the F/OSS process should be organized, indicating what the core processes mandatory for an efficient project and process management are, nor indicating how to integrate these main processes with processes specific to projects. Thus, each project defines its process freely, but often leaving aside support processes, or relying on various services only providing simple support for these processes, which may not be sufficient when projects, their community and the number and types of possible issues grow.

In such a situation even if some projects consider these aspects of process integration, and recognize the value of support processes, it is extremely difficult reproduce the improvements they achieve as there is no model on which their approach is relying that could be reused. Even the evaluation of achieved results and the evaluation of the improvements that could have been done after issue detection if other information was available is difficult.

2.3 Stakes and Challenges

In the introduction of this Chapter, we presented some of the new ideas that were the source of motivation of F/OSS. These ideas have now become common sense and new ones are emerging. The F/OSS environment has now matured and the initial needs of sharing information have become needs to manage the process of sharing information. Currently a core need is to improve the F/OSS process by providing more control to project managers, as well as to contributors and users. As such, F/OSS resource, activity and process management represent the new stakes and challenges for the F/OSS environment. They define the features Information Systems tailored for F/OSS will have to be able to tackle in the future.

2.3.1 Managing Resources

Handling the community. The key F/OSS resource is its community. Projects live from the contributions made by the different members of the communities they are creating. The involvement of the contributors has a direct influence over the success of a project. Further as they are directly implied in all processes of a F/OSS project, their behavior has a direct impact on the execution of these processes. These contributors have a knowledge, interests and competencies which can be used to help achieve the goals of the project they are participating to. Managing a community implies being able to harness the competence that exists in the community. This is often the difference between successful and less successful projects. Such a profiling is useful to help evolve a community, e.g., to start a new project or to locate potential developers.

Another noticeable feature of F/OSS projects is the multiplicity of *roles* undertaken by participants, particularly committers. One task of the latter is to build packages; another is to build releases of the project's software, and yet another is to test packages before these are added to the project's code base. A developer is responsible for at least three tasks: he can produce code, signal bugs or run tests. Different roles are not clearly distinguished in current F/OSS projects. That is, a participant uses the same account – with the same privileges and information access – for each task. For instance, producing a package requires access to the content being packaged, and the information generated includes a formal package, package description information and a license. Accessed information is used by committers who are responsible for building an application and for package testers. However, as it has no importance for end-users who are only interested in the release, access to it should be impossible if the community member endorses the end-user role.

As such, projects have to foster interest of their community in order to increase the involvement of its community members. Further, this implies that community members must be handled as a precious resource and that the roles of each community member have to be properly split. Thus, the first stake

F/OSS information systems are facing is to enable the precise handling of projects' communities, which includes the following requirements:

- **Contributors overview.** Project's managers should be able to have access to a complete overview of their project's community members. Information being available should include the number of participants to the project, the interests, knowledge and competencies of each actor, the responsibilities each of them is being assigned, the achievements made by each member in the scope of the project, the results obtained and thus the quality of work done by each of them. This information is meant to provide a global overview of available resources, in order to use them at best.

- **Contributor search.** In addition to having access to an overview of the community, another requirement consists of having means to find contributors based on their properties. These properties can be various such as the interests of the community members, their competencies or experience. Such a search capability is central for instance for retrieving actors matching given needs and then assigning tasks to these people or proposing them to participate in newly created projects or sub-projects. Further, as required competencies may not be available within the community of a project, managers should have the ability to search for contributors beyond the community.

- **Responsibility assignment.** Another requirement of F/OSS oriented Information Systems is to be able to assign responsibilities to the different members of the community. The goal here is to define what an actor is allowed to do and what he has to do as part of his involvement in the project. Handling contributors assignments must be as easy as possible.

- **Workload overview.** In order to avoid situations where community members are overloaded by multiple tasks, and thus in order to be able to better plan responsibility assignment, another aspect of community management is the ability to have an overview of contributors workload. Such an overview is needed to avoid potential issues due to factors directly bound to the workload of the contributors. Examples of such factors include for instance an increase in software defects or manual tests to be run which may give to much work for the set of contributors responsible for related tasks. Being able to detect such situations can help react fast and reassign responsibilities and share them among more contributors.

Thus these requirements cover the most important aspects of community management: knowing the community, searching it for competencies, assigning responsibility and making sure that the workload of contributors is not too heavy in order to ensure a given level of quality.

Ensuring information availability. Being distributed, projects can involve multiple activities, multiple responsibilities which can be distributed over multiple actors and tools. In such a context, a challenge is to keep track of projects' activity, hold the knowledge produced by each project and then ensure the availability of this information and provide access to it.

Currently, some information is often held by individuals such as Project managers or developers, some is logged, some is not, making the reminder lost for the community. People holding information as part of their knowledge can potentially leave projects as they are rarely contractually bound to their responsibilities. Indeed, the whole F/OSS model mainly runs on a personal involvement and interest basis. Thus, an information system aimed at F/OSS management, should enable the construction of webs of information related to different projects, linking the different information held by different processes of each project, in order to produce global project knowledge.

Ensuring information integrity. As information availability, information integrity is mandatory in a context where information is completely distributed. Indeed information can be produced or modified by

activities external to the activity using it. This is the case for instance for packages which are produced in the scope of the production process, and are used by the testing process, or even for projects relying on other projects to achieve their work, which is the case for dependencies. In such situations, in order to avoid errors, inconsistencies or redundancy of verifications, another stake is to provide means to define and ensure distributed integrity rules.

An overview of enforced integrity rules should be available to information users. Indeed, users have to be aware of the rules being enforced when producing or modifying information in order to avoid its misuse.

Finally, these integrity rules should not only be data-based, enforcing possible values, but also activity, action, process and role related. Indeed, as some rules may depend on the context the information is accessed, the information integrity management provided by F/OSS information systems should be aware of this and provide enough flexibility to have such fine grained information integrity enforcement.

Enabling flexible resource retrieval. As for community management, a major challenge for F/OSS information systems is to ease the retrieval of information disseminated over multiple sites, tools and people. Information handling and access to it should be integrated in order to provide a innovative means for information retrieval. The latter should be able to gather information from the distributed environment, and be independent from resources' nature. Indeed, retrieval should be based on resource properties, the meta information associated to every resource, be it content or community.

Further, while locating distributed resources should be easy, being able to locate similar or replacement resources should be also made possible. Indeed, while searching for specific resources is common, distributed environments where resources are not and cannot be well known imply behaviors such as the search for resources having some properties and which can be used "in place of" other ones with no side effect.

For instance, users may want to search for applications based on their features: a spreadsheet or a calculator; they may wish to retrieve patches to correct a concrete bug; or may wish to use a license presenting some properties.

Dealing with licenses. One of the key features of F/OSS is that every resource is bound to a license which defines how the resource can be used. The purpose of this is to deny anybody the right to exclusively exploit another person's work [102].

A large amount of licenses can be used for Open Source. The Open Source Initiative (OSI) [139] provides a set of approved licenses which respect the OSI Open Source definition. The purpose of this list is to provides to the community a reliable way of knowing if a piece of software offers the qualities to be expected from Open Source Software. As of September 2006, 56 licenses are declared approved by the OSI [137]. Similarly, the Free Software Foundation comments a list of licenses and classifies them as GPL-compatible, GPL-incompatible and Non-Free Software Licenses [65]. The different types of licenses along with their differences are further discussed in [102].

While this multiplicity of license types provides a richness which enables F/OSS projects to better define the user community their product is aiming, it also raises issues related to license compatibility. Indeed not every license can be combined with each other. This is for instance the case with the General Public License (GPL) [66] which is not compatible with most of other licenses. While license compatibility issues can occur for instance in the case of composite projects which are a putting together interdependent different projects, it can also occur within single projects using different licenses or when integrating Open Source Software with an existing Information System. With the growing number of existing licenses, the combinational complexity is exploding.

This implies that the license choice has to be carefully done, and that the license is one of the criteria to be considered when selecting an Open Source resource to be integrated. However, currently there is no support for the early detection of complex compatibility issues in order to avoid them.

Handling dependencies. Another important stake for F/OSS is to improve the content resource dependency management throughout their historical evolution (versions/variants/forks, etc.) This is mandatory in order to avoid problems such as resource conflicts – mainly due to the use or provision of identical resources, be it files, pipes, network ports – or missing resources, but it is also needed to provide more flexibility to users, and make the use of F/OSS resources more intuitive for their users.

Multiple kinds of dependencies are involved. Build (or source) dependencies define the prerequisites for building and compiling a resource. Binary dependencies specify the prerequisites for installing a resource. Configuration dependencies provide the prerequisites for configuring a resource once it is installed. Conflicts specify the incompatibilities between resources at installation and configuration levels. Finally replacements define the relations between resources that may replace each other's functionality such as the mail server sendmail [167] can replace the mail server exim [48].

All these dependencies may be obtained by enriching the meta-data associated to content resources and thus improving the expressiveness of the language used to describe dependency relations. The enrichment of dependency meta-data, the separation of information related to different aspects of dependencies, and the ability to handle *n-ary* conditions for describing them [156] can provide increased flexibility for dependency management. It also guarantees the availability of means to ensure backward compatibility with existing tools. Having more expressive dependencies, is mandatory to enable sophisticated static analysis tools. Such analysis include for instance the detection of dangling dependencies, unsatisfiable dependencies, update closure, update selection [40].

2.3.2 Managing Activities

Handling Activities. Resources involved in the F/OSS process are being provided then manipulated by different activities. Each F/OSS project involves activities, some being handled internally, such as production or debugging, and some other being handled by external projects and being integrated in the process, such as an external testing service or distribution through mirrors or other means. In such a context it is mandatory to know these activities, and more specifically, the ones relying on internal services of the project and the ones relying on third party activities needed by the project.

Handling Roles. A key challenge for F/OSS projects is related to their ability to define precisely all the roles existing within the project. Indeed, security schemes can be extracted from this information, which are the basis for defining what contributors are allowed to do within the project. It is mandatory to know how these roles relate to the different activities involved. For instance, within the patching activity, which roles are able to submit new patches, and which ones are able to commit them.

Such information is essential for community management dashboards. Indeed, it provides a panorama of the exact roles and thus potential duties of any contributor in the project. This information can help detect which roles are not assigned. Based on this and combined with information related to the interests, knowledge and competencies of contributors, role assignment or reassignment can be eased. This ensures that contributors only have roles needed for executing the tasks they have been assigned, an thus avoids risks related to *forgotten rights* given to them. Finally, this can help detect situation where no contributors fits the needs of a specific role, and thus accelerate the enrollment process.

Sharing Activities. As F/OSS projects are meant to be open their interaction should be made as easy as possible. This implies that projects should be able to show to the external world, in a uniform globally understandable manner, what activities, and thus behaviors, a project offers to the world. This should be expressed as sets of interfaces indicating provided services.

Integrating new Activities. While F/OSS activities where traditionally related to development, testing, debugging, distribution or even documentation, with the spread of F/OSS and the emergence of business

models around F/OSS new activities not development related are appearing and need to be integrated with the traditional ones. This is for instance the case of support and training activities. F/OSS information systems should ease their integration in the existing pool of activities and roles, and enable the retrieval of most suitable contributors able to undertake the new roles

2.3.3 Managing the F/OSS Process

Handling Processes within a Project. In order to enable process improvement, a background easing process handling needs first to be provided. Currently it is extremely difficult to have an exhaustive list of processes involved in different F/OSS projects. Some research work has been done on automatic process extraction [95]. However, currently no project provides an overview of all processes, with the ability to trigger them and analyze them.

It may be also difficult to know who is in charge for the execution of the tasks related to these processes, and even more difficult to assign and reassign these tasks on the fly, depending on particular events such as for instance the express need of the competencies of a particular contributor which are not available anywhere else. In such a situation, all other tasks of this actor could be reassigned to other competent actors in order to free him and assign the task needing his competencies to him. Apart from contributor knowledge, other situations for task assignment and reassignment can be based on contributors workload, project priorities, etc.

Streamlining the F/OSS Process. While having a panorama of processes and contributors involved in a project is mandatory for enabling efficient project management, it is not enough. It is essential to be able to streamline them to reflect the fact that projects are graphs of inter-related processes. Thus it is essential to be able to chain processes, build work flows, measure them, create rendez-vous or synchronization points.

Measuring the F/OSS Process. The display of F/OSS process information is directly bound to the measurement of the underlying processes. Measuring each process enables to keep track of what is happening in the project and thus provides information to feed dashboards. In an open and distributed environment such as F/OSS, measurement is a real challenge. Knowledge about what has to be measured, why, how and when it has to be measured has to be kept. This information can help process analysis improvement over time by explaining why some measures are made, and how they have to be used even if the contributors having implemented them have left the project in the mean time. This also enable the listing of useful measures in order to keep a pool of generic metrics for future reuse or even for sharing measurement schemes with other projects. Ideally, measures should be built on the fly, even not expected ones, by registering them on the information system and making them use available information.

Current tools enable some basic measures however, the most useful ones, the ones providing information about the process as a whole and enabling metrics able to help with process improvement, need to access various information potentially provided by different activities. Some of these can be external to the project. For instance, while knowing the evolution of the number of defects registered for a project provides operational information, being able to combine this measure with other ones such as the workload (in the project, or even in other projects) of the developers having introduced these defects, can help analyze the situation. Thus the ability to build transversal measures, crossing the boundaries of used tools and projects are central to F/OSS process improvement.

Know-how sharing. While F/OSS is currently focused on code sharing, a challenge is to go toward know-how sharing. As projects grow in size, their need for managing their processes grows too. However, not all projects have the required know-how. Indeed, it is straightforward that all projects do

not have the resources and knowledge to achieve process description, orchestration, measurement and analysis.

Therefor, a major stake for F/OSS is to go toward a situation where large projects could share their know-how about process handling, measurement and evaluation with small projects. Projects having both labor and knowledge could define these metrics, document them indicating what they are measuring and how results have to be interpreted, do the same with key performance indicators and share them under a F/OSS license. Projects not having the resources needed to achieve such analysis, or just wanting to benefit from such features, could thus include them into their project with minimal effort.

Displaying information about the Process. Another challenge for F/OSS is to provide enough information to efficiently monitor the F/OSS process. Users should be able to build dashboards providing all the information they need to achieve their task, whatever the role of the user accessing the dashboard is. Such dashboards should provide operational information indicating how the processes are being executed, showing if this execution is in-line with the predictions previously made, if unexpected events are happening, etc. Such dashboards could also support strategic decisions, by enabling projections and estimations of the impact of decisions such as involving more contributors as testers or distributors. Another aspect to be considered is the detection of potential issues that may occur due to issues external to the Project. Finally, F/OSS dashboards should also give the opportunity to their users to interact with them, in order to react to the information and take decisions accordingly to the analysis they are doing. For instance, if a project depends on an external activity such as a testing service, if the latter lacks testers to achieve its task, project managers should be able to detect this, and move contributors to help with the testing activity in order to avoid delays are multiple bugs due to a lack of testing.

However, such advanced features imply the need to have precise access to information which is disseminated over the network, to the activities as well as to the processes involved in each project. To ensure such a transversal view of the distributed Information System and provide an integrated access to it, dashboards themselves need a transversal access to the information, processes and activities, depending on the role of the user for which it has to be displayed.

Building interoperable F/OSS Information Systems. Finally, F/OSS information systems raise new challenges regard to interoperability. The distribution of the different elements composing them, be it actors, activities, software components or processes imply the need for information format standards and information manipulation standards.

2.4 Environmental Constraints

Since its beginning, the F/OSS environment underwent multiple changes and has now to face new stakes and challenges. As F/OSS grew, new organizational requirements for handling growing communities, large and interleaved projects appeared. In the mean time the landscape of available tools for handling the different aspects of F/OSS also grew, and thus technical issues bound to the heteroclicity of these tools also appeared.

In order to be able to propose an adequate solution for handling these new challenges, F/OSS environment's particularities need to be highlighted. By nature, F/OSS implies a set of constraints specific to environments with no central point of control. They have to be considered as they play a key role in the design of information systems designed for F/OSS.

Community involvement. As described before, community contributions are mainly bound to the interests actors have in the topics of the projects, their will to help, to participate in large projects or contribute emerging ideas as well as to the will to understand, share knowledge and learn from others. The

success of a project depends on its ability to harness the participation of a community. Hence, measures improving F/OSS process include extra-technical aspects bound to ability to increase the percentage of end-users willing to use the tools provided by projects.

Unstructured environment. The F/OSS environment is unstructured regard to many aspects. From the resource perspective, information involved in the F/OSS process is not standardized. For instance, there is no rule indicating what meta data should be bound to packages, what a patch exactly is. All approaches are possible and thus depending on the projects, different resource description is possible. From the activity perspective, there is no rule defining what are all the operations which should be made available when providing an activity. For instance, it is difficult to know the functionalities a test management tool should provide and how it should be integrable with other activities, as defect management. As such it is difficult to know what the functionalities one can expect from such tools are. Finally, from the process and role perspectives, every project can have its own definition of the responsibilities assigned to a role such as *tester* or *distribution manager*, and thus, related processes such as *testing* or *distribution* can have multiple meanings.

Such lack of structures is part of the liberty provided by the F/OSS philosophy. However, in order to easily integrate projects and understand how projects are organized, on what information they rely and what information they can provide, a means structuring the environment while leaving maximum liberty to projects should be provided.

High Distribution. Multiple projects can be working on the same issues and not even know each other; for external user it can be difficult to find these different projects, and even more difficult to understand their differences be it for using the tools they produce or contribute to their production. Some projects like SourceForge [170] or GForge [72] offer a whole development infrastructure for F/OSS development. They provide a meeting point for producers and users where they can publish and find information. They classify projects, describe them and propose tools to browse them by properties such as development status, intended audience, license, translations or activity. However existing services are isolated from each other and thus only provide information about the projects using them. The freshmeat [69] website tries to tackle some of these issues by providing a central index for F/OSS projects enabling them to disseminate information about themselves, and the freshports [70] website lets people browse the entire FreeBSD [67] ports collection, it provides cross references, charts, graphs and link. However, these solutions only allow sparse interaction between projects and potential users. Indeed, users do not have access to information such as project needs in terms of contributor competencies.

F/OSS resources are also highly distributed. It is the case for technical resources as well as for community resources. For instance, contributors to the Mandriva Linux distribution come from multiple different countries. The Debian Linux distribution maintains a page indicating the different locations of their developers willing to provide this information [26] and Figure 2.2 was generated using the program xplanet [204] from the list of coordinates found on this page. In such a distributed context, it is difficult for F/OSS contributors to meet each other. In fact they rarely meet, know each other and communicate using the tools which are at their disposal, namely e-mails, mailing lists, wikis, IRC (Internet Relay Chat) [88] and other tools provided by projects such as conferencing tools like Skype [168]. As responsibilities are distributed among a distributed community, and as often contributors have no obligation to be involved in the projects they are contributing to, they can potentially stop contributing at any time. If this is the case, knowledge and experience can be lost, but also information about what has been done, what remains to be done as well as emerging ideas can be lost. Thus means facilitating communication, share and keep knowledge are essential in order to limit such a loss and to ease the replacement of contributors by other ones. Technical resources distribution is also high. Servers providing the different services highly depend on contributors providing them. For instance mirrors for distributing software can be located all over the world. While such a distribution is natural and wanted, it should be made as

seamless as possible. Issues concerning it such as server downtimes or unavailability, should be hidden to the community. Information integrity should be ensured by design in order to Similarly, the integration of new services to the pool of services provided by a project should be made easy. This is directly bound to the need of structures and resource description.

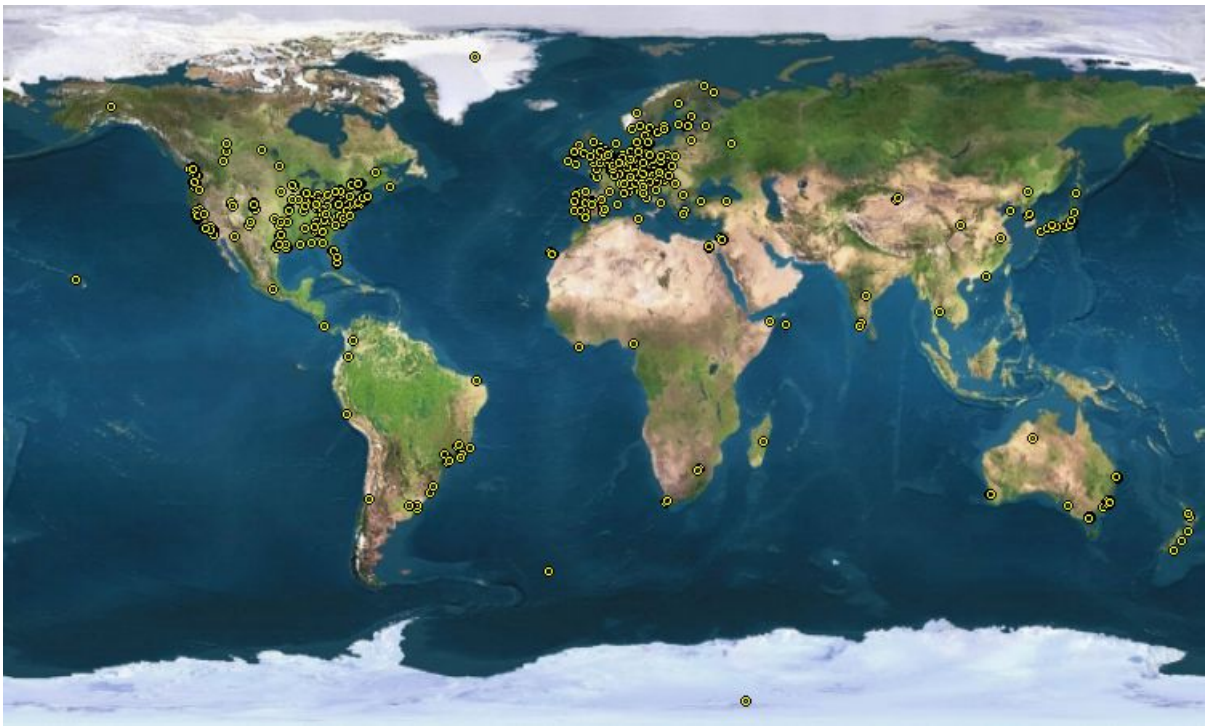

Figure 2.2: Map of Debian developers locations

Finally, even processes are distributed. Indeed a process such as the production of a distribution involves multiple subprocesses such as content production, content testing, defect management, packaging, then distribution. While some of them are executed in a context directly controlled by the project and are bound to a set of integrated tools, other parts such as some tests have to be run on the user side outside of the environment projects can control.

High Dynamism and Evolution. A crucial feature of F/OSS is the strong presence of evolution and the dynamism of projects, their content and their development. New projects are created by people all the time, and existing projects can take new directions. A project may decide to develop new applications, which also evolve to accommodate new functionality or bug fixes. The community involved in a particular F/OSS project evolves as well: developers join, some eventually get promoted to committers; committers can take time out to test packages, to cater for a new release, or may even leave the project in order to form a new one. F/OSS projects often start small with a few implied people and may become huge involving hundreds of participants.

Further this dynamism is also highlighted by the update activity of these projects. Unlike enterprise software where this dynamism is hidden as upgrades are provided through major versions, and critical bug fixes are provided on a punctual basis, the whole F/OSS philosophy is based on this dynamism. Indeed, the evolution of F/OSS is driven by the multiple updates, problem reports, bug fixes contributed by the community. While closed source software approach is to hide problems to the public, FOSS shows them instead hoping that someone will fix them as fast as possible. For instance the Debian project social contract stipulates that they "will not hide problems" [29]. This approach leads to very active projects.

The Eclipse project [34] provides in addition to its *stable build*, an *integration builds* integrating most recent changes to the project on a regular basis, *nightly builds* providing the changes made day after day. One of the most active projects on SourceForge, Gaim [71], had between november 2005 and november 2006, an average of 8 bugs opened by day and 9 bugs closed by day, and in August 2006, an average of 533 read transactions and 17 write transactions per day took place in Gaim's code base for an average of 101 files updated per day. Mandriva Linux 11 code base counted about 3500 modules of source code and represented an image of around 20 Giga bytes. Note that several millions users download each Mandriva Linux release.

This dynamism is part of F/OSS and provides precise indicators concerning the wealth of F/OSS projects [42], Thus information systems must be able to handle such dynamism by supporting it, and by informing the community of modifications occurring.

Anonymity Needs. F/OSS environment is often misunderstood as being completely anonymous as anybody can contribute to projects not identifying himself. While anonymity is wanted and required in some cases, full anonymity is not always authorized. Indeed anonymous access is often allowed to many services like ftp, or content management systems like subversion for reading purpose. However in case where code sources or binaries are subject to particular restriction implied by licenses, identification might be required, even for read access. In order to obtain write access, users must be identified for security and information integrity reasons.

As projects grow in size, the complexity of their code base grows and more anonymity restrictions can have to be set up. This is the especially the case for code bases containing sensitive information, such as operating systems' sources. For instance, in order to become a registered Debian developer, users have to pass through a process of identity, intentions and technical skills verification [28] . This process is partially run online through the evaluation of submitted patches, but to be completed, it involves that a Debian maintainer signs the GnuPG key of the new developer, which implies that people have to meet in person. Other projects have other ways of dealing with anonymity and have other requirements. For instance, FreeBSD has a disciplined group of core developers, this core team approves source code commiters, similarly the ports management group approves ports commiters, and for documentation the same process is applied. Once approved, new commiters are being assigned a mentor which works on the same code tree as them and which is responsible for everything their pupils do in the FreeBSD project. The role of the mentor is to answer questions, review submitted patches by the newcomers. If anything goes wrong there is no real penalty, but it reflects badly on both the mentor and the new committer and thus results in trust degradation in them. As time passes, the mentor verification becomes more loose, but he's still responsible for the commiter being assigned to him. The identification of committers is done through a username and a SSH key which are needed to identify on FreeBSD servers such as the CVS [24] code repository.

The means used to enforce identification are thus different depending on the projects. This implies that projects must be able to freely chose the anonymity approach they wish. Depending on projects' internal politics or even on licenses, different identification schemes must be applicable.

Chapter 3

Existing Approaches and Solutions

In this chapter we provide a the state of the art focusing on the different aspects of the F/OSS Process. Covered topics are the analysis of F/OSS community organization 3.1, F/OSS process management approaches 3.2, quality assessment 3.3, information display to ease project management 3.5, and existing solutions for ensuring F/OSS interoperability 3.6. We finally conclude in section 3.7 by analyzing this state of the art toward the stakes and challenges presented in section 2.3.

3.1 F/OSS Project and Community Organization

We have explored in chapter 2 the specificities of the F/OSS environment. One of these specificities concern the way the Open Source community is organized, the factors implying users involvement as well as its distribution itself. As most F/OSS projects rely on the work of volunteers, attracting people who contribute their time and technical skills is of paramount importance, both in technical and economic terms.

This reliance on volunteers leads to some fundamental management challenges: volunteer contributions are inherently difficult to predict, plan and manage, especially in the case of large projects. Due to the volunteer nature of most F/OSSprojects, it is difficult to rely on participants. Volunteers may become busy and neglect their duties. This may lead to a steady decrease of quality as work is not being carried out. The problem of inactive volunteers is intensified by the fact that most free software projects are distributed, which makes it hard to quickly identify volunteers who neglect their duties. During this time quality in the project can decrease as required functions are not carried out. The Debian approach to this issue has been described in [117]. This project uses different sources of information such as developers hints, email-based activity confirmation and a quality assurance team. Authors recommend to take preventive measures such as explain the problem to prospective volunteers, introducing maintainers redundancy to limit the damage of inactive maintainers and limit the number of low-interest packages.

An analysis of the evolution over time of the human resources in large F/OSS projects, using the Debian project as a case study, has provided an insight of how volunteer involvement affects released software, and the developer community itself [157]. The study has led to the following three conclusions: globally, the involvement duration of the volunteers is stable; experienced volunteers seek for responsibilities and increase their involvement over time; the voluntary effort is stable even when volunteers leave as tasks are taken over by other volunteers. However it has also highlighted a worrisome trend toward a growing number of packages per maintainer. This situation may imply scalability problems as the number of packages in the distribution increases if the project does not grow by a proportional number of developers.

This trend has been confirmed in [158]. Authors have studied the evolution of the stable versions of Debian from the year 1998 onwards. They highlighted the drastic evolution rate of the distribution which doubles in size (measured by number of packages or by lines of code) approximately every two

years. This result combined with the huge size of the system (about 200 MSLOC and 8,000 packages in 2005) may lead to significant management issues in the future. Indeed since the number of packages is growing linearly and since the specifics of each package imposes a limit on the number of packages per developer, this means that projects also need to grow in terms of developers at the same pace. However, such a growth is not easy, and causes problems of its own, specially in the area of coordination. This is something that has probably influenced the delays in the release process of the last stable versions of Debian. Therefor, emphasis should be but on both development and community management and organization.

Further, the motivation of contributors to F/OSS projects is central for project success [160]. This motivation is not just related to computer science interests, but also to economics, law, psychology and anthropology. Therefor, all factors that motivate volunteers to contribute to a project should be considered when reorganizing the structure of a project. Changing the internal structure or policy in an existing project, providing a new approach to the F/OSS Process can indeed have a strong impact on the success of projects and as such managing community resources must be as important as managing content resources to ensure F/OSS projects success.

In [162] the mechanisms sustaining developers' contribution to the community activities over time is identified. The paper tests the hypothesis that "Independently of developers' exogenous preferences, the more their exposure to the F/OSScommunity social environment, the more their contribution to the community activities". Employing data relative to 14,497 developers working on SourceForge.net during two years (2001-2002), a model testing the aforementioned hypothesis has been estimated making sure that the hypothesis is actually an empirically grounded result. The key result of this research is that, as the exposure to the F/OSS community social environment is able to foster developers' contribution beyond the level granted by their predetermined preferences, this leads directly to the evidence that the F/OSS community is provided with a mechanism sustaining and enhancing developers' incentives to produce and diffuse code.

The great disparity of principles and rules of social organization in F/OSS communities on the one hand, and of spirits and significance of belonging on the other hand have been highlighted in [30]. Authors present a paradox, Şdistant communityŤ, which raises questions about how to produce a whole when people are separated and about how to create cohesion over large distances. The stakes involved in solving these issues concern both the creation of cooperation and the creation of commitments. The paper highlights transversal mechanisms which are used to ensure control over both the work and the workers as well as processes shaping the career path of a F/OSS developer.

While F/OSS projects aim at gathering people around a common idea in order to reach a common goal and thus create their own community, some projects try to organize the different communities themselves. For instance, the SELF [166] IST project aims to provide a platform for the collaborative sharing and creation of open educational and training materials about Free Software and Open Standards. The first goal of the project is to provide information, educational and training materials on Free Software and Open Standards presented in different languages and forms. The second goal of SELF, is to offer a platform for the evaluation, adaptation, creation and translation of these materials. The production process of such materials is based on the organizational model of Wikipedia [198]. The SELF Platform specifically embraces universities, schools, training centers, Free Software communities, software companies, publishers and government bodies. As it is meant to be open, all users are encouraged to participate in the production process and the exploitation of results.

The Tightening knowledge sharing in distributed software communities by applying semantic technologies (TEAM) [181] IST project addresses the need for a knowledge sharing environment with advanced capabilities suitable for the distributed engineering and management of software systems. TEAM aims at developing an open-source software system, seamlessly integrated in a software development environment for enabling decentralized, personalized and context-aware knowledge sharing.

None of the projects and solutions presented in this section put emphasis on the tight relation be-

tween processes and the community involved in the process. However, the Unified Activity Management (UAM) [124] project defines an organizing framework for supporting collaborative work around the concept of human activity by creating a Unified Activity representation, architecture, and user experience. UAM meshes formal business processes with informal human collaborations to support business activities. An activity is any coherent set of actions taken toward some end, be it specific or vague. Activities vary from small single-person tasks to large-scale collaborative projects. Unified Activity is an explicit representation of Business Activity by descriptive meta-data (using RDF) that links together people, content, tools, events, and related activities, in a flexible semantic representation that brings together varied, expressions of activities and their associated materials.

These research projects funded by the European community show that is widely recognized that F/OSS projects' communities should be organized and managed. However, little effort is put in this task by F/OSSprojects. Some projects keep in touch with their community members through mailing lists, other choose more advanced means to tackle the community management issue through tools like wikis, and many small ones put no effort at all in this task. Indeed, projects mainly focus on core activities where community involvement is needed such as development, testing or translation.

A good example of what can currently be called "'advanced community management"' is Mandriva's mandrivaclub [111]. This club is a place built around a wiki where community members can find all documentation they need, forums, a shared knowledge base, and can chat with Mandriva team. However as neither users' motivation nor their interests or knowledge are considered by such a solution, advanced community management based on such information to help users contribute or help the project find skilled contributors is not possible. Thus we are far from effective and efficient community management.

3.2 F/OSS Process Management

While many program managers, project managers, and developers are already familiar, thanks to their experience or to the literature [187], with F/OSS process' strengths, the opportunities it provides and the ways it may be used in projects, there is no consensus on what the F/OSS process is made of. This leaves open the question of how to implement it and then how to manage it in a project environment.

An overview of how F/OSS projects are organized is presented in [46]. The paper explains related terminology and overviews main involved processes. The processes include decision-making within the project management, accountability of bugs to packages, communication among developers, generation of awareness about the project in the software community, managing source code, testing and release management.

The impact of Software process maturity on Free Software projects success has been studied in [118] through various statistical analysis on 40 successful and 40 unsuccessful randomly chosen projects. The results showed that the maturity of some processes are linked to the success of a project. This study identified the importance of the use of version control tools, effective communication through the deployment of mailing lists, and found several effective strategies related to testing. The identification of processes employed by successful free software projects is of substantial value to practitioners since they give an indication of which areas deserve attention. These results emphasized the importance of applying software engineering insights to the open source development model.

In [164] the authors highlight some issues arising in the modeling of techno-social process found in F/OSS development. They focus their work on the modeling of Apache Web server and Mozilla browser project processes. As previously existing descriptions of these processes where informal and narrative, they allow no analysis, visualization, computational enactment, reuse or comparison to be done. The authors use the *rich pictures* [123] method for discovering F/OSS processes to organize,associate observed development roles, tools and tasks. In order to provide a way to understand and perform these processes, the use of PML (Process Modeling Language) [130] is proposed. Among the benefits of modeling Open

Source processes authors cite the help models provide to new contributors to make them productive faster in the enactment of these processes. Another benefits include the enabling of continuous process improvement techniques, the provision of a coordination resource which can be used by distributed developers to synchronize their activities, roles and artifacts. Recent progress in the development of automated mechanisms to support and streamline the process discovery effort has been then described by the authors in [95].

ISO/IEC 15504 [81] provides a framework for the assessment of processes [80]. This framework can be used by organizations involved in planning, managing, monitoring, controlling, and improving the acquisition, supply, development, operation, evolution and support of products and services. It guides users on how to utilize a conformant process assessment within a process improvement program or for process capability determination. Within a process improvement context, process assessment provides a means of characterizing an organizational unit in terms of the capability of selected processes. Analysis of the output of a conformant process assessment against an organizational unit's business goals identifies strengths, weaknesses and risks related to the processes. This, in turn, can help determine whether the processes are effective in achieving business goals, and provide the drivers for making improvements.

Future directions in process improvement in the context of large and complex systems development have been explored in [193]. Among other conclusions, the auhors highlight the need for processes to follow sound management and quality principles. In particular, the people doing the work must plan their own work and that work must be precisely and continuously tracked. Everyone in the organization must be involved in and completely committed to the organizationŠs quality management program, and management must recognize and reward superior work. These conclusions can be applied to the F/OSS environment, however, there is currently no mean to have such a global view of the F/OSS process.

Process management is thus central to the success of projects. It relies on process discovery and notation. However while process management is being broadly adopted in the enterprise environment, its application in the F/OSS environment is sparse. In this section we explore different existing languages for process management then we present different project management approaches using these languages. Finally, we present how the distribution process is tackled in order to provide a better understanding of the role of process management in F/OSS.

3.2.1 Process Management Notations

Numerous process-related standard are available be it for defining the processes or executing them, be it from an internal or external perspective. Here follows some of them focusing on the following aspects of processes: their semantics, the definition of their interactions, and their notation.

Business Process Modeling Language (BPML) [12] is a meta-language for the modeling of business processes, like XML, is a meta-language for the modeling of business data. BPML provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional finitestate machine. It has been published by Business Process Management Initiative (BPMI) [11] as a standard providing a general approach to express business processes in organizations. The main components of BPML are: activities, processes, contexts, properties, and signals. Activities are elements performing particular functions. A process is a complex activity which can be called by other processes. A process independent with other processes is named a top-level process. A process executed within another process is a nested process. Contexts are essential in BPML. A context represents an environment that contents a sequence of related activities. It aims to exchange information and manage execution. A context includes local definitions that only apply within the area of that context. Local definitions contain properties, processes, and signals. Contexts can be nested and a child context (recursively) inherits the definitions of its parent contexts and may override them. A process can have a context which is used jointly by all activities that are included as part of that process. Information is exchanged by the properties and they can only exist within a context. A property definition has a

name and a type while each property instance has a value in the range of the type of the corresponding definition. People think of properties as attributes (or instance variables) of a process. Signals are used to manage the carrying out of activities executing within a common context other than through basic routing constructs such as sequences.

BPML was a rival language with other standards such as IBM's WSFL (Web Services Flow Language) [103] and Microsoft's XLANG [184] (Web Services for Business Process Design). WSFL is an XML language for the description of Web Services compositions. It considers two types of Web Services compositions. Flow models specify the appropriate usage pattern of a collection of Web Services, in such a way that the resulting composition describes how to achieve a particular business goal; typically, the result is a description of a business process. Global models specify the interaction pattern of a collection of Web Services; in this case, the result is a description of the overall partner interactions. XLANG [184] is an extension of WSDL, the Web Service Definition Language [203]. It is a notation for the specification of message exchange behavior among participating web services. XLANG provides both the model of an orchestration of services as well as collaboration contracts between orchestrations.

XLANG and WSFL combined into the Business Process Execution Language for Web Services language (BPEL4WS) [9] taking features from both the block-structured language XLANG and the graph-based language WSFL. Work on the business process language published in BPEL4WS and initiated by IBM, BEA, Microsoft, SAP and Siebel has been continued as an OASIS [148] (Organization for the Advancement of Structured Information Standards) specification [1] called Web Services Business Process Execution Language (WS-BPEL) [134]. WS-BPEL enables users to describe business process activities as Web services and define how they can be connected to accomplish specific tasks. Processes in WS-BPEL export and import functionality by using Web Service interfaces exclusively and can be described in two ways. Executable business processes model actual behavior of a participant in a business interaction. Abstract business processes are partially specified processes that are not intended to be executed. An Abstract Process may hide some of the required concrete operational details. Abstract Processes serve a descriptive role, with more than one possible use case, including observable behavior and process templating. WS-BPEL is meant to be used to model the behavior of both Executable and Abstract Processes. WS-BPEL provides a language for the specification of Executable and Abstract business processes. By doing so, it extends the Web Services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

The Web Service Choreography Interface (WSCI) [192] is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services. WSCI describes the dynamic interface of the Web Service participating in a given message exchange by means of reusing the operations defined for a static interface. WSCI works in conjunction with WSDL; it can, also, work with another service definition language that exhibits the same characteristics as WSDL. WSCI describes the observable behavior of a Web Service. This is expressed in terms of temporal and logical dependencies among the exchanged messages, featuring sequencing rules, correlation, exception handling, and transactions. WSCI also describes the collective message exchange among interacting Web Services, thus providing a global, message-oriented view of the interactions. WSCI does not address the definition and the implementation of the internal processes that actually drive the message exchange. Rather, the goal of WSCI is to describe the observable behavior of a Web Service by means of a message-flow oriented interface. This description enables developers, architects and tools to describe and compose a global view of the dynamic of the message exchange by understanding the interactions with the web service.

[1] OASIS is a consortium that drives the development, convergence, and adoption of e-business standards. It produces different Web services standards for security, e-business, and standardization efforts in the public sector and for application-specific markets

While BPML was extending process semantics, these were not fully supported by existing visual notations. Further, as the objectives of the BPMI are to support process management by both technical users and business users, the development of a notation that is intuitive to business users yet able to represent complex process semantics was needed. In order to avoid limited market acceptance of process management by end users a working group proposed that the industry should converge upon a standard business process notation: the standard Business Process Modeling Notation (BPMN) [13]. BPMN's goal is to provide businesses with the capability of understanding their internal business procedures in a graphical notation and to give organizations the ability to communicate these procedures in a standard manner. Furthermore, the graphical notation is meant to facilitate the understanding of the performance collaborations and business transactions between the organizations [131]. Both BPML and BPEL4WS languages can be mapped onto BPMN.

The XPDL specification provided by the Workflow Management Coalition (WfMC) [202] addresses the same modeling problem than the BPMN but from a different perspective. While, BPMN provides a graphical notation to facilitate human communication between business users and technical users, of complex business processes, XPDL provides an XML file format that can be used to interchange process models between tools. XPDL includes a common meta-model for describing the process definition and also a companion XML schema for the interchange of process definitions [195].

3.2.2 Process Management Engines

Multiple F/OSS Workflow Engines use the previously described process management languages and notations and offer support for project management.

JBoss jBPM [91] is a platform for multiple process languages supporting workflow, BPM, and process orchestration. jBPM supports two process languages: jPDL and BPEL. jPDL combines human task management with workflow process constructs that can be built in Java applications. JBPM enables the creation of business processes that coordinate between people, applications and services. The JBoss jBPM process designer graphically represents the business process steps in order to facilitate a strong link between the business analyst and the technical developer.

ObjectWeb Bonita [10] is a flexible cooperative workflow system, compliant to WfMC specifications for handing long-running, user-oriented workflows providing out of the box workflow functionalities to handle business processes. A comprehensive set of integrated graphical tools for performing different kind of actions such as process conception, definition, instantiation, control of processes, and interaction with the users and external applications. Bonita is a browser-based environment with Web Services integration that uses SOAP and XML Data binding technologies in order to encapsulate existing workflow business methods and publish them as a J2EE-based web services.

Apache Agila [4] is centered around Business Process Management, Workflow and Web Service Orchestration. It's composed of two specialized modules: Agila BPM and Agila BPEL. Agila BPM is basically handling tasks and users who have to complete these tasks. It's a very flexible and lightweight workflow component. Agila BPEL is a BPEL-compliant Web Services Orchestration solution.

The Apache Open For Business Project [7] is an open source enterprise automation software project providing an Open Source ERP, Open Source CRM, Open Source E-Business / E-Commerce, Open Source SCM, Open Source MRP, Open Source CMMS/EAM, and so on. Among other tools, OFBiz provides a Workflow Engine [206] which is based on the WfMC and OMG specifications and uses XPDL as its process definition language.

3.2.3 Production Management Tools

Apache Ant [5] is a Java-based build tool which evaluates a set of dependencies, then executes commands. Unlike make-like tools which are shell based, Ant is extended using Java classes and its configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run

by an object that implements a particular Task interface. As Ant is written in Java it provides system independence.

Maven [6] is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven was originally started as an attempt to simplify the build processes in the Jakarta Turbine project [8]. There were several projects each with their own Ant build files that were all slightly different and JARs were checked into CVS. A standard way to build the projects and a clear definition of what the project consisted of as well as an easy way to publish project information and a way to share JARs across several projects were the requirements which bootstrapped the Maven project. The result is a tool that can be used for building and managing any Java-based project. Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with: making the build process easy, providing a uniform build system, providing quality project information, providing guidelines for best practices development and allowing transparent migration to new features.

Amos [19], is EU funded project which aimed at making it easier to build software based on the composition of Open Source Code. One of the difficulties in this case is finding the right pieces of code, a task which usually requires deep knowledge of many software products. The idea behind Amos is to use high level descriptions of code assets, and perform a search using these descriptions to find the set of packages which best (according to some measure) fulfills a set of user requirements. The idea has been materialized in a tool, composed of a user interface, a database of descriptions, and a matching engine which interacts both with the user (through the interface) and with the database, where descriptions are stored.

3.2.4 Distribution Management

QSOS is a method for qualifying, comparing and selecting F/OSS in an objective, traceable and argued way [152]. It relies on interdependent and iterative steps aimed at generating software ID cards and evaluation sheets that support the selection of the best solution in a given context. This method defines frames of reference based on licenses, communities, functional grids which help evaluating software in terms of functional coverage and risk for both users and service providers. The selection and comparison of software is then done in the scope of an evaluation context. QSOS aims at improving software description and enable its selection. It is a method for collecting information then reusing it for selecting software.

The OpenSuse *build service* [146] provides a complete distribution development platform to create Linux distributions based on SUSE Linux. Its open interfaces allow external services to interact with the build service and use its resources. A server infrastructure hosts all sources, provides a build system to create packages, provides a download and mirror infrastructure for distributing packages and serves as the communication framework. Interaction with the build service can be done through an open API, a web interface or via command line. The OpenSuse service aims at connecting open source communities, provide a means to develop distributions while controlling issues like dependencies.

Red Hat Network (RHN) is an architecture whose design is also articulated around an API [201]. As for OpenSuse, it essentially focuses on code distribution. Indeed, RHN is used to download distribution ISO's, patches and software packages as well as to update systems based on user customization. The network is accessible through an *Access API*. The key abstraction RHN provides is the notion of *channels*, which corresponds to a set of packages. Every client machine that is connected to a specific channel can be updated when the content of the channel changes. A base channel corresponds to the core system and other types of channels are built on top of it. For instance, a development channel is used by developers to distribute their work. A Testing & QA channel is used to for bug reporting. The architecture defines actions for each channel. An example action could be to remove packages whenever

a new version is available, or to rollback to a previous version of the system when a compilation error occurs.

3.3 F/OSS Quality Assessment

The F/OSS community often considers that the F/OSS model guaranties a high level of quality by itself. However, this is far from being always true. Indeed, it relies on the efficiency of its organization. Although the basics of this organization are determined by the openness of sources, it relies of course on the community of developers and on its internal organization [94].

A general process for evaluating programs, with specific information on how to evaluate F/OSS programs is proposed by [196]. This process is designed to compare F/OSS and proprietary software side-by-side and determine which one (if any) best meets ones needs. This process is based on four steps: identify candidates, read existing reviews, compare the leading programs' basic attributes to your needs, and then analyze the top candidates in more depth. Important attributes to consider include functionality, cost, market share, support, maintenance, reliability, performance, scalability, usability, security, flexibility/customizability, interoperability, and legal/license issues. However, while enabling software comparison, this does not indicate if the software "winning" the comparison meets one's quality needs.

Thus, this raises the issue of what quality really is. Many people have an intuitive feeling of what quality entails, but when it comes to specify quality in a precise manner it turns out that the concept is very hard to define [25]. Quality is a concept which is made up of many different components [169] and the definition people can give of quality mainly depends on their perception of what quality is.

Exploratory interviews with F/OSS developers highlighting such components F/OSS quality is related to have been performed in [119]. While these interviews were restricted to a subset of the community involved in the products life cycle – and were thus omitting the final judges of products quality, namely the users –, this analysis provides some interesting results which can be used as a starting point for the implementation of quality process improvement strategies. First, the analysis results highlight the multiplicity and differences between the development practices and processes employed by interviewed projects. Further, while every interviewed person stressed the importance of processes, few projects follow all of them and most projects employ only some of the described processes. Further, actual quality problems have been identified. These include unsupported or orphaned code, configuration management issues, security updates management, lack of documentation, difficulty to attract volunteers as well as communication and coordination issues. Most of these quality problems are bound to a lack of efficient process management within the project. This shows that while developers may have the intuition that process management is important for quality assessment, they may not have the competences to achieve this task. This pin-points a potential need for process management specialists in F/OSS.

Quality Assessment is directly bound to process and metrics management. As the now abandoned ISO 8402 [76] norm defined it, quality assurance is all *planned and systematic activities* directed toward fulfilling the requirements for quality. Involved processes may influence the quality of a project. For instance, a bad activity coordination process can lead to latency issues which directly impact on the frequency of distribution releases and also on volunteers willingness to contribute as it can be considered as being too annoying. In order to study quality improvement, orchestrated processes involving tools and specialized metrics are needed to actually measure quality and evaluate quality improvement techniques. Such evaluation has been done in [120] through a number of statistical analysis based on over 7,000 defect reports which have been recorded about the free software project Debian. These analysis have been used to make observations on the effectiveness of this project. Gathering information about processes execution then orchestrating these processes is thus central for Quality Assessment.

In the reminder of this section, we explore a set of tools, and frameworks for Quality Assessment, highlighting their specificities and weaknesses regard to these two aspects. Then we present some effort

for integrating Quality Assessment.

3.3.1 QA Tools

Quality Assessment tools and frameworks are meant to organize tests, define their requirements and their goal, execute them, keep track of obtained results, and create test suites. Test suites aim at gathering tests concerning a specific domain in order to achieve testing coverage of this domain. However, not all existing tools provide all these features. Here follows a description of some F/OSS Quality Assessment tools.

Linux Test Project (LTP) [107] is an example of a test suite. It is a joint project started by SGI and maintained by IBM, whose goal is to deliver a collection of tools for testing the Linux kernel and related features to the open source community that validate the reliability, robustness, and stability of Linux. LTP also gathers test results, a Linux Test Tools Table providing the F/OSS community with a comprehensive list of the tools used for testing the various components of Linux and a code coverage analysis tool who's aim is to graphically identify the areas in the kernel impacted by the execution of these tests. This enables developers use the LTP test suite more effectively and also know the tests contributions that are needed to improve the test suite.

Bugzilla [15] Bug-Tracking System allows to track bugs and code changes, communicate with teammates, submit and review patches and manage quality assurance. It also ensures accountability. For each bug many information are provided, such as the bug reporter, the version of software, platform, priority, product, component, OS, severity, initial state, the person the bug has been assigned to, the persons that have to be informed of its creation, an url, a summary and description. Registered user can leave comments and propose patches. Bugzilla allows to search existing bugs by status, product, keywords, but one of its most interesting feature is its ability to generate precise and complete tabular or graphical reports. Indeed, the user is able to pick any data information existing in the bug database and use it as an element of the report. Examples of usable data are the summary of the bugs, the products that are concerned, the components of the products, their version, the comments, the url, keywords, bug status, resolution, severity, priority, concerned hardware, related software, email addresses and number of bugs and bug changes. The users define XYZ axis, puts filters to the different data then generates a report be it tabular, or graphical (bar, line or pie charts). Thus any information can be used as an indicator. Furthermore Bugzilla offers the capability to build bugs dependencies and to illustrate them using graphs or trees. Thus bug blocks can be indicated and graphically represented. Bugzilla has also a "voting" feature allowing users to vote for a bug. All users can be given a certain number of votes and when voting, an user indicates that the bug is of the highest importance and needs to be fixed rapidly. The number of available votes per user for a given product depends on how the administrator has configured the relevant product.

Testopia [183] is a test case management extension for Bugzilla. It is designed to be a generic tool for tracking test cases, allowing for testing organizations to integrate bug reporting with their test case run results. Though it is designed with software testing in mind, it can be used to track testing on virtually anything in the engineering process. Bug-Tracking Systems or Defect-Tracking Systems allow individual or groups of developers to keep track of existing bugs in a project and to help manage software development.

Fitness [52] is a lightweight, open-source framework aiming at enhancing collaboration in software development. It allows to collaboratively define Acceptance Tests through a wiki. Acceptance tests differ from unit tests in the sense that they define what the code should do from a functional and end users point of view. Indeed, while test units provide testing at a smaller granularity than acceptance tests do, and ensure that the code is correctly built, fitness test cases define business logic. They are readable by customers and they aim at ensuring that the application is doing what it is meant to. These two types of tests are complementary and both should be highlighted. Acceptance test are expressed as tables containing inputs and expected outputs. Tests are created through the wiki and can be directly run from

the framework. This is done by triggering so called custom "fixtures" which map customer language to the implementation. Created tests can be organized as test suites and can contain cross references to other test suites.

RTH [161] is a web-based tool designed to manage requirements, tests, test results, and defects throughout the application life cycle. The tool provides a structured approach to software testing and increases the visibility of the testing process by creating a common repository for all test assets including requirements, test cases, test plans, and test results. Regardless of their geographic location, rth allows testers, developers, business analysts, and managers to monitor applications. The tool includes modules for requirements management, test planning, test execution, defect tracking, and reporting.

Test Case Web (TCW) [180] is an online TCM system built with PHP and a SQL backend. It provides an efficient means for generation, organization, and execution reporting of test cases among projects and by multiple testers and versions. It provides various at-a-glance views of the test suite for easy status determination and test suite navigation. TCW also provides basic reporting capabilities and per-project access control.

Test Link [182] is a open source web based test management and test execution system. The tool enables quality assurance teams to create and manage their test cases as well as organize them into test plans. These test plans allow team members to execute test cases and track test results dynamically, generate reports, trace software requirements, prioritize and assign. The tool is based on PHP, MySQL, and includes several other open source tools. It integrates with bug tracking systems such as Bugzilla.

Salome-TMF [163] is a test management tool aiming to provide an open framework allowing, the automatic tests execution, the production of documents, and the management of defects/requirements. It uses the ISO 9646 [77] definition of tests. Tests can be manual or automatic. They are organized in suites and in sets of test suites called families. Test campaigns are sets of tests which will be executed with different datasets in different environments. For making test execution fully automatic, Salome-TMF integrates a script language based on Java, as one of several plugins which extend Salomé-TMF functionalities.

The Software Testing Automation Framework (STAF) [173] is an open source, multi-platform, multi-language framework designed around the idea of reusable components, called services (such as process invocation, resource management, logging, and monitoring). STAF provides an automation infrastructure, STAX, enabling users to focus on building their own automation solution. STAX execution engine automates the distribution, execution and results analysis of test cases. STAX also provides a powerful GUI monitoring application which allows testers to interact with their tests and monitor the progress of their jobs. The services for creating an end-to-end automation solution STAF provides include: EventManager, Cron, Email, HTTP, Namespace, NamedCounter, FSExt (File System EXTension), and Timer.

Open Office Automated GUI Testing Project [144, 177] provides a test framework with test scripts and an application (VCL TestTool) to test almost the whole Open Office [143] application automatically. The VCL TestTool scripts are written in BASIC with some additional functions especially for the office. The VCL TestTool communicates via TCP/IP communicates with the TCP/IP-Interface of Open Office and can test each installation of OpenOffice.org on a PC or in a local area network (LAN). The test tool offers to run tests (full, single stepping, procedure stepping), interrupt them, put break points, and browse errors. Result reports are produced in an hierarchical tree list format easy to analyze and browse.

GNU/Linux Desktop Testing Project (GNU/LDTP) [106] is aimed at producing high quality test automation framework and cutting-edge tools that can be used to test GNU/Linux Desktop and improve it. It uses the Accessibility libraries to access the application's user interface. The framework also has tools to record test-cases based on user-selection on the application. To test an application, GNU/LDTP core framework uses recorded test-cases and Appmap, a tool which goal is to control application startup in any UNIX environment. The status of each test-case is given as output. GNU/LDTP can test any GNOME application which are accessibility enabled, Mozilla, Openoffice.org, any Java application

(should have a UI based on swing) and KDE 4.0 applications based on QT 4.0.

Dogtail [32] is a GUI test tool and automation framework written in Python and released under GPL. It uses accessibility technologies to communicate with desktop applications. Dogtail scripts are written in Python and executed like any other Python program. Dogtail and LDTP share the same goals, however Dogtail mainly differs from LDTP as it uses dynamic discovery of accessibles at run time to identify desktop applications and widgets, while LDTP uses "Application Maps" generated beforehand for this task.

As the range of available open source testing tools is extremely wide, Opensourcetesting.org [145] provides users with a central access to them. Each tool is described by a profile and categorized accordingly to the family it belongs to. The website splits the tools in two main groups, testing tools and unit testing tools. Each of them is further sub-categorized: testing category contains tools for functional testing, performance testing, test management, bug tracking, link checking, security, while the unit testing category contains tools for creating unit tests in different languages such as in Ada, C/C++, HTML, Java, Javascript, .NET, Perl, PHP, Python, Ruby, SQL, Tcl or XML.

On the one hand, this broad choice of tools allows projects to deal with Quality Assessment as they want. For instance Mandriva has its own Quality Assurance Lab's Contributor's Corner [113] which aims at centralizing information about how quality assurance is dealt with at Mandriva, so that the community can participate in their public test campaigns and help them improve the quality of the Mandriva Linux distribution. The lab tests, validates and certifies Mandriva's own or integrated software and hardware compatibility. The tools it uses are Bugzilla for detect management, its companion for testing Testzilla as well as a database containing all Mandriva's RPMs providing information about them as well as dependencies, changelogs, etc. On the other hand, there is no standardization guiding projects in the choice of the tools for handling quality.

3.3.2 QA Integration

One of the goals of the EDOS [36] IST project is to research and experiment solutions which will ultimately allow to dramatically reduce the costs and delays of quality assurance in the process of building an industry grade custom GNU/Linux distribution or custom application comprising several. It will design, implement and experiment an integrated quality assurance framework based on code analysis and runtime tests, which operates at the system level. EDOS QSD [41] is a portal aiming at providing to the Linux distribution quality assurance actor a centralized view and control of the quality assurance process, focusing mainly on Linux distribution testing. The QSD portal functionalities include testing process reporting and control, testing platforms management and testing actors management.

The Software Quality Observatory for Open Source Software (SQO-OSS) [172] is a consortium of leading European Open Source projects, consultants and research institutions from Greece, the UK, Germany and Sweden that is developing a comprehensive suite of software quality assessment tools. These tools will enable the objective analysis and benchmarking of Open Source software. SQO-OSS goal is to assist European software developers in improving the quality of their code, and to remove one of the key barriers to entry for F/OSS software by providing scientific proof of its quality. The project plans to introduce a range of innovative software quality metrics and to combine them through the use of data mining and AI to provide clear quality evaluations for both software users and developers. The project aims at delivering a plug-in based quality assessment platform, featuring a web and IDE frontend, implement the proposed software metrics, publishing a league table of Open Source software applications, categorized by their quality. The SQO-OSS platform will be released under a BSD license to allow for further development by the F/OSS community and industry.

QUALOSS [154] is a starting IST project who will develop a high level methodology to benchmark the quality of open source software in order to ease the strategic decision of integrating adequate F/OSS components into software systems. The quality assessment methodology, unlike other existing methodologies, will combine data from software products (its source code, documentation, etc) with data about

the developer community supporting the software products in order to estimate the evolvability and robustness of the evaluated software products.

3.4 F/OSS Process Measurement

3.4.1 Measurement and Quality Models

Measures and measures selection

ISO provides a set of technical reports for the measurement of the quality of software products. The first international consensus on the terminology for the quality characteristics for software product evaluation has been released by ISO through the ISO 9126 standard. This standard is made of four different documents focusing on different aspects of software quality. The standard defines quality models [84] and external metrics [78] which indicate the quality from a point of view external to the software. These metrics are built on top of internal metrics [79] qualifying the quality from an internal perspective. Finaly, quality in use metrics [82] are used to qualify the software from a usage point of view. This standard proposes 6 different aspects of software quality (functionality, reliability, usability, efficiency, maintainability and portability) wich have 28 sub-characteristics and are used to classify the different metrics being used. ISO 9126 has been enhanced to coordinate it with the ISO 15939 standard [85] which provides an information model to help determining what has to be specified during measurement planning, performance and evaluation. The resulting standard is ISO 25000 [83] which is split in 5 divisions: quality model division (ISO 2501n), quality requirements division (ISO 2503n), quality management division (ISO 2504n), quality evaluation division (ISO 2505n) and quality measurement division (ISO 2502n) which aims at replacing the ISO 9126 standard. The quality measurement division is formed of five standards providing a measurement reference model and guide [86], measurement primitives [87], and dealing with internal quality measurement, external quality measurement and quality in use measurement.

The GQM (Goal, Question, Measure) method [189] relies on the fact that measurement to be useful and reliable has to be aligned with business objectives. Thus, this method eases the definition of what has to be measured and selection of related measures. The first step is to define a set of corporate, division and project business goals and associated measurement goals for productivity and quality. Then questions that define those goals as completely as possible in a quantifiable way have to be generated. The measures to be collected in order to be able to answer those questions and track process and product conformance to the goals have to be chosen. Mechanisms for data collection have then to be developed, data needs to be collected, validated and analyzed to provide feedback to projects for corrective action. Finally, the data has to be analyzed to assess conformance to the goals and to make recommendations for future improvements. Although the direct benefit of GQM is establishing meaningful metrics, it is particularly useful guiding and monitoring software processes, assessing new software engineering technologies, evaluating and certifying improvement activities.

F/oss quality assessment

There are not many widely spread metrics for the specific assessment of F/OSS and Linux distributions quality. The challenge that faces organizations considering a particular open source software product is that these products differ significantly from their commercial counterparts. Evaluating and then choosing a commercial package focuses on defining the right relationship with the chosen vendor: contract negotiations, price discounts, service level agreements (SLAs), maintenance and support commitments, and so forth. The processes that organizations use to succeed with commercial software are inapplicable to the new world of open source. However, some models and initiatives exist to help with such an evaluation.

The Business Readiness Rating (BRR) [14, 141] is being proposed as a new standard model for rating open source software. It is intended to enable the entire community (enterprise adopters and developers) to rate software in an open and standardized way. The calculation employed in the Business Readiness Rating model weights the factors that have proved to be most important for a successful deployment of open source software in specific settings. Among these one can find functionality, quality, performance, support, community size, security, and others. The Business Readiness Rating model is open and flexible, yet standardized. This allows for a broad implementation of a systematic and transparent assessment of both open source software and proprietary software.

Navica's Open Source Maturity Model (OSMM) [128] is designed to be a lightweight process that can evaluate an open source productŠs maturity in a short time (up to two weeks). It assesses the maturity level of all key product elements such as software, support, documentation, training, product integration and professional services. As output, OSMM provides a numeric score between 0 and 100 that may be compared against recommended levels for different purposes, which vary according to whether an organization is an early adopter or a pragmatic user of IT.

Capgemini's Open Source Maturity Model (OSMM) [59] describes how an Open Source product should be assessed to ensure that the product meets the IT challenges a company face. The OSMM accomplishes this by linking an extensive product analysis with a thorough review of the company and its IT issues.

The OSQ (Open Source Quality) [149] is an active Berkeley University project that investigates techniques and tools for assuring software quality: finding and removing defects in software systems, as well as improving current methodology for designing high-quality software systems at the outset. The project consists of both experimental and theoretical components. The project does not depict the metrics used and concentrate in the software tools and requirements

F/oss process quality

The EDOS project [36] aims at developing technology and products to improve the efficiency of two key F/OSS processes: the generation of a new version of a distribution from the previous version and the production of a customized distribution from an existing one. To achieve this goal, EDOS has defined a set of metrics split in different groups (dependency management, production, quality assurance, content dissemination, package use) to measure the efficiency of the processes in question [42]. These metrics have been then implemented in a mirror monitoring tool [43]. For displaying the computed metrics, a portal has been set up for displaying the metrics issued by EDOS tools. Two distinct portals have been set up, one for each distribution editor involved in the project: Mandriva [39] and Caixa Mágica [37]. Both portals run a common XWiki application consisting of an object data model and a set of Groovy scripts that manipulate that data model. The portal includes a package browser who lets the user view package specific information while consulting the dependency metrics. The quality assurance part displays submitted test reports, as well as the associated metrics. The download channels section aims at displaying content dissemination metrics. And a dashboard lets the user create a customized dashboard comprising a subset of the available indicators [44]. This portal has helped measuring the improvement of the dissemination process introduced by the replacement of the traditional mirror approach by a P2P one. Recently, a EDOS portal has been set up for the Debian Linux project [38]. To represent indicators pertaining to an open-source project in a reusable way, the EDOS Project issued two RDF schemas. MOAP stands for "Metrics Of A Project" and may be useful for measuring the pulse of any F/OSS project and for creating dashboards, while MOALD stands for "Metrics Of A Linux Distribution" and focuses on metrics related to dependency management and P2P dissemination. The EDOS portal relies on these vocabularies.

3.4.2 F/oss Model Measurement

Relatively easy accessibility of high volumes of information about open source software makes it an interesting target for quantitative analysis meant to discover some hidden properties and trends of this software development model. FLOSSMole [89, 57] (Free/Libre Open Source Software Mole and formerly OSSMole) is a mining project which aims to provide data and reports about existing F/OSS projects and teams. A particularity of FLOSSMole is that it promotes compatibility both across sources of F/OSS data and across research groups and analyzes. The project gathers, shares and stores comparable data and analyzes of F/OSS development for academic research. Having such a collaborative and compatible data and analysis repository enables reproducible, extendable and comparable research on F/OSS. The project provides scripts for analyzing the raw data and provides some tools enabling users to gather their own data. The raw data used by FLOSSMole is donated from other research teams and projects to create common frames of communication. Collected data includes page views, downloads, bandwidth consumed by downloading, and number of comments posted.

The way hidden properties and trends of F/OSS can be acquired have been described in [194]. This research analyzed the largest open source hosting facility Ů SourceForge - to obtain quantitative information about existing projects. It focused on aspects such as project activity, average number of developers per project, number of language translations per project. Cross-comparisons were done with the results provided by FLOSSMole on the following aspects: number of developers, intended audience, usage of licenses, targeted operating systems, language of implementation, declared development status, registration history and declared topics. Among other conclusions this analysis showed that the main target audience for F/OSS projects are the developers. While fostering developers involvement, this leaves appart the question of growing the community of users. Further, the analysis showed that the GPL licensing type is the most common in the F/OSS world. A last interesting conclusion of this analysis is that a vast majority of open source projects declares their development status at the ŚunstableŠ level or even in the planning phase. As only minority of F/OSS projects ever makes it to the level of being popular (and thus successful) this leads to further questions concerning the reasons which make a F/OSS project successful or not.

In [159] authors use publicly available storages (source code snapshots, CVS repositories, etc), as a source for analyzing and characterizing the evolution of F/OSS project. Since the base information is public, and the tools used are libre and readily available, other groups can easily reproduce and review the results. Obtained characterization is then used as the basis for qualitative analysis including correlations and comparative studies of projects. The proposed approach is shown, as an example, applied to Mono, a F/OSS project implementing parts of the .NET framework.

Such a use of publicly available data is also made by different tools for analyzing the F/OSS development process. For instance, CVSAnalY [3] is a tool that extracts and manages statistical information out of the activity that happens in a control version repository such as CVS and most recently Subversion. It parses repository logs and transforms them in either information exchange XML and database SQL formats. It has a web interface - called CVSAnalYweb - where the results can be retrieved and analyzed in an easy way. Another example of such an analysis tool is DrJones [45]. It is a software that allows to perform a software archeology analysis on software that is stored in a CVS or Subversion versionning repository. DrJones analyzes how old the software system is on a per-line basis and extracts figures and indexes that make it possible to identify how ‘old’ the software is, how much it has been maintained and how much effort it may suppose to maintain it in the future. Dr. Jones counts the SLOC, the number of files, the number of authors and it gathers file timestamps to evaluate occurred changes. A list of other tools used to extract information about the F/OSS development process are listed and made available on the Libresoft [104] website.

3.4.3 Collaborative intelligence

Ohloh.net [136] is a resource for open source intelligence on thousands of open source projects. Ohloh collects software metrics from a variety of sources including the project's source code and the software development infrastructure used by the project's development team. It provides information about the developers involved in the project, showing details about their activity (i.e. the frequency on contributions), the languages used by the project, and the licenses under which the source code is released. A codebase history shows the evolution of the source code of a project. It specifically shows the total size of a project's source code over time. This graph reveals at a glance how long the project has been around, and the relative pace of development over time. It's generally a good sign to see sustained, constant activity over a long period of time. This means that people are continually updating it (fixing bugs and/or improving features), and that the project has staying power.

The main objective of the FLOSSMetrics (Free/Libre Open Source Software Metrics) Project is to construct, publish and analyze a large scale database with information and metrics about F/OSS development coming from several thousands of software projects, using existing methodologies, and tools already developed [56]. The targets of the projects are to Identify and evaluate sources of data and develop a comprehensive database structure, built upon the results of the CALIBRE Project [18].

3.5 Information display through dashboards

In today's information society, F/OSS projects, as many enterprises and organizations, maintain massive data repositories. The volume and the complexity of data make their analysis and usage a time-consuming task while it is difficult to focus on what is really important. Project managers are often lacking tools to efficiently monitor the evolution of a project in real time or tools presenting the status of the project. Such an issue not only affects productivity, but also efficiency, accuracy, and quality as taking decisions in such a context becomes cumbersome.

To solve this kind of deficiencies *dashboard* tools have been developed. Multiple definitions have been proposed in the past. For instance, in [190], dashboards are considered as personalized, adaptive, graphical displays of tactical metrics, whereas in [125], a dashboard is defined as a concise, context-specific display of key metrics for quick evaluation of multiple subsystems. In [33], dashboards are defined as *multilayered performance management systems, built on a business intelligence and data integration infrastructure, that enable organizations to measure, monitor, and manage business activity using both financial and non-financial measures*. Stephen Few proposed a less restrictive definition in [51] after having searched common characteristics of different dashboards. As a result, he defined a dashboard as *A visual display of the most important information needed to achieve one or more objectives which fits entirely on a single computer screen so it can be monitored at a glance.*

Today, many different dashboard solutions exist on the market. Most of them are proprietary software, however some F/OSS solutions exist, like Pentaho [150]. There are mainly three types of providers. Business Intelligence (BI) vendors such as Business Objects, Cognos and MicroStrategy provide broad suites which include dashboards but also handle data integration, reporting, query and analysis, performance and information management. Vendors specialised in reporting and/or analysis such as Principa, Hyperion and Advizor provide dashboard solutions in addition to their core applications. Niche players, such as Celequest, arcplan, Theoris, iDashboards and QPR Software to name a few, have developed robust dashboard solutions which offer various interesting characteristics and features. Pentaho provides a complete BI platform that includes reporting, analysis, data mining, data integration and dashboards. Its components are released under the Mozilla Public License.

Dashboards are designed to monitor many types of data and support all kinds of business activities that might benefit from an overview of . We can distinguish three major types of dashboards—strategic, analytical (or tactical), and operational—each focused on different problems or functional areas [51],

[33].

Strategic dashboards. Strategic dashboards focus on high-level measures of performance and often include forecasts, comparisons to targets and brief histories. They compile data mainly from data warehouses and data marts in a personalized manner, tailored to the specific needs of their users. As they are intended for organisation's executives and managers, strategic dashboards provide the information needed by decision-makers to monitor the health and opportunities of their business. Among other essential functions, strategic dashboards have to offer a coherent vision with regard to strategic objectives, measure the performance and progress towards goals, and signal dysfunctions [50]. They act as a common reference which simplifies communication between actors.

Analytical dashboards. Another type of dashboards is Analytical Dashboards. Their purpose is to inform analysts of potential areas of interest that require examination. They highlight relationships between information and help identify potential problems, helping analysts uncover latent trends and make efficient recommendations. Compared to strategic dashboards, analytical dashboards must be able to provide rich comparisons, extensive history and more subtle performance evaluators. Analytical dashboard usually rely on OLAP data sources, data warehouses and data marts and rarely require real-time data. They support rich interaction and incorporate sophisticated analysis and data mining tools or directly communicate with such tools.

Operational dashboards. Operational environments require more dynamic dashboards than Analytical and Strategic ones. In such environments, activities and events are constantly changing and might require attention or immediate response at any moment. The purpose of operational dashboards is to maintain awareness of such activities and events. Typical examples of activities that can benefit from operational dashboards include the monitoring of production processes, of computing resources and of financial markets. Operational dashboards usually require real-time or near real-time data often pulled from transactional systems. Compared to the previously seen dashboard types, the information that appears on operational dashboards is often more specific, providing a deeper level of detail through drill-down capabilities.

F/OSS projects often argue that they provide a dashboards. In reality, analytical and strategic capabilities are rare not to say inexistent. In most cases only operational dashboards are provided. They provide for instance information about the number of contributors to a F/OSS project, the activity of these contributors, the frequency of releases, the number of problem reports, the number of exchanged messages, etc. with different possibilities to classify this information. The information communicated through them is rarely sufficient to detect potential issues, detect causes having led to these observations and take decisions to adapt the strategy of a project accordingly. Relationships between observed situations are difficult to highlight as operational dashboards are not designed for this. F/OSS projects could benefit from analytical and strategic dashboards, however this implies that access to related information is available.

3.6 F/oss Interoperability

Interoperability is the ability for a system or a product to work with other systems or products without special effort of the part of the user or as the ability of Software and Applications to interact. Increasingly, enterprises are cooperating with other enterprises. Nowadays, competitiveness is largely determined by the ability to seamlessly interoperate with others. Eruropean networks of excellence and projects such as INTEROP [75] and ATHENA [75] tackle this issue at the enterprise level. Not only large organizations and SMEs set up cooperation agreements with other enterprises, but also F/OSS projects

are combining forces. As such, F/OSS process improvement is directly bound to the ability of projects to interoperate. Thus this issue also need to be considered in the F/OSS context.

Interoperability is considered to be achieved if the interaction can, at least, take place at three levels: data, application and business enterprise. First, technical Interoperability is needed to exchange data between applications or projects and can be achieved through the use of common communication means and languages such as XML or RDF and open standards, semantic Interoperability is needed to make applications understand exchanged data, this can be achieved through specific ontologies, finally organizational Interoperability deals with the issue of connecting and making interoperable organizations having made their own business, technical and ontological choices. While the technical interoperability issue is currently mostly well known, interoperability research work is mostly related to the seek for semantic interoperability, but organizational interoperability is simply left apart.

The Flink group [53] has defined a Linux ontology and aims at demonstrating the benefits that result from formalizing knowledge about the Linux operating system. In [73], the authors discuss the application of ontology-based knowledge engineering to Linux. Various possible applications such as package management or information search are discussed which would all benefit from a comprehensive ontology of the domain. The use of an ontology to describe Linux is similar to our approach as it provides a common ground of understanding.

The Friend of a Friend (FOAF) [58] project is creating a Web of machine-readable pages describing people, the links between them and the things they create and do. It is a community driven effort to define an RDF vocabulary for expressing metadata about people, and their interests, relationships and activities. FOAF is tackling head-on the wider Semantic Web goal of creating a machine processable web of data. It facilitates the creation of the Semantic Web equivalent of the archetypal personal homepage: My name is Leigh, this is a picture of me, I'm interested in XML, and here are some links to my friends. And just like the HTML version, FOAF documents can be linked together to form a web of data, with well-defined semantics.

DOAP [31] is a project to create an XML/RDF vocabulary to describe open source projects. This project aims at providing an internationalizable description of a software project and its associated resources, including participants and Web resources as RDF schemas. The project provides basic tools to enable the easy creation and consumption of such descriptions in all the popular programming languages and ensures interoperability with other popular Web metadata projects (RSS, FOAF, Dublin Core). Finally, DOAP vocabulary is extensible for specialist purposes. Note that DOAP does not aims at handling software releases, nor at planning data internal to the project such as task assignments or milestones.

QualiPSo [153] is a recently started European IP project working on open source process improvement. It's goal is to investigate and implement development processes through an open forge, including business models, methods and tools to foster the wide adoption of Open Source Software by European organizations from ITC players to end users. It focuses on aspects such as license management, documentation and information management, and interoperability. Unlike other projects focusing on the technical or semantic aspect of interoperability, QualiPSo aims at handling all the aspects of interoperability mentioned above. It will provide a model for information contained in F/OSS including elements such as project, participants, tasks and planning, requirements, bugs, documents, new functionality proposals, source code, project versions, mail, forums, meetings, source code management.

3.7 Conclusion: A fragmented World

As seen in this chapter, organization, process management, interoperability and information display are hot topics. As they all contribute to process improvement, they have thus to be considered in the F/OSS context. Nevertheless, none of existing approaches and tools provides consistent support for projects to enable F/OSS process streamlining. None considers the F/OSS Process as a whole highlighting how activities are interwoven. Further, the community management issue is globally tackled poorly.

Fragments of solutions exist to tackle specific issues. However, these attempts to handle F/OSS subprocesses are not integrated in a global view, and some, such as project metrics management, still need to be tackled. In fact, most issues involving interaction between F/OSS activities, or involving information exchange are unaddressed.

From the tool perspective, the same issue can be highlighted. A large number solutions aim at tackling issues related to specific activities of the F/OSS process but locally without being integrated in a large picture. Let cite, for instance, defect management tools (Bugzilla [15]), testing tools (Bugzilla test runner [16], Fitnesse [52], Salome [163]) and other tools (SourceForge) [170], GForge [72], Alioth [2], Libresource [105]) providing an integrated solutions for managing F/OSS projects. The dashboards provided by some projects [35, 68] are a good symptom of this issue as they focus on specific information hardly usable.

We have seen that F/OSS tools for process and workflow management exist, however these tools are often designed for the enterprise environment, and are not considering the specific stacks and requirements of the F/OSS environment we explored in Section 2.4 and Section 2.3. Further these tools imply a machinery which while being mandatory in the enterprise context and very useful in both enterprise and F/OSS context, might be difficult to implement in an open environment.

In such a context we can argue that the issue of F/OSS process improvement is left open. A transversal approach to the process management issue is required. It has to offer means to handle project as a whole considering involved activities, their content, community, and allowing reasoning about F/OSS processes, tasks, metrics, etc. in a distributed context in order to enable F/OSS process streamlining.

Part II

Model

Chapter 4

Model Requirements

This chapter defines the requirements of a model for describing and using Free and Open Source resources and managing the underlying process. The purpose of this model is to provide the means necessary to design Information Systems able to streamline the F/OSS process. These requirements can be split in two groups. Information management requirements are introduced in Section 4.1 and Process Management requirements are presented in Section 4.2. The last part of this Chapter, Section 4.3, lists selected elements of solution.

4.1 Information Management

The first set of requirements is related to the management of information and thus of F/OSS related data. The model has to ensure the following properties.

Information Gathering and Access. The key requirement for F/OSS information management is the ability to gather information and make it available to the community needing it. The solution has thus to ensure the existence of needed information and that information generated by the execution of the F/OSS process is kept for future ussage.

Information Description Uniformity The solution has to describe uniformly both content and community or any other F/OSS artifact. Such artifacts represent any resource that is related to the F/OSS environment and which is thus potentially used within the F/OSS process.

Information Integrity The solution should enforce information integrity at two different levels. First, artifact structure must be clearly defined, indicating mandatory information. For instance, a code package cannot exist without a set of dependencies and a license attribute. Second, possible usages of these artifacts must be clearly defined and the solution has to enable the definition of integrity rules bound to artifact usage. For instance, prior to installing a code patch, all previous patches have to be installed.

Lookup Flexibility End-users have to be able to locate artifacts based on their properties. For instance, code units should be searchable in terms of functionalities they provide, their platform requirements or the license being used. Similarly, one should be able to locate other members by specifying the competences or topics of interest.

4.2 Process management

The second set of requirements is related to the management of the F/OSS.

Activity Description . Similar activities are often used by different projects. The details concerning these activities are rarely available, rarely clearly described and are even more rarely easily obtainable. Therefor, it is difficult to know what are the different behaviors each activity should provide and which information it should provide in order to ensure the correct and efficient execution of the activity itself, but also respect to the integration of the activity with other activities. In order to enable information sharing among activities and thus activity de-fragmentation, the solution has to provide a way for projects to clearly describe their activities.

Cross-activity coordination . The solution must support interactions between activities. For instance, defect reporting entails binding a defect report to code; this report can be localized by participants of a debugging activity based on its properties. These interactions do not only occur within projects but also between projects. For instance, a new product release issued by a project can trigger a set of processes in other projects.

Process Description Another requirement is related to the description, or modeling, of the processes of a project. While F/OSS processes are often loosely declared, they should be clearly described for multiple reasons. The straightforward reason is to ease the introduction of new participants in the community of a project. Having processes which are clearly described, and which highlight the different steps involved along with required competencies, is mandatory for fast comprehension of project processes and efficient integration of new community members. Such process description also enables efficient process analysis. Having clearly defined processes with indication how they involve the community, and how these processes are chained enables problems detection, which can lead to process comparison and modification.

Process Streamlining Processes should be chain-able in order to express the F/OSS process of projects and thus indicate how the different activities of projects integrate and how the different resources are related. The goal being to provide a global view of each F/OSS project.

Measurability In order to support the analysis of F/OSS Information Systems and of related processes, the solution has to provide means to measure these processes, events occurring in the system, and community behavior. Such measurement has to be possible despite of the fragmented nature of the F/OSS process. The solution must enable transversal measures involving multiple activities and potentially involving multiple projects. As in the F/OSS context activities are distributed and as modifying other projects to implement new metrics is not possible, the solution has to provide means to declare transversal metrics. Finally, to match the F/OSS ideology, measures must be sharable among community members, retrievable and reusable.

Extensibility As resources types involved in F/OSS process activities are potentially unlimited, the set of artifacts provided by the solution cannot be closed. Therefor, the solution has to provide a generic means to the F/OSS community to declare new artifacts. User defined artifacts have to benefit of the same properties the core artifacts provided by the solution do. While some core activities common to all F/OSS Information Systems have to be provided, the solution has to be generic enough to allow the declaration of new activities and processes and integrate them with the rest of the project.

4.3 Elements of solution

F/OSS integration . To support virtual collaborative processes in the F/OSS environment, the solution should highlight and integrate in a common model following different elements:

- F/OSS *resources* are consumed and produced by the process. These are code units, defect reports, etc. as well as community members since their competence gets harnessed to produce software.
- *Activities* are mainly interfaces for exploiting resources. They define the different means available to manipulate resources through operations. New activities can be defined during the lifetime of a project.
- *Processes* The set of processes defined for a project represents the set of all possible actions in the scope of this project. A process is the description of a set of activity operations to be called in a given sequence to reach a particular goal. A process can involve multiple operations provided by multiple activities. Each process is bound to requirements in terms of competencies, knowledge or interests.
- *Roles* provide an organizational structure for people working on f/oss projects. They bind processes to F/OSS contributors. Privileges given to actors for executing processes and manipulating resources depend on the different roles assigned to them.
- *Tasks* are the assignment of the execution of a process to a community member. The responsibilities of community members are reflected in the tasks that have been assigned to them.

Descriptive approach: Artifacts and Attributes . In order to make understandable F/OSS Information Systems, the resources they are using, the activities and processes which are involved, a means to decouple the description of such Information Systems from their implementations is required. An important aspect of the proposed approach is to enable the definition of Artifacts and then bind *attributes* to them.

An artifact is an envelop describing a F/OSS resources ensuring that all information needed to describe and thus use this resource is available. Attributes represent an ontology describing Artifacts with other Artifacts. For instance, they help identify software needs or wishes that end-users may have. Indeed, the natural way for defining applications to install and to use is to specify the category it belongs to as well as their properties, e.g., *a text processor able to include some multimedia files like pictures taken with a Samsung V349 camera*. Nevertheless, in existing approaches this abstraction of attribute is artificially added through categories and keywords on top of a distribution model mainly based on filenames and versions (such as packages). Attributes can also include structured information to support other activities. For instance, *patch for* and *upgrade to* attributes can be incorporated to ease dependency management or defect tracking activities. The aim is to permit the localization of resources in an associative manner, i.e., based on properties of these resources rather than simply using names.

Substitutability . In order to be flexible, artifact look-up supports the notion of *substitutability*. The idea is that if a look-up operation does not find the required artifact, then it will return artifacts that can be used in place of the required artifacts. For instance, two different licenses may have equivalent IPR implications. Thus, a look-up for a license may return the other license. In the context of contributors enrollment, a contributor can be substitutable by another one for a given task if they both have required competences and if they are both interested in the task to be achieved.

Process logging . A key aspect of Information Systems is to keep information related to a given context. In order to automate such a feature, when executed each step of a process is logged along with information providing the context of the execution. This context includes information such as the entity executing the process, the project in whose scope the process is executed, a timestamp, the action which is made, the process itself, and all other needed information which has been provided to execute the process step.

Process measurement flexibility . An important element in any process is feedback. This is needed to understand if new resources or activities are needed, or just to understand if the process is underperforming. Metrics are used by dashboards to display information about projects and are thus the meat for process analysis. As such they should not be static, they need to be continuously adapted to the evolution of users requirements. We argue that metrics should be considered as a F/OSS artifact that can be reused, combined in the definition of new metrics and evaluated through a single interface which gives access to all metrics in a unified manner.

Extensibility The diversity and high dynamics of the F/OSS environment imply the need for great adaptability. The proposed solution should thus enable the evolution of the different F/OSS artifacts through the creation of new resources, new activities, new processes and new measures.

These elements of solution are integrated in a model known as the *Process Reference Model* (PRM) model. The PRM model enables F/OSS project management and information systems description through the declaration of artifacts, declaration activities, their combination through processes and measurement through metrics.

Chapter 5

F/oss Process Reference Model

This Chapter presents a process reference model (PRM) for modeling the activities, roles and resources of the F/OSS process. This model allows F/OSS activities to be more efficiently handled, compared and retrieved. The goal of the PRM is to define the key content and community artifacts of the F/OSS process and to formalize the relations between these. We formalize the proposed model in section 5.1 then discuss its properties in section 5.2.

5.1 FOSS-PRM Model

Figure 5.1 depicts the different layers of the core of the PRM model and show how the data structures relate to the different elements it provides. The role of the bottom layer provides the data structures as well as the mechanics to handle them. The second layer of the model aims at modeling F/OSS. It provides a set of elements for describing projects, the community associated with them, the different resources which are involved in the projects, related integrity rules which have to be enforced, available activities handled by these projects and the processes each project runs along with the requirements and rights involved in the execution of these processes. The third layer ensures the execution of described processes. It enables the assignment of processes to community members trough tasks and coordination of processes through events. Finally the top layer provides a means for achieving process-wide analysis through the definition and execution of metrics and through the logging of the PRM activity.

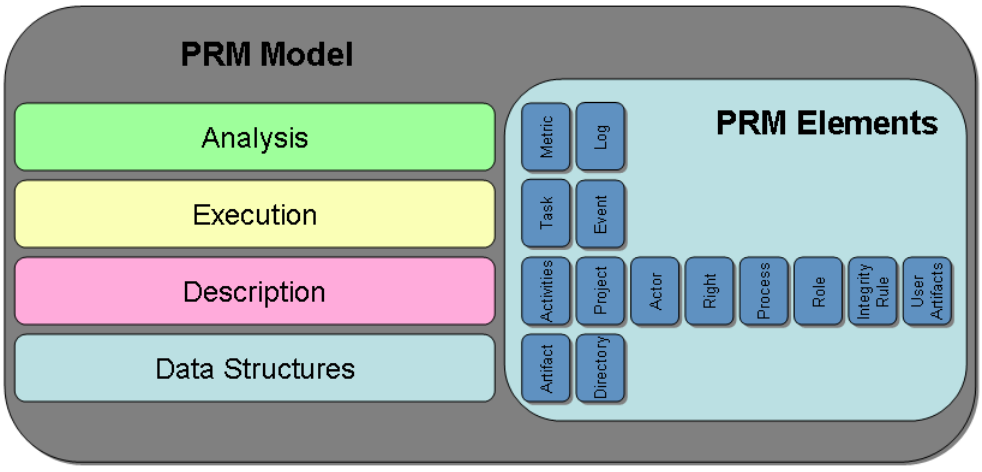

Figure 5.1: PRM Layers Overview

We start by presenting how the model structures data through the concept of Artifacts before providing an overview of the different PRM Artifacts. To conclude this section, we describe the main operations of the model.

5.1.1 Artifacts and Attributes

As mentioned, an Artifact is any F/OSS resource. Artifacts are typed; the type defines the operations applicable as well as possible constraints on the artifact during its life-cycle. Each Artifact a_i belongs to the Artifact set $\mathcal{A}_i$. All $\mathcal{A}_i$ sets are disjoint, and therefore non-comparable, reflecting the fact that there is no sense in comparing information such as a License with a Test. New Artifact types can be defined through the PRM.

$$\forall i,j,\ i \neq j.\quad A_i \cap A_j = \varnothing$$

Table 5.1 provides a description of the different key Artifacts involved in the f/oss Process.

Attributes

Artifacts can either be *Atomic* or *Compound*. Atomic denotes a single resource; Compound denotes an Artifact composed of other Artifacts which are then called its *Attributes*. Artifacts can be combined into Artifact expressions and directories (denoting a collection of attributes of different types, see Section 5.1.1). These are useful for locating artifacts based on a combination of different attributes. Thus the set of attributes of any artifact a_i is noted $a_{i_{\mathcal{A}}}$. An attribute $attribute$ of a_i can be retrieved using the attr operation as follows: $\mathsf{attr}_{attribute}(a_i)$.

Each Artifact type A_i has an associated algebraic structure $(A_i, \simeq)$. The $\simeq$ operator defines a *substitutability* relation on A_i. This relation is used to express a loose or semantic equality which indicates that it is legitimate to use an Artifact in place of another one. For instance, a functionality can be used in place of another enclosing functionality; a license can be used in place of compatible licenses, etc. The expression $a_1 \simeq a_2$ captures the fact that a_1 can be used in place of a_2. The properties of $\simeq$ depend on the type of A_i, e.g., whether it defines a partition or partial order on A_i, or neither or more complex computations being context dependent. The substitutability relation eases the definition of checking algorithms.

The $\simeq$ algebraic structure has to be defined for each Artifact. Table 5.2 defines possible relations and provides examples for each of them.

Artifact Expressions

In order to provide flexibility when searching for Artifacts, the model provides an expression language for specifying Artifacts in the *lookup* operation. For instance, when searching for content, one can specify that it:

1. Comes from any mirror server (s_1, s_2, ..., s_m) but not from an unofficial mirror server (s_{m+1}, s_{m+2}, ..., s_n);
2. May not use license l;
3. Must have functionality f_1 and f_2.
4. Version v must be equal to or newer than version 2.
5. Any content will do.

An attribute expression e is built from $\wedge$ (*and*), $\vee$ (*or*) operators and can be negated using $\overline{overline}$.

Artifact	Description
Actor	F/OSS Community member taking part in diverse activities of the F/OSS Process
Activity	List of possible operations related to a specific project activity, such as Testing, Production, Community Management
ContactInformation	Informaiton about an Actor such as his name, url and email
Date	Simple date using any format
Email	E-mail address
Event	Artifact used for asynchronous communication and synchronization of processes or tasks
EventType	Type of the Event
IntegrityRule	Rule to be enforced for ensuring the integrity of Process execution
IntegrityRuleExpression	First class object expressing a rule to be enforced.
Log	Capture of Actor behavior
Metric	Reusable user-defined measure of any aspect of the F/OSS Process
MetricOccurence	Information concerning when a measure has to be taken
MetricSet	Set defining a logical relation between different measurements
MetricViewPoint	General context in which a measure is done
Number	Numbers
Process	orchestration of F/OSS operations, possibly taking up time, expertise or other resource, which produces some outcome
Project	Umbrella indicating F/OSS Processes, Actors and Roles involved in the life cycle of a set of
ProcessCategory	The category the Process belongs to
ProcessType	Type of the Process, can be *once-off* or *recurring*
Right	Allowed Action in the scope of a Task
Role	Responsibility within the F/OSS process expressed as a collection of Processes
Task	Assignment of a Process to an Actor in the scope of a Project and Role
TaskStatus	Current Status of the Task
Text	Plain text information
Topic	Human readable information that may be used to represent interests, knowledge, competences
URL	Uniform Resource Locator

Table 5.1: F/OSS Artifacts Overview.

Relation	Artifact	Explanation
Equivalence	UnitType	A piece of content is equivalent of another one if they are symmetric, transitive, reflexive
Partial Ordering	Date	Dates formats do not matter to evaluate substitutability. Thus *2006.03.01* $\simeq$ *March the 1st 2006* $>$ than *2006.02.13* i.e. if it is antisymmetric, transitive, reflexive
Context Dependent	Patch	A Patch p_1 can substitute another Patch p_2 if they correct the same Defects and if no installed content needs p_2. Thus Patch substitutability depends on the Substitutability of different Artifacts and also depends on the installation context.

Table 5.2: Possible substitutability relations and examples

$$
\begin{array}{rcl}
e & :== & \{a_1, ..., a_n : \mathcal{A}_i \mid cond_i\} \\
& | & e_1 \ \wedge \ e_2 \\
& | & e_1 \ \vee \ e_2 \\
& | & \overline{e}
\end{array}
$$

The condition $cond_i$ is an optional restriction on the set $\{a_1, ..., a_n\}$ whose elements all belong to the Artifact set $\mathcal{A}_i$. This restriction can use all operators that can be applied to the elements of $\mathcal{A}_i$ and the operator $\simeq$. Thus, the five preceding search criteria are expressed as:

1. $(s_1 \vee s_2 \ldots\ldots \vee s_m) \wedge \overline{(s_{m+1} \vee s_{m+2} \ldots\ldots \vee s_n)}$;
2. $\overline{l}$
3. $f_1 \wedge f_2$
4. $(v \mid v \geq 2)$
5. $(v \mid \text{true})$

Artifact Directories

As mentioned, different types of Artifacts are used within a project, e.g., licenses, code descriptions, locations, etc. We use the term *directory* to denote a grouping of Artifacts of different types. Directories can thus be used to:

- Represent any compound Artifact with its Attributes
- Represent any unanticipated grouping of Artifacts, without having to create a corresponding compound Artifact

$\mathcal{A}_i \sqsubset \mathcal{D}$ is an expression that evaluates to true if, and only if, the directory $\mathcal{D}$ contains the Artifact set $\mathcal{A}_i$.

For the ease of writing, Artifacts are auto-boxed to directories. Indeed, an Artifact a_i can be implicitly used as a directory $\mathcal{D}$ containing a single set itself containing the Artifact a_i. Thus, for instance, $a_i \sqsubset \mathcal{D}$ is true if and only if the directory $\mathcal{D}$ contains the Artifact set $\mathcal{A}_i$ where $a_i \in \mathcal{A}_i$.

Directories are built from Artifact sets and Directories using the $+$ and $-$ operators. The semantics of directories is defined in Table 5.3. A particular Artifact set $\mathcal{A}_i$ contained in a directory $\mathcal{D}$ is selected using the (.) operator. The $\ll_i$ operator is a utility function that defines a directory from an existing directory; the two directories differ by a component set.

Matching

Another requirement highlighted in Chapter 4 was the need for a means for efficient and simple resources matching. This is provided by the Artifact matching procedure. In this procedure, Artifacts, Directories and expressions are compared. Each Artifact type component is compared through the matching operation *m()* with the corresponding target type be it a single Artifact, an Artifact set or an Artifact Directory. Indeed, the trivial matching case consists of matching an Artifact $a_{template}$ with another one a_{target}; where $m(a_{template}, a_{target})$ verifies if $a_{template}$ is substitutable with a_{target}. That is,

$$m(a_1, a_2) \quad \Leftrightarrow \quad a_1 \simeq a_2$$

$$
\begin{array}{lcl}
\mathcal{A}_i + \mathcal{A}_j = \mathcal{D} & \Rightarrow & \mathcal{A}_i \sqsubset \mathcal{D} \wedge \mathcal{A}_j \sqsubset \mathcal{D} \\
\\
\mathcal{D}_1 + \mathcal{A}_j = \mathcal{D}_2 & \Rightarrow & \mathcal{A}_j \sqsubset \mathcal{D}_2 \wedge \forall\, \mathcal{A}_i \sqsubset \mathcal{D}_1, \mathcal{A}_i \sqsubset \mathcal{D}_2 \\
\mathcal{D}_1 + \mathcal{D}_2 = \mathcal{D}_3 & \Rightarrow & \forall\, \mathcal{A}_i \sqsubset \mathcal{D}_1, \mathcal{A}_i \sqsubset \mathcal{D}_3 \wedge \forall\, \mathcal{A}_j \sqsubset \mathcal{D}_2, \mathcal{A}_j \sqsubset \mathcal{D}_3 \\
\\
\mathcal{D}_{ij} - \mathcal{A}_j = \mathcal{D}_i & \Rightarrow & \neg\, (\mathcal{A}_j \sqsubset \mathcal{D}_i) \wedge \forall\, \mathcal{A}_i \sqsubset \mathcal{D}_{ij}, \mathcal{A}_i \sqsubset \mathcal{D}_i \wedge \mathcal{A}_i \neq \mathcal{A}_j \\
\mathcal{D}_{ij} - \mathcal{D}_j = \mathcal{D}_i & \Rightarrow & \forall\, \mathcal{A}_i \sqsubset \mathcal{D}_{ij}, \mathcal{A}_i \sqsubset \mathcal{D}_i \wedge \forall\, \mathcal{A}_j \sqsubset \mathcal{D}_j, \neg\, (\mathcal{A}_j \sqsubset \mathcal{D}_i) \wedge \mathcal{A}_i \neq \mathcal{A}_j \\
\\
\mathcal{D}.i & = & \mathcal{A}_i \\
\mathcal{D} \ll_i \mathcal{A}_i\text{'} & = & \mathcal{D} - \mathcal{D}.i + \mathcal{A}_i\text{'}
\end{array}
$$

Table 5.3: Artifact set and directory algebra.

The matching process for Artifact Sets A and Directories D relies on Artifacts matching as defined previously. Artifact matching provides a boolean result indicating if the target Artifact provides the properties searched by the template Artifact. Artifact Sets matching extends the matching process to sets of Artifacts, while Directory matching enables Artifact expressions handling by the matching process. While Artifact and Artifact Set matching tries to match *all* specified criteria, Directory matching allows to match only *a selection* of these criteria. This is achieved *via* Attribute Expressions. Table 5.4 summarizes the semantics for matching of Artifact sets and Directories. Matching is denoted by the matching function $m()$. It indicates that $A_{template}$, $D_{template}$ in order to match A_{target}, D_{target}, all of the elements of $A_{template}$, $D_{template}$ have to match corresponding elements of A_{target}, D_{target}.

$$
\begin{array}{lcl}
m(A_{template}, A_{target}) & \Leftrightarrow & \forall\, a_j \in A_{target}. \exists\, a_i \in A_{template}. m(a_i, a_j) \\
m(D_{template}, D_{target}) & \Leftrightarrow & \forall\, D_{template}.i\ .\ \exists\, D_{target}.j\ .\ m(D_{template}.i, D_{target}.j) \wedge \\
 & & i = j
\end{array}
$$

Table 5.4: Semantics of Attribute Set and Directory matching.

$$
\begin{array}{lcl}
m(e_{template1} \wedge e_{template2}, A_{target}) & \Leftrightarrow & m(e_{template1}, A_{target}) \wedge m(e_{template2}, A_{target}) \\
m(e_{template1} \vee e_{template2}, A_{target}) & \Leftrightarrow & m(e_{template1}, A_{target}) \vee m(e_{template2}, A_{target}) \\
m(\overline{e_{template}}, A_{target}) & \Leftrightarrow & \neg\, m(e_{template}, A_{target})
\end{array}
$$

Table 5.5: Semantics of expression matching.

5.1.2 PRM Artifacts

The PRM Model provides different key Artifacts needed to express any F/OSS process, which are built on top of the data structures presented in the previous section. This section describes these Artifacts and presents how they are related to each other. An overview of these Artifacts, is illustrated by Figure 5.2.

To detect the key concepts of the F/OSS process we used a method named *conceptual maps*. This method raises questions about the ownership and responsibility of information through the detection of information flows existing between a given set of concepts. It helped us extracting a list of core F/OSS Artifacts and Activities. More information about conceptual maps is available in Appendix A.

Figure 5.3 provides a map illustrating F/OSS concepts' interrelations. On this map, orange nodes represent PRM key Artifacts, green nodes depict concepts related to these Artifacts, and teal nodes are concepts binding the Artifacts with each other.

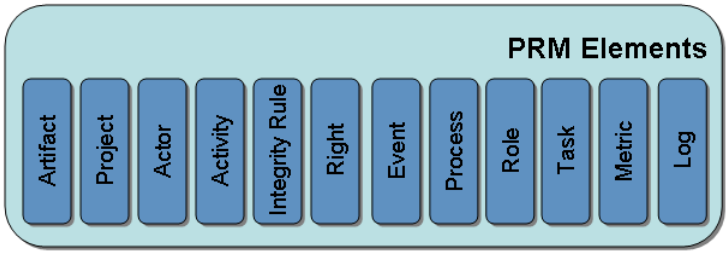

Figure 5.2: Core PRM Artifacts

In the following sections, note that specified Attributes for the different Artifacts are **all** mandatory unless specified otherwise.

Actor

The Actor PRM Artifact represents any participant in the F/OSS Process. It can be a community member (unique person, or institution.) For each Actor a, a_D is an Artifact Directory representing the interests, knowledge and competences of a in different topics represented by Topic Artifacts. The directory D contains the expressions $D.interests$, $D.knowledge$ and $D.competences$ corresponding to these topics. There are two specific types of Actors: individual contributors or institutions grouping multiple Actors and having a single contact point. However, as the Artifact can be extended, we only keep minimal information in the Actor Artifact and let users handle such differentiations by extending the ContactInformation Artifact. The resulting Attributes list is provided by Table 5.6.

Attribute Name	Attribute Type	Description
contact	ContactInformation	Contact information about the Actor
interests	Directory $_{Topic}$	Interests of the Actor
knowledge	Directory $_{Topic}$	Knowledge of the Actor
competences	Directory $_{Topic}$	Competences of the Actor

Table 5.6: Actor Artifact Attributes

The Attributes of the ContactInformation Artifact are listed in Table 5.7

Attribute Name	Attribute Type	Description
name	Text	Contact's name
URL	URL	Contact's URL
email	EMail	Contact's email

Table 5.7: ContactInformatiom Artifact Attributes

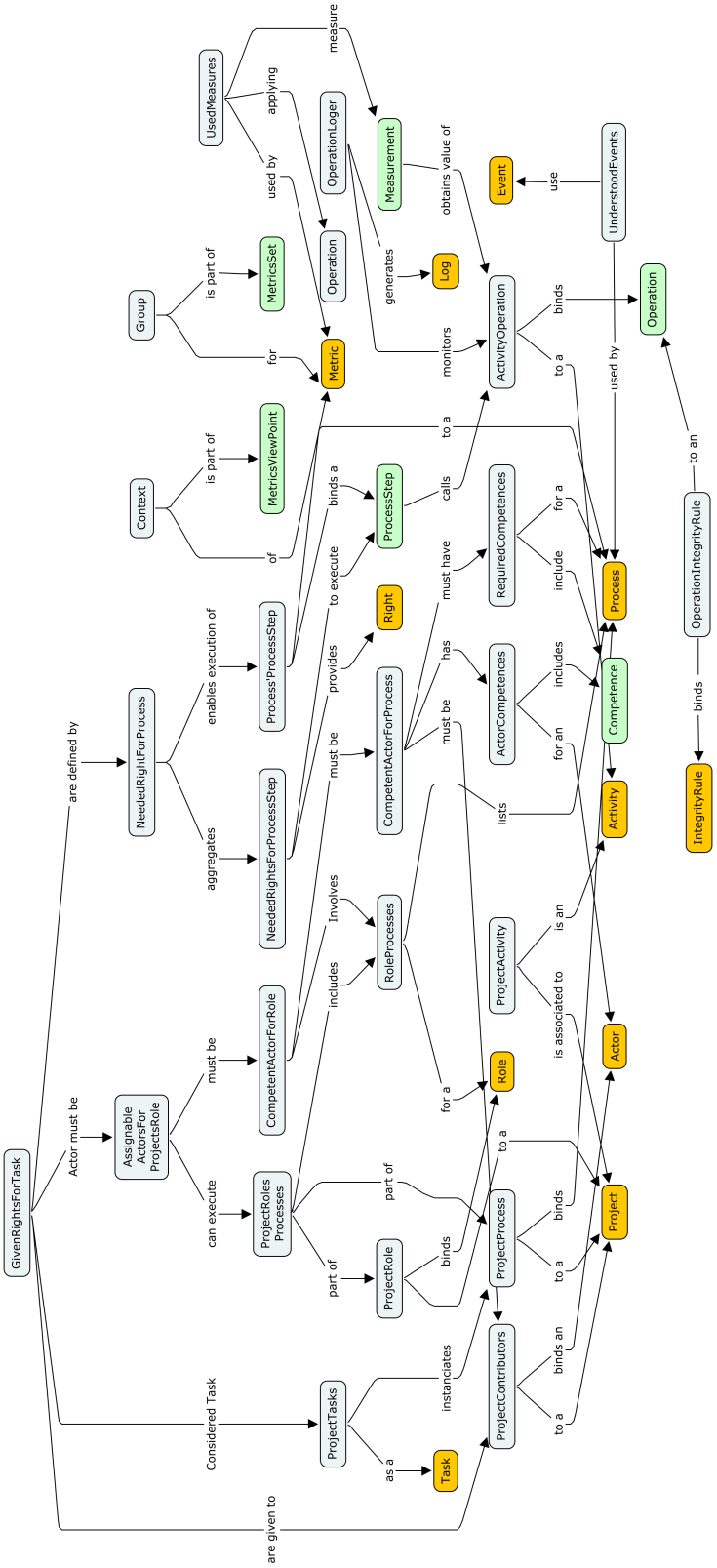

Figure 5.3: Conceptual Map of Core PRM Artifacts

Project

The Project PRM Artifact provides information needed to describe a F/OSS project. A Project p gathers community members $p_A \subseteq Actor$ around a given set of topics $p_{To} \subseteq Topic$ in order to provide a goal represented as a set of functionalities p_F. For instance in the case of a F/OSS project such as OpenOffice.org the functionalities which are put forward may be the ones indicating that the project provides a word processor, a spreadsheet, and so on, while the topics inform Actors that the project is related to an office software suite. Projects can have a parent Project and multiple sub-Projects. Table 5.8 summarizes Project's Attributes list.

Attribute Name	Attribute Type	Description
name	Text	Project's name
acronym	Text	Project's acronym
description	Text	Project's description
contact	Actor	Contact Actor for the Project
topics	$\mathbb{P}$Topic	Topics the Project is working on
url	Url	URL of the Project
parentProject	Project	Parent Projects of the Project
subProjects	$\mathbb{P}$Project	sub-Projects of the Project

Table 5.8: Project Artifact Attributes

Activity

Each F/OSS Project involves a collection of Activities. As stated before, common F/OSS activities include Community Management, Metrics Management, Rights Management, Projects Management, Distribution Management, Production Management, Defects Management, Testing Management or Dependency Management, Process Management and Task Management.

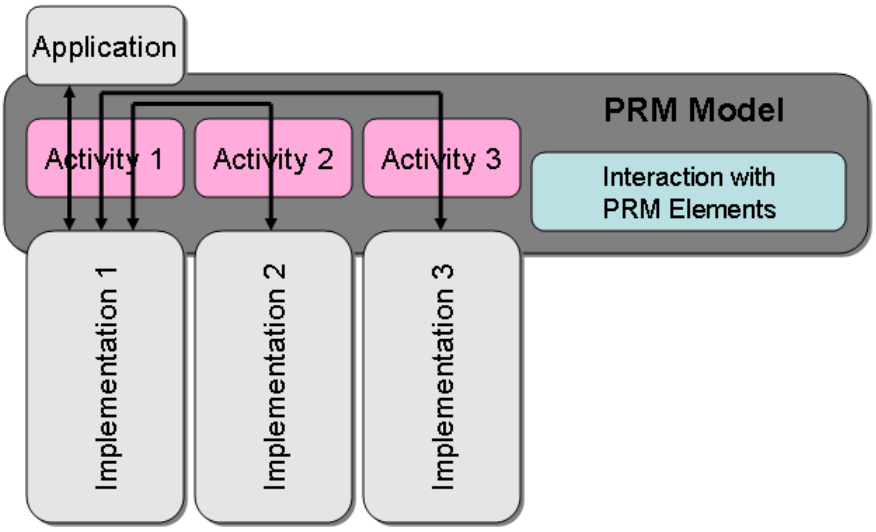

Figure 5.4: PRM Interactions

A F/OSS Activity indicates how the Artifacts it is responsible for can be handled. It defines the means which have to be provided by their implementations, making sure that required information is available

and retrievable. These means are represented as operations. For instance, the operations provided by the Artifact Management Activity allow to match Artifacts, retrieve them and verify their substitutability .

Activities have to be implemented then their operation be used to handle Artifacts. Figure 5.4 illustrates the interaction of an application with an implementation 1 which interacts with two other activity implementations 2 and 3.

The set of Activities of a F/OSS Project p is noted p_{Ac}. Each activity ac provides a set of operations ac_O. A single operation $\lambda \in O$ of the interface ac is then noted $ac.\lambda$. Each operation can take an Artifacts Directory as parameter, and can return a Directory of Artifacts.

The attributes of the Activity Artifact are listed in Table 5.9. Note that $\text{Directory}_{\lambda,\mathbb{P}\,IntegrityRule}$ indicates a Directory containing an operation λ along with an associated set of integrity rules.

Attribute Name	Attribute Type	Description
name	Text	Integrity Rule name
description	Text	Activity description
topics	$\mathbb{P}$Topic	Topics the Activity is related to
operations	$\mathbb{P}\text{Directory}_{\lambda,\mathbb{P}\,IntegrityRule}$	the set of operations provided by the Activity

Table 5.9: Activity Artifact Attributes

The PRM includes a set of core Activities providing all necessary means to provide the minimal set of operations on top of which other F/OSS Activities can be built. These Activities include Community Management, in order to handle contributors, Project Management in order to create and handle Projects, to define rights and obligations of contributors in the scope of existing Projects, Events Management in order to enable asynchronous communication, Metrics Management for measurement as well as an Activity for Activity Management itself.

Further, the PRM provides a means for adding new Activities and remove them through the declare and remove operations. These operations make available the Activity, and thus related operations, to PRM users in the scope of a Project and thus ensures the evolvability of Projects through the addition of new Activities. Related operations are further discussed in the next section.

Integrity Rule

For each operation of each declared activity, the PRM can define Integrity Rules. These rules ensure that the Activity and resources used within the activity do not endanger the F/OSS Process. They ensure thus the correctness and integrity of the behavior of each operation. Multiple Integrity Rules can be associated with these operations. The attributes of the Integrity Rule Artifact are listed in Table 5.10

Attribute Name	Attribute Type	Description
name	Text	Integrity Rule name
description	Text	Integrity Rule description
topics	$\mathbb{P}$Topic	Topics the Integrity Rule is related to
operation	λ	Operation to which the Integrity Rule is bound
rule	IntegrityRuleExpression	Expression of the Integrity Rule

Table 5.10: Integrity Rule Artifact Attributes

An integrity rule ir is an Artifact including an expression ir_{expr} which is a first class object and defines the rule to be enforced. Enforcement is done prior to the execution of the operation. Associating ir to an operation λ of an Activity ac results in the verification of ir each time $ac.\lambda$ is called.

The set of integrity rules for $ac.\lambda$ is noted $ac.\lambda^{IR}$. When $ac.\lambda$ is called, each $ir \in ac.\lambda^{IR}$ must be enforced. If any ir fails, the operation call is canceled.

Note that Integrity Rules can also be associated to Artifact types. In such a case, the Integrity Rule is checked at Artifact instantiation.

Right

Multiple Actors can interact with each Activity, each having specific responsibilities. Controlling F/OSS Activities' usage is mandatory in order to be able to better understand the whole F/OSS Process, to be able to coordinate activities, refine Actors' responsibilities and thus be able to highlight workflows, analyze them, streamline the F/OSS process.

Thus, for security reasons, some behavior must be restricted and only allowed to authorized community members. It is thus mandatory to have a way to declare in the scope of a Project who is allowed to do what and then be able to easily retrieve such information and handle it.

The Right PRM Artifact aims at describing how Actors can behave in the scope of each Activity. As the rules and constraints related to rights management may differ depending on the Project, Rights are generic and expressed in function of PRM operations. A Right defines which operations an Actor can trigger *if he receives the express authorization - the Right- to do it* and the concrete context (represented by a Task Artifact, which is presented in a further section) in which they can be called.

Rights declaration, assignment, revocation as well as verification are handled through a set of operations which are described in the next section. Table 5.11 summarizes Right's Artifact attributes.

Attribute Name	Attribute Type	Description
Task	Task	Task in the scope of which a Right can be used
activity	Activity	Activity for which the Right is valid
operation	λ	operation for which the Right is valid
extensionOf	Directory_{Right}	The Right

Table 5.11: Right Artifact Attributes

The PRM does not only provides a means to express Rights, it also provides full support for Rights management, from Community members registration to Rights assignment.

Process

The interactions with the PRM are a sequence of calls to the different Activity operations provided by Projects. These sequences are described using the Process Artifact. The Process Artifact indicates thus how users have to interact with the PRM to reach a goal. This allows to have different Process descriptions which can be exchanged by projects.

The structure of the process is formalized in figure 5.5. A Process π is a series of calls, to any Activity operation $ac.\lambda$ passing a directory D of parameters. These calls can be organized as a graph, containing loops and conditions and each of them returns a value v as an Artifact Directory. Further a Process can be executed a given nuber of times (^{n}), 0 or multiple times ($^{-}$) or at least one time ($^{+}$).

Processes can be executed in parallel. To synchronize them, Events are used. The waiting process can wait for a Event type through the observe operation, and other processes can raise an Event of this type through the raise operation of the Event management Activity. Events and related operations are described later in this chapter.

The requirements of a process π of a Project p are defined by Actors $a \in p_A$, which are responsible for process definition. The requirements are expressed as an Artifact Directory noted π_D. It represents

$$
\begin{array}{rcl}
\pi_i & ::= & \pi_j ; \pi_k \\
 & | & \pi_j \,||\, \pi_k \\
 & | & \pi_j^n \\
 & | & \pi_j^+ \\
 & | & \pi_j^- \\
 & | & cond ? \pi_j : \pi_k \\
 & | & * cond \pi_j \\
 & | & ac.\lambda(D) -> v \\
 & | & nil
\end{array}
$$

Figure 5.5: Process Structure

the interests, knowledge and competences in different topics which are expected from Actors undertaking the responsibility of executing π. This ensures that only adequate Contributors are able to execute it. As for Actors interests, knowledge and competences are represented by Topic Artifacts. The directory D contains the expressions $D.interests$, $D.knowledge$ and $D.competences$ corresponding to these topics. Process requirements can thus be matched to Actors interests, knowledge and competences. The attributes of the Process Artifact are listed in Table 5.12.

Attribute Name	Attribute Type	Description
name	Text	Name of the Process
description	Text	Process description
topics	$\mathbb{P}$Topic	Topics the Process is related to
interests	Directory $_{Topic}$	Interests of the Actor
knowledge	Directory $_{Topic}$	Knowledge of the Actor
competences	Directory $_{Topic}$	Competences of the Actor
process	π_i	The process expressed in terms in calls

Table 5.12: Process Artifact Attributes

Role

Once processes have been declared they can be grouped using Roles. The set of Processes r_π indicates the Processes part of the Role r. Any Actors receiving r is in charge of executing the Processes contained in r_π.

A Role only describes a set of Processes, which can be potentially used in the scope of any Project. As such, in order to assign r to an Actor a in the scope of a Project p, r must be declared for p. Usable Roles in the scope of p are noted p_R.

Role assignment to a can only be done if a interests, knowledge and competencies match the requirements associated with the different Processes of the Role. The attributes of the Role Artifact are listed in Table 5.13.

Attribute Name	Attribute Type	Description
name	Text	Name of the Role
description	Text	Role description
processes	$\mathbb{P}$Process	Processes part of the Role

Table 5.13: Role Artifact Attributes

Task

Once a Role r have been declared for a Project p and assigned to Actors, the execution of Processes $r_\pi, r \in p_R$ can be done through Tasks. A Task t is the assignment of a Process $\pi \in r_\pi$ to an Actor a in the scope of p. Tasks are assigned automatically to Actors at Role attribution time.

Tasks define Actors responsibilities regard to a Process as long as the task is assigned. Assigning a task to an actor a regulates the way a can interact with the Activities of a Project p. Task assignment implies that the rights needed to execute the calls described by π have to be attributed to a. Thus, the assignment of π to a in the scope of p implies that a possesses the Rights to call all operations λ involved in in the step s of π.

A Task $t \in a_T$ has a status represented by a TaskStatus Artifact $t.status$. It can be of the following value: assigned, ongoing, abandoned, paused or completed. The ProcessType of the Process the Task is based on, defines if the Task is recurring or once-off. Once completed, a once-off Task is completed, it is unassigned from the Actor and thus related Rights are removed. Recursive Tasks $t.status$ becomes *assigned* once completed, which means that the process can be executed aain, instead. The attributes of the Task Artifact are listed in Table 5.14.

Attribute Name	Attribute Type	Description
project	Project	Project in the scope of which the Right can be used
process	Process	Process to be executed for the Task
actor	Actor	Actor having the responsibility to execute the Task
role	Role	Role the Process is part of
assignedBy	Actor	Actor having assigned the Task
status	TaskStatus	status of the Task
assignementDate	Date	Date when the task has been assigned
startDate	Date	Date when the Task has been started
endDate	Date	Date when the Task has been finished (abandoned or completed)

Table 5.14: Task Artifact Attributes

Event

An Event is an envelope for signaling information asynchronously to PRM users as well as a means for synchronization. They can be used in multiple situations. For instance, a Linux distribution project may want to be informed of new releases of a project it is integrating to its distribution. Further being automatically notified of bug fixes provided by the community of integrated projects can help avoid searching for this fixes manually. Further, some Events may trigger the execution of processes. This may be useful in the case of community members quitting a project: when unregistered a process seeking for a replacement contributor could be run automatically.

There are two main types of Events. Execution Events are raised when a PRM operation is called. This type of Events holds context information such as the operation having been executed, the Task

which triggered it as well as temporal information. Alarm Events are raised to indicate that a special state is reached. As for Execution Events, they carry context information, but also any other information. They are used for raising metrics execution results as well as for handling user Events. Indeed, only Alarm Events can be used to implement user defined Events. Table 5.15 summarizes Event attributes.

Attribute Name	Attribute Type	Description
type	EventType	type of the event.
raisedBy	Actor	Actor who has raised the Event
project	Project	Project in which scope the Event has been raised
process	Process	Process in which scope the Event ahs been raised
date	Date	date when the Event has been raised
content	Directory	content of the Event

Table 5.15: Event Artifact Attributes

PRM users can raise, (un)subscribe and observe Alarm Events, nevertheless Execution Events are only raised by Activity implementers. Event observation is a blocking operation and can be used to synchronize Processes on Events. Any Process can generate its own events and inform observing processes that something happened. For instance, a process may indicate that it has been terminated.

Back to the example introduced above, Events can be used for informing that a new stable version of a software has been released by a project p_1. A Linux distribution provider p_2 may wish to be informed of such releases until a given date, in order to include it in it's next release. In such a situation, the process of distribution adopted by p_2 might be:

$$\begin{aligned}\pi_{p_2} \quad = \quad & EventManagementActivity.observe(StableReleaseEvent, Time) -> result; \\ & InclusionManagementActivity.include(result);\end{aligned}$$

This process indicates that a StableReleaseEvent has to be waited until the Time date. If such an Event is raised, then the result is passed to the InclusionManagementActivity of the project, for trying to integrate the result to the distribution. To trigger this inclusion, a StableReleaseEvent has to be raised by the process run by p_1 as follows:

$$\begin{aligned}\pi_{p_1} \quad = \quad & ...; \\ & EventManagement.raise(StableReleaseEvent);\end{aligned}$$

Metric

Measurement is a key requirement for all processes. In our model, a metric represents a measure and is an Artifact type of the model. The syntax of a metric is given in Figure 5.6 and the Attributes of the Metric Artifact are listed in Table 5.16

The context of the measure is described by elements of two attribute sets: the $metricViewPoint$ and $metricsSet$ Attribute sets. The viewpoints of a metric represent the most general contexts a metric Artifact belongs to. For instance it may define that the metric is related to the testing Activity and to the activity analysis Activity. Elements of the MetricsSet attribute set define a logical relation between different separate metrics. The metric's MetricsSet attribute defines thus the set of logically related metrics the metric is part of. The Metric also includes an Artifact describing the unit of measurement

Attribute Name	Attribute Type	Description
name	Text	Metric's name
description	Text	Metric's description
viewpoint	ℙMetricViewPoint	Measurement domain
set	ℙMetricSet	set of Metrics this Metric belongs to
unitOfMeasurement	Text	Unit of measurement of the returned value
metricExpression	Measurement Expression	expression indicating what is measured

Table 5.16: Metric Artifact Attributes

$$
\begin{array}{lcl}
measure & ::= & ac.\lambda(D_{param})\ s.t.\ ac.\lambda \rightarrow Number \\
 & | & numerize(ac.\lambda(D_{param}))\ s.t.\ numerize \rightarrow Number \\
 & & \\
metricExpression & ::= & operand\ s.t.\ operand \in \{\mathbb{R}, measure\} \\
 & | & operand\ (op\ metricExpression)_{cond}\ s.t.\ op \in \{+, -, *, /, \%\}
\end{array}
$$

Figure 5.6: Metric Execution Expression definition

of the value returned when evaluating the metric. The value attribute can be for instance *dollars in hundreds* or *percentage*.

Metrics are calculated *ondemand* which indicates that the measure has to be done on explicit user demand through the Metrics Management Interface. However, a measure could also be taken each time a given PRM Activity operation $a.\lambda$ is called in order to handle dynamic metrics that have to be continuously measured on a real time basis instead of limiting the PRM to punctual measurements. While such a feature is not part of the model core, it has been added as an extension of the PRM and will be further explained in part III

The core of the PRM metrics management is described as a *measurement expression* metricExpression which is a combination of different single measures. Figure 5.6 defines such a measure as a call to a λ operation of an Activity ac provided a Directory D_{param} holding parameters to pass to the $ac.\lambda$ operation. The operation $ac.\lambda$ must return a $Number$ Artifact or if it not the case, the result must be treated by a $numerize$ mechanism returning a Number. Such a mechanism can for instance, in the case of arrays, count elements of the array, if the array contains Numbers, sum them and extract the max or min value. In the case of directories, the mechanism should be able to extract a field of the Directory to get numeric values.

The metric execution expression takes single measures and combines them using numeric operators allowing to add, subtract, multiply, divide results or keep the rest of a division. The $metricExpression$ can also be weighted with a value. Further, a condition $cond$ can be provided in order to define that only measures fulfilling $cond$ have to be taken into account by the $metricExpression$.

In order to achieve measures over time, the Logging Activity has to be queried in order to retrieve information having occured at a given time, or during a time slot to be measured such as "between $date_1$ and $date_2$" or "from $date_1$ up to now".

An execution expression allows users to build complex metrics based on values held by the PRM. As Metrics are Artifacts this implies that they can be retrieved. Further, as Metrics can be reused within the $measure$ execution expression of other Metrics, this allows any community member to compose them to calculate other Metrics. The set of Metrics referred by the execution expression $measure$ of a Metric m is noted m_{calls}.

The Metrics Management part of the PRM provides all needed means for handling Metric Artifacts and execute measures on the PRM. It allows responsible users to register new Metrics, associate them to projects, evaluate them but by using the Event Management features of the PRM, it also provides a means for declaring interests in some metrics in order to be informed when they are triggered or when some threshold values are reached using PRM Events. Related operations are described in the next section.

Log

The Log PRM Artifact is an Artifact meant to capture and keep track of the PRM usage. Each time an Activity operation is called; the corresponding ExecutionEvent raised by the called Activity is captured by the Logging Activity. The capture generates a Log Artifact which is then held by the Logging Activity. The latter can be queried to obtain information about past usage of the PRM. The Logging Activity provides operations to retrieve information from captured Logs. As such, Metrics can be built to extract this information.

Each Log Artifact provides complete information about a logged event. The context of the Log is thus represented by the following attributes: the task t having triggered the call (and which provides information about the Actor having executed the call, the role of the actor, the project and the process) and the process step s in the scope of which the call was made, a Directory of arguments $args$ passed to the operation, the returned result $result$ as a Directory, a timestamp Date ts indicating when the call occurred and a Number Artifact indicating the duration d of the call. Note that the Activity and operation information can be extracted from the ProcessStep attribute. The attributes of the Log Artifact are summarized in Table 5.17.

Attribute Name	Attribute Type	Description
task	Task	task having led to the call
step	ProcessStep	process step having executed the call
args	Directory	arguments passed to the operation
result	Directory	result of the call
date	Date	date of the call occurrence
duration	Number	duration of the call in milliseconds

Table 5.17: Log Artifact Attributes

All symbols of the PRM formalism are summarized in Table 5.18. These symbols are used in the remainder of this Thesis.

5.1.3 PRM Artifacts substitutability

As every Artifact, all PRM Artifacts respect a substitutability relation. We detail this relation in the following paragraphs.

Activity Two Activities ac_1 and ac_2 are substitutable if ac_2 provides a set of operations substitutable to ac_1's set. This implies that the operations attribute represented by Directories holding the λ operations and associated integrity Rules of ac_1 is substitutable with the operations attribute of ac_2.

$$ac_1 : Activity \simeq ac_2 : Activity \quad \Leftrightarrow \quad \mathsf{attr}_{operations}(ac_1) \simeq \mathsf{attr}_{operations}(ac_2)$$

Symbol	Description
λ	Operation
λ^{IR}	Set of Integrity rules associated to an operation
a	Actor
A	All Actors
a_{Rg}	Set of Actor's Rights
a_R	Set of Actor's Roles
a_T	Set of Actor's Tasks
a_R^p	Set of Actor's Roles in the scope of a Project
a_π^p	Set of Actor's Processes in the scope of a Project
a_π^t	Set of Actor's Processes in the scope of a Task
a_{Rg}^p	Set of Actor's Rights in the scope of a Project
a_T^p	Set of Actor's Tasks in the scope of a Project
a_{Rg}^t	Set of Actor's Rights in the scope of a Task
a_D	Directory containing Actor's interests, knowledge and competences
ac	Activity
Ac	All Activities
$ac.\lambda$	Operation λ of an Activity
ac_O	Set of Operations of an Activity
e	Event
ir	Integrity rule
ir_{expr}	Integrity Rule Expression
π	Process
π_D	Directory containing Process requirements as Actor's interests, knowledge and competences
l	Log
m	Metric
M	All Metrics
m_{calls}	Set of Metrics called by a Metric
p	Project
P	All Projects
p_π	Set of Project's Processes
p_A	Set of Project's Actors
p_M	Set of Project's Metrics
p_T	Set of Project's Tasks
p_{To}	Set of Project's Topics
p_F	Set of Project's Actors
p_{Rg}	Set of Rights assignable in the scope of a Project
p_R	Set of Roles assignable in the scope of a Project
p_{Ac}	Set of Project's Activities
r	Role
r_π	Set of Processes part of a Role
rg	Right
s	Process step
t	Task

Table 5.18: Summary of PRM symbols

Actor Actor substitutability can be needed by Project Managers when searching for community members able to replace another community member. From projects Managers viewpoint Actors are mainly described by their Interests competencies and Knowledge. Thus an Actor a_1 is substitutable by an Actor a_2 if a_2 interests include a_1 interests, a_2 competencies include a_1 competencies, and a_2 knowledge includes a_1 knowledge.

$$\begin{aligned} a_1 : Actor \simeq a_2 : Actor \quad \Leftrightarrow \quad & \mathsf{attr}_{interests}(a_1) \subseteq \mathsf{attr}_{interests}(a_2) \wedge \\ & \mathsf{attr}_{knowledge}(a_1) \subseteq \mathsf{attr}_{knowledge}(a_2) \wedge \\ & \mathsf{attr}_{competencies}(a_1) \subseteq \mathsf{attr}_{competencies}(a_2) \end{aligned}$$

Nevertheless, other information can help in defining if a_1 is substitutable by a_2. For instance Actors reputation, evaluation of Actor competencies through the execution of metrics or voting can provide useful indicators and can be evaluated in PRM extensions. Thus Actors substitutability can be defined by the two above rules, but can also be refined in order to provide more accurate results using other information provided by the PRM and available metrics.

Event Events are substitutable if their content carries substitutable information and if their type is the same. As they are bound to time, their substitutability also depends on the time of their occurrence. Indeed, evaluating if an Event has the same impact than another one is directly bound to when the Event occurred. For instance, when waiting for a software update event, users are only interested in event occurring after the beginning of the observation, and thus past event or not interesting anymore. As we can not handle dynamic conditions for the substitutability, we have to leave this verification to Event users.

An Event e_1 is thus substitutable with Event e_2 if and only if the type of e_2 is the same as the one of e_1 and if the content of e_2 can substitute e_1's content. This can be expressed as follows:

$$\begin{aligned} e_1 : Event \simeq e_2 : Event \quad \Leftrightarrow \quad & \mathsf{attr}_{type}(e_1) == \mathsf{attr}_{type}(e_2) \wedge \\ & \mathsf{attr}_{content}(e_1) \simeq \mathsf{attr}_{content}(e_2) \end{aligned}$$

Integrity Rule Integrity Rules are substitutable if they are related to the same operation of an Activity and if they enforce the same expression Rule. This is formalized as follows:

$$\begin{aligned} ir_1 : IntegrityRule \simeq ir_2 : IntegrityRule \quad \Leftrightarrow \quad & \mathsf{attr}_{activity}(ir_1) == \mathsf{attr}_{activity}(ir_2) \wedge \\ & \mathsf{attr}_{operation}(ir_1) == \mathsf{attr}_{operation}(ir_2) \wedge \\ & \mathsf{attr}_{rule}(ir_1) \Rightarrow \mathsf{attr}_{rule}(ir_2) \end{aligned}$$

Process Two Processes are substitutable if they aim at achieving the same goal, do it in the same way, define the same requirements for their execution and are of the same type. The goal and used means for reaching it is defined through the use of Process expressions as presented in previous section. These expressions provide the exact sequence of calls to PRM Artifacts' operations. As such they can be compared, and one can make sure that these sequences are the same.

Thus a Process π_1 is substitutable by a Process π_2 if π_2 required interests include π_1 required interests, π_2 required competencies include π_1 required competencies, π_2 required knowledge includes p_1 required knowledge and if the process expression of p_1 and p_2 are identical.

$$\begin{aligned} \pi_1 : Process \simeq \pi_2 : Process \quad \Leftrightarrow \quad & \mathsf{attr}_{interests}(\pi_1) \subseteq \mathsf{attr}_{interests}(\pi_2) \wedge \\ & \mathsf{attr}_{type}(\pi_1) == \mathsf{attr}_{type}(\pi_2) \wedge \\ & \mathsf{attr}_{knowledge}(\pi_1) \subseteq \mathsf{attr}_{knowledge}(\pi_2) \wedge \\ & \mathsf{attr}_{competencies}(\pi_1) \subseteq \mathsf{attr}_{competencies}(\pi_2) \wedge \\ & \mathsf{attr}_{process}(\pi_1) == \mathsf{attr}_{process}(\pi_2) \end{aligned}$$

Project The information that differentiates Projects is the topics on which they are working and if they provide substitutable Activities. We consider thus a Project p_1 as substitutable with Project p_2 if and only if the topics on which p_2 is working cover the topics on which p_1 is working and if the set of Activities of p_2 can substitute p_1's Activities. This can be expressed as follows:

$$p_1 : Project \simeq p_2 : Project \quad \Leftrightarrow \quad \begin{array}{l} \mathsf{attr}_{topics}(p_1) \simeq \mathsf{attr}_{topics}(p_2) \wedge \\ \mathsf{attr}_{activities}(p_1) \simeq \mathsf{attr}_{activities}(p_2) \end{array}$$

Right Right Substitutability is useful to verify if a given Right offers the same liberty another right offers. Such type of substitutability relies on the comparison of the privileges a Right provides. Indeed a Right rg_1 can be substituted by a Right rg_2 if and only if all the operation calls rg_1 is allowed to do can also be done by rg_2.

$$rg_1 : Right \simeq rg_2 : Right \quad \Leftrightarrow \quad \begin{array}{l} \mathsf{attr}_{activity}(rg_1) == \mathsf{attr}_{activity}(rg_2) \wedge \\ \mathsf{attr}_{task}(rg_1) == \mathsf{attr}_{task}(rg_2) \wedge \\ \mathsf{attr}_{operation}(rg_1) == \mathsf{attr}_{operation}(rg_2) \end{array}$$

Role Role Substitutability can be used to avoid Role redundancy. Such type of substitutability relies on the comparison of the set of Processes a Role gathers. Indeed a Role r_1 can be substituted by a Role r_2 if and only if each Process of r_1 can be substituted by a Process of r_2.

$$r_1 : Role \simeq r_2 : Role \quad \Leftrightarrow \quad \forall \pi_1 \in \mathsf{attr}_{processes}(r_1), \pi_1 \simeq \pi_2 \mid \pi_2 \in \mathsf{attr}_{processes}(r_2)$$

Log, Metric, Task For these three PRM Artifacts, the substitutability is only possible in the case of Artifacts having the exact same values. Indeed, the meaning of these Artifacts is bound to the value of their attributes, and if any of them change, the way the Artifact must be interpreted changes. For instance the Log Artifact provides a description of a PRM usage that happened at a given moment. There is no meaning in substituting such an Artifact by another one if they are not strictly equal. In the case of Metrics, as they are defined mainly by an execution expression and by an occurrence expression, the only possibility for two Metrics to by substitutable is if their occurrence expressions and execution expressions are identical. Finally in the case of Tasks, semantically if the Actor involved in the Task execution, or the underlying Process or Role change, the Task nature changes completely and cannot thus be substituted.

5.1.4 PRM Activities and Operations

Core PRM Activities provide a set of operations allowing handling the different PRM Artifacts presented in previous section. These are thus interfaces which are used by user applications to interact with their implementations. As the PRM is meant to be a information coordinator between different projects, the information provided by PRM core Activities is logically centralized. Indeed, in order to ensure information integrity, correctness and accessibility across Project boundaries, such centralization is required. However, the PRM could also be applied at the level of single projects with no need for such centralization if inter-project communication is not wished. Figure 5.7 lists these core Activities. Note that the integrity Rule and PRM Artifacts are not directly handled through specific Activities.

Section 5.1.2 presented PRM data structures and related primitives. In this section we describe the most important operations provided by the core Activities presented in Figure 5.7. These are summarized by Tables 5.19, 5.20 and 5.21. These tables list operations involved in the creation, handling or modification of Artifacts. The usage of these different operations as well as the way user applications interact through Activities are illustrated in chapter 7.

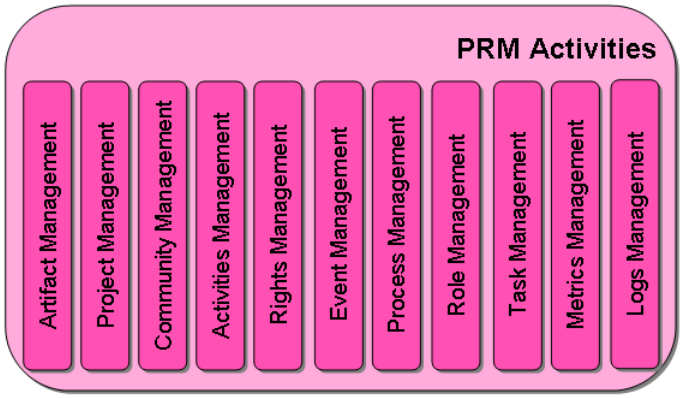

Figure 5.7: Core PRM Activities

typographic conventions. serif font is used for operation names and that $\mathsf{op}()^{p}_{a_{\pi_i}}$ expresses a operation op called by an Actor a in the scope of a Project p and a process π_i.

Artifacts

Artifact Type declaration The PRM provides basic Artifact types. However, this set of types can be extended by registering new types in order to fit users needs. The registration of a new Artifact type implies the provision of four different information. First, a directory D_t consisting of Artifact Types must be provided. This directory defines the structure of the artifact, i.e. the attributes it consists of. Second, a directory D_v consisting of boolean values indicates which of the attributes defined by D_t have to be provided at Artifact instantiation time. Then, a set of integrity rules to be verified at artifact instantiation time can be provided if needed and finally the Project for which the Artifact Type has been registered has to be declared.

Artifact Instantiation The PRM enforces the integrity of newly created Artifacts at different levels. Each Artifact type A possesses a set of integrity rules A_{IR} which have to be enforced at A's instantiation time. When instantiating an Artifact type A, if $A.i$ has been marked as $mandatory$ at declaration time, the Attribute i must receive a value. Artifacts missing mandatory Attributes, can not be created. The need for F/OSS Information and Process integrity explain such restrictions. Indeed, Activities may rely on some Attributes and may expect information provided by Artifacts to be correct.

Note that Artifacts can only be instantiated in the scope of a Project. This implies that the corresponding Artifact Type be registered for the Project, or the instantiation will fail.

Artifact Lookup The lookup operation retrieves Artifacts corresponding to a set of properties from a set of Artifacts. This operation is modeled as follows. It takes an Artifact expression $expr$ (represented as a Directory) and a Set of Artifacts Ac as arguments, then relies on a matching function which compares $expr$ with every $ac_i \in Ac$ and returns Ac_{sub} a subset of Ac containing all Artifacts matching $expr$. The semantics of the lookup operation are defined as follows.

$$\text{lookup(e, A)} \quad \rightarrow \quad A_{sub} \subseteq A \mid \forall\, a_i \in A_{sub}.m(a_i, e)$$

The use of Artifact Expressions provides lookup flexibility. Indeed when searching for Artifacts, one may wish to retrieve Artifacts whose Attributes have to a given value while not caring about the value of other Attributes of the Artifact. Every Activity can provide its own lookup operation relying on this operation to ease the retrieval of a specific type of Artifacts.

Activity Name	Description	
Artifact Management	registerArtifactType(Type Directory, Valuation Directory, $\mathbb{P}\, I\mathcal{R}$, Project):Type	Registers a new Artifact Type for a Project
	newArtifact(Type, Values Directory, Project):Artifact	Instantiates a new Artifact for a Project
	lookup(Directory,$\mathbb{P}$Artifact):$\mathbb{P}$Artifact	Attribute aware Artifact search within an Artifact set
	lookup(Directory):$\mathbb{P}$Artifact	Attribute aware Artifact search
	exists(Artifact):Boolean	verifies if an Artifact exists
	substitutable(Artifact, Artifact):Boolean	Substitutability verification
Activity Management	declareActivity(Name, $\mathbb{P}$(λ, $\mathbb{P}\, I\mathcal{R}$)):Activity	New Activity declaration in terms of operations
	bindActivity(Activity, Project):void	Associates an Activity to a Project
	lookup(Directory):$\mathbb{P}$Artifact	Attribute aware Activity search
	exists(Activity):Boolean	verifies if an Activity exists
	isAssociated(Activity, Project):Boolean	verifies if an Activity exists
Actor Management	registerActor(Directory):Actor	Registers a new Actor
	getContactInformation(Text$_{name}$, URL, EMail):ContactInformation	Retrieves a standard ContactInformation Artifact
	setContactInfo(Actor, ContactInformation):void	Defines Actor's Contact information
	setContactInfo(Actor, Directory):void	Defines Actor's Contact information
	setCompetence(Actor, Directory):void	Defines a Competence
	setInterest(Actor, Directory):void	Defines an Interest
	setKnowledge(Actor, Directory):void	Defines a Knowledge
	getContactInfo(Actor):ContactInformation	Gets Actor's Contact information
	getCompetence(Actor):Directory	Gets Actor's Competences
	getInterest(Actor):Directory	Gets Actor's Interests
	getKnowledge(Actor):Directory	Gets Actor's Knowledge
	setActive(Actor, Boolean):void	(De)activates an Actor
	isActive(Actor):Boolean	Returns if an Actor is currently set as active
	lookup(Directory):$\mathbb{P}$Artifact	Attribute aware Actor search
	exists(Actor):Boolean	verifies if an Actor exists
Event Management	raise(Event):void	Event raise
	registerListener(EventListener):void	Registers an EventListener
	registerListener(EventListener, Time):void	Registers an EventListener for a given amount of time
	unregisterListener(EventListener):void	Unregisters an EventListener
	observe(Event, Time):Event	Synchronization on an Event, waiting until it is raised
	exists(Event):Boolean	verifies if an Event exists
EventListener Management	raised(Event):void	Notifies the Listener that an Event has been raised
Log Management	createLog(Actor, Date, Project, Process, Activity, λ, $Directory_{parameters}$, $Directory_{result}$):Log	Logs a call to a PRM operation
	lookup(Directory):$\mathbb{P}$Artifact	Attribute aware Log search
	query(Text$_{queryexpression}$, Directory$_{queryparameters}$):Directory	Log querying
	exists(Log):Boolean	verifies if a Log exists
Metric Management	newMetric(Name, Description, MetricViewPoint, MetricSet, OccurenceExpr, MeasurementExpr):Metric	New Metric creation
	registerMetric(Metric):void	Metric registration
	enableMetric(Metric, Project):void	Enables Metric usage in the scope of a Project
	enabledMetric(Metric, Project):void	Checks Metric availability in the scope of a Project
	disableMetric(Metric, Project):void	Disables Metric usage in the scope of a Project
	evaluateMetric(Metric, Project, Actor):Number	Metric execution by an Actor in the scope of a Project
	usesMetrics(Metric, Metric):Boolean	Checks if a Metric depends on another Metric
	dependsOnMetrics(Metric):$\mathbb{P}$Metric	Metrics used by a Metric
	usedByMetrics(Metric):$\mathbb{P}$Metric	Metrics using a Metric
	lookup(Directory):$\mathbb{P}$Artifact	Attribute aware Metric search
	exists(Metric):Boolean	verifies if a Metric exists

Table 5.19: PRM operations (part 1).

Activity Name	Description	
Process Management	declareProcess(Name, ProcessStep$_{first}$, $Directory_{interests}$, $Directory_{knowledge}$, $Directory_{competences}$, ProcessType):Process	New Process creation for a Project
	declareProcessStep(Activity, λ):ProcessStep	Declares a Step for a Process as a call to an operation
	declareProcessStep(Process):ProcessStep	Declares a Step as a Process
	declareConditionStep(ProcessStep$_{previous}$, Condition, ProcessStep$_{iftrue}$, ProcessStep$_{iffalse}$):ProcessStep	Process Step sequencing depending on a defined condition
	declareLoopStep(ProcessStep$_{previous}$, Condition, ProcessStep$_{whiletrue}$, ProcessStep$_{whenfalse}$): ProcessStep	Process Step sequencing depending on a defined condition
	setProcessStepSequence(ProcessStep$_{previous}$, ProcessStep$_{next}$):void	Process Step sequencing
	lookup(Directory):ℙArtifact	Attribute aware Process search
	exists(Process):Boolean	verifies if a Process exists
	exists(ProcessStep):Boolean	verifies if a ProcessStep exists
Project Management	declareProject(Text$_{Name}$, Text$_{Acronym}$, Text$_{Description}$, Actor, ℙTopic, Url, Project$_{Parent}$):Project	New Project creation
	setTopics(ℙTopic):void	sets the Topics the project is working on
	setDescription(Text):void	sets the description of the Project
	setContact(Actor):void	sets the contact Actor for the Project
	registerContributor(Actor, Project):void	registers an Actor as a Project contributor
	activateContributor(Actor, Project):void	activates an Actor for a Project
	deactivateContributor(Actor, Project):void	deactivates an Actor for a Project contributor
	getContributors(Project):ℙActor	returns the contributors for a Project
	getProjects(Actor):ℙProject	returns a list of Project an Actor contributes to
	lookup(Directory):ℙArtifact	Attribute aware Project search
	exists(Project):Boolean	verifies if a Project exists
Right Management	declareRight(Activity, λ, Task):Right	New Right declaration in the scope of a project and Task
	assignRight(Right, Actor):void	Right assignment to an Actor
	removeRight(Right, Actor):void	Right removal to an Actor
	allowed(Task, Activity, λ):boolean	Activity Operation access verification
	allowed(Actor, Project, Role, Process, Activity, λ):boolean	Activity Operation access verification
	lookup(Directory):ℙArtifact	Attribute aware Right search
	exists(Right):Boolean	verifies if a Right exists

Table 5.20: PRM operations (part 2).

Activity Name	Description	
Role Management	declareRole(Text$_{name}$, Text$_{description}$, ℙProcess):Role	New Role declaration
	associateRole(Role, Project):void	Associates a Role to a Project
	assignRole(Role, Actor, Project):void	Associates a Role to an Actor
	unassignRole(Role, Actor, Project):void	Deassociates a Role from an Actor
	containsProcess(Role, Process):Boolean	Verifies if a Process is part of a Role
	isRoleAssociated(Role, Project):Boolean	Verifies if a Role has been associated with a Project
	isRoleAssigned(Role, Actor, Project):Boolean	Verifies if a Role has been assigned to an Actor in the scope of a Project
	isRoleAssigned(Role, Project):Boolean	Verifies if a Role has been assigned to any Actor in the scope of a Project
	getRoles(Project):ℙRole	Returns the Roles associated with a Project
	getUnassignedRoles(Project):ℙRole	Returns the Roles associated with a Project and not assigned to an Actor
	getRoles(Actor, Project):ℙRole	Returns the Roles assigned to an Actor for a Project
	getContributors(Role, Project):ℙActor	returns the contributors of a Project having a Role
	lookup(Directory):ℙArtifact	Attribute aware Role search
	exists(Role):Boolean	verifies if a Role exists
	proposeRole(Role, Actor, Project):void	Proposes a Role to an Actor
	proposeRoleContribution(Role, Actor, Project):void	Allows Actors to proposes their participation for a Role
	answerRoleProposition(Role, Actor, Project, Boolean)	Allows Actors to answer a role proposition
	isRoleProposed(Role, Actor, Project):Boolean	Verifies if a Role has been proposed to an Actor in the scope of a Project
	isRoleAccepted(Role, Actor, Project):Boolean	Verifies if a Role has been accepted by an Actor in the scope of a Project
	getRolesProposed(Actor, Project):ℙRole	Returns the Roles proposed to an Actor for a Project
	getContributorsPropositions(Role, Project):ℙActor	Returns the Actors having proposed their contribution for a Role for a Project
	getPotentialContributors(Role, Project):ℙActor	returns the contributors of a Project having a Role
Task Management	newTask(Actor, Project, Process, Role)	New Task creation
	getTasks(Actor, Project):ℙTask	Returns the Tasks an Actor has to do for a Project
	getTasks(Actor, Project, Role):ℙTask	Returns the Tasks an Actor has to do for a Role in a Project
	changeTaskStatus(Task, TaskStatus):void	Changes the status of the Task
	executeTask(Task, Directory):Directory	Executes a the first Step of the Process of the Task providing a Directory of parameters and returns the result
	executeStep(Task, ProcessStep, Directory):Directory	Executes a ProcessStep providing a Directory of parameters and returns the result
	nextStep(Task):ProcessStep	Indicates the next Step to be executed.
	currentStepOfTask(Task):ProcessStep	Returns current position in Process
	lookup(Directory):ℙArtifact	Attribute aware Task search
	exists(Task):Boolean	verifies if a Task exists

Table 5.21: PRM operations (part 3).

Artifact existence. The exists operation verifies if the Artifact passed as parameter has been created. As for the lookup operation, this operation can be provided by each Activity. In this case, this operation verifies if the Artifact has been declared through the Activity.

Activity

Activity Declaration Activities can be registered with the PRM using the declareActivity operation. Declaring an Activity makes available its operations to the community members. The scope of the declaration is a selected Project, meaning that different Project may have access to different Activities depending on their needs.

The declareActivity operation creates an Activity which is added to the set of available Activities Projects can use. This operation takes a Name n and a set of λ operations. Each of the operations receives a set of integrity rules ri to be enforced when each λ is called.

$$\text{declareActivity(n, } \mathbb{P}(\ \lambda, \mathbb{P}\ ri)) \quad \rightarrow \quad ac$$

Activity Association. Once an Activity a has been declared, to be used by a Project p, it must be associated to it. The operation bindActivity takes an Activity ac and a Project p and adds ac to the Activities of p. Multiple projects can use the same Activity.

$$\text{bindActivity}(ac,p)) \quad \rightarrow \quad ac \in p_{Ac}$$

Actors

Actors' handling operations allow the registration of new actors and modification of their properties are meant, while enforcing their integrity. These properties include contact information, as well as Artifact Directories representing their interests, knowledge and competences.

Events

Apart from the operation for creating new Events, there are two main operations for handling Events. The observe operation, allows an Actor, to be blocked while waiting for an Event to be raised. The raise operation provides a means for raising events and thus unblock observing Actors and feed awaiting subscribers.

The observe PRM operation takes two parameters, an Event e_o to be observed and a Time $time$. When the operation is called, the execution process calling it is blocked until an Event $e_p \mid e_o \simeq e_p$ is raised or until the time limit $time$ is reached. If a corresponding Event e_p is raised before $time$, e_p is returned to the caller, if not, nothing is returned.

Logs

Log Management Activity provides the Log creation operation createLog. This operation is meant to be exclusively called by PRM Activities in order to keep track of PRM usage. Therefore, in order to log the usage of an Activity, this operation should be called by the implementation of the Activity itself. Further, Rights to call this operation should be given only to Activities themselves. Information to be provided as parameter is described in Table 5.17. The following integrity rule must be enforced: the Artifacts passed as parameter must be registered. Indeed, a Log entry must contain valid data, thus, the Actor, Project, Process, Activity and operation must exist. This can be verified using the exists operation provided by each Activity.

Metrics

Metrics Handling relies on a set of operations providing features ranging from Metrics creation to Metrics execution. The newMetric operation allows PRM users to create Metrics. Its execution requires users to provide all related information as described in the previous section. This includes a name, a description, a MetricsViewPoint, MetricsSet, an occurrence expression and an execution expression.

Once Metrics are created, the registerMetric operation is used in order to make them available to the community. It is then ready to use by any community member needing it. To activate a Metric for a Project, the enableMetric operation has to be used. This operation takes a Metric and a Project as parameter. Once enabled, $oncall$ Metrics are automatically evaluated when targeted operations are called, and $ondemand$ Metrics are available to all users. symmetrically, the disableMetric operation can be used to disable a Metric for a Project, and enabledMetric checks if a Metric is enabled for a given Project.

Note that enabling and disabling Metrics requires some integrity rules to be enforced. Indeed, in order to enable a Metric for a Project, all activities part of the set $Ac_{metricExpression}$ which operations are triggered by the $measure$ execution expression must be available for the Project p :

$$\forall\, ac \in Ac_{metricExpression}, ac \in p_{Ac}$$

Further, all Metrics $m_i \in m_{referredMetrics}$ referred by the execution expression of m, must be activated for the project p:

$$\forall\, m_i \in m_{referredMetrics}, \mathsf{enabledMetric}(m_i, p)$$

Then, when disabling a Metric m for a Project p, one has to make sure that m is not used by any other Metric registered for p:

$$\forall\, m_{referred} \in m_{referredMetrics}, \nexists\, m_{other} \in p_M \mid m_{referred} \in m_{other_M} \wedge m \neq m_{other}$$

The last operation, executeMetric launches the evaluation of a Metric in the scope of a Project. This operation has to be called manually as Metrics are only evaluated on demand. The Metric is evaluated by resolving and executing its $metricExpression$ execution expression and evaluating recursively all Metrics it is depending on. The result of the Metric execution is then signaled as a SignalEvent encapsulating the Metric, the execution date and the computed result.

The result of the evaluation of the Metric can only be retrieved by an Actor a if he has the right to execute all the Metrics m_i the Metric m it is executing is composed of. Indeed, the Actor has to have the rights allowing him to call the different Activity operations needed to obtain the expected result. This depends on the project p within which a is executing m, the Role r a endorses and on the process π. Thus, for each m_i of m the following integrity rule must be enforced:

$$\forall\, ac.\lambda \in m_{calls}, \mathsf{allowed}(a, p, r, \pi, ac, \lambda)$$

Process

The PRM provides a set of operations enabling the construction of Processes. These allow Actors to build Processes by declaring them using the declareProcess operation, and declaring ProcessSteps using the declareProcessStep operation. To be able to declare a ProcessStep relying on a call to an Activity operation $A.\lambda$.

The different ProcessSteps are sequenced using the setProcessStepSequence operation. Conditional paths in the sequence can be defined by declaring special ProcessSteps using the declareConditionStep operation, while loops can be defined using the setLoopStep operation.

Project

Project Declaration Projects can be created through the PRM using the declareProject. At creation time information to fill all Project's attributes but provided Activities and sub-Projects must be provided. Thus, a Name must be specified, as well as an Acronym, a Description, the contact Actor, and a set of Topics the project is working on. If the Project possesses a parent Project, it can be specified at declaration time. All these Artifacts must exist before setting them as attributes unless the Project declaration fails.

Provided user defined Activities and sub-Projects are not known at Project creation time, for this reason these attributes can be left blank. If Activities have to be added, this can be done through the bindActivity operation of the Activity Management Activity. When a Project declares that it has a parent, the Sub-Project attribute of the Parent Project has to be accordingly modified in order to add the new child.

Contributors Declaration and Listing. The Project management Activity offers a set of operations for declaring, retrieving and verifying Project contributors. Indeed, registerContributor registers an Actor a as a contributor for a Project p. getContributors allows to retrieve all contributors to a Project p, and getProjects allows to retrieve all Projects an Actor a contributes to.

Rights

Right Declaration The declareRight operation is used to declare new Rights. A Right is the right to access a PRM operation in the scope of a Task t. The Activity ac, operations λ and t need to be passed as parameters to the operation and for integrity reason, these Artifacts have to exist at Rights declaration time.

Right Assignment Rights are assigned to Actors using the assignRight operation. Prior to do it, the Actor and Task must exist. After having assigned a Right rg to an Actor a, a must be allowed to execute calls to the operation λ of the Activity ac part of the Process and Role for which the Task t has been created:

$$\begin{aligned} \mathsf{assignRight}(rg, a) \quad \Rightarrow \quad \forall\, rg \in a_{Rg_{received}} \wedge \mathsf{allowed} \quad & (a, \mathsf{attr}_p(\mathsf{attr}_{task}(r)), \\ & \mathsf{attr}_{role}(\mathsf{attr}_{task}(r)), \\ & \mathsf{attr}_{\pi}(\mathsf{attr}_{task}(r)), \\ & \mathsf{attr}_{ac}(r), \\ & \mathsf{attr}_{\lambda}(r)) \end{aligned}$$

Right Verification When interacting with any interface of the PRM, the Rights are used as follows. A verification of the availability of an authorization for the Actor to call the selected PRM operation occurs. The Actors assigned Rights set is searched for a Right allowing the Actor to trigger the called operation in the scope of the Task he is executing. This is done through the PRM operation allowed which takes an actor a an Activity ac, an operation λ, a Role r, a Process π and a Project p as parameters and returns true if and only if a possesses a Right rg in the scope of p, π, r:

$$\begin{aligned} \mathsf{allowed}(a, p, r, \pi, ac, \lambda) \quad \Leftrightarrow \quad & \mathsf{containsProcess}(r, \pi) \wedge \\ & \mathsf{isRoleAssociated}(p, r) \wedge \\ & \mathsf{isRoleAssigned}(r, a) \wedge \\ & \exists\, r_{ac.\lambda} \in a_{p.\mathcal{R}\}_{received}} \end{aligned}$$

If such a Right exists, one has to verify if this Right is valid. Indeed one has to verify if the Right has not been disabled for the Project or revoked for the Actor. Finally, the operation returns if the Actor can call the operation or not.

Another version of the allowed verifies if an Actor has the Rights required to call every $ac.\lambda$ operation of a Process π. This operation relies on the previous one and has the same parameters but no Activity nor operation as those are extracted from the Process:

$$\mathsf{allowed}(a, p, r, \pi) \quad \Leftrightarrow \quad \forall\, ac.\lambda \in \pi_O, \mathsf{allowed}(a, p, r, \pi, ac, \lambda)$$

Roles

Role Declaration In order to create collections of Processes, so called Roles, which can be used in the scope of Projects and assigned to Actors, the PRM offers the declareRole operation. This operation gathers a set of Processes under a name and a description which are provided as parameters to the operation. For integrity reasons, the Processes must all have been declared prior to being included to a Role.

Role to Project Association The associateRole operation takes a Role r and a Project p as parameter in order to associate them and thus indicate that r can be used in the scope of p. After a call to this operation, r is added to p_R:

$$\mathsf{associateRole}(r, p) \quad \Rightarrow \quad r \in p_R$$

Role to Actor Assignment In order to assign a Role r to an Actor a in the scope of a Project p, one has to use the assignRole operation. It takes as parameter r, a, and p and if these Artifacts exist and if r is associated with p, adds r to the set of Roles of the Actor for the Project a_R^p.

$$\mathsf{assignRole}(r, a, p) \quad \Rightarrow \quad r \in a_R^p \Leftrightarrow \mathsf{isRoleAssociated}(r, p)$$

Further, once r is associated with a for p, a must be assigned the Rights to execute all the Processes of r. This has to be done by creating associated the Tasks for each Process $\pi \in r_\pi$ using newTask as follows:

$$\forall\, \pi \in r_\pi, \mathsf{newTask}(a, p, \pi, r)$$

Then for each Task t of the set of newly created Tasks T, the declareRight operation has to be called in order to create corresponding Rights for every operation involved in t's process and then the assignRight operation has to be called in order to assign the created Right to a:

$$\forall\, t \in T, \forall\, ac.\lambda \in (\mathsf{attr}_{process}(t))_O, \mathsf{assignRight}(\mathsf{declareRight}(ac, \lambda, t), a)$$

Role Verification Three operations enable the execution of different verifications related to Roles. The first one,containsProcess takes a Role r and a Process π as parameters and verifies if π is part of r:

$$\mathsf{containsProcess}(r, \pi) \quad \Leftrightarrow \quad \pi \in r_\pi$$

The second operation, isRoleAssociated, takes a Project p and a Role r as parameter and verifies if r has been associated with p:

$$\mathsf{isRoleAssociated}(r, p) \quad \Leftrightarrow \quad r \in p_R$$

Finally, the last operation, isRoleAssigned, verifies if a Role r has been assigned to an Actor a in the scope of a Project p:

$$\mathsf{isRoleAssigned}(r, a, p) \quad \Leftrightarrow \quad \mathsf{isRoleAssociated}(p, r) \wedge r \in a_R^p$$

Tasks

Task Declaration A Process π can only be executed by Actors having been assigned π. The assignation of π to an Actor a is done through the creation of a new Task for a Project p, and assignment withdrawal is done through the changeTaskStatus operation.

The newTask operation creates a new Task by associating an Actor to a Process of a Role for a Project. These four Artifacts have to exist prior to the creation of the Task. Another parameter allows defining the type of the task. No other information is necessary at Task creation time, as the Actor creating the task and the creation date are known and can be automatically retrieved, the status is set to "assigned" automatically, and the start and end dates are unknown.

When assigning a Process π to an Actor a in the scope of a Role r and Project p to create a Task t one has to make sure that π is part of r, that r is associated with p, and that r is assigned to a in the scope of p. Further,a must be registered as a contributor to p. Thus if isContributor(a, p) returns false, registerContributor(a, p) must be called. Finally:

$$\begin{aligned} \text{newTask}(a, p, \pi, r) \quad \Leftrightarrow \quad & \text{isContributor}(a, p)\ \text{containsProcess}(r, \pi) \wedge \\ & \text{isRoleAssociated}(r, p) \wedge \text{isRoleAssigned}(r, a, p) \end{aligned}$$

Task Execution Once a Task is assigned to an Actor, it can execute the underlying Process using the executeTask operation. This operation executes the first ProcessStep of the Process, changes the status of the Task to *ongoing* and returns the result of its execution as an Artifact Directory. The next Step can be obtained using the nextStep operation. Then the following steps can be executed using the executeStep operation. Each of these two operations take a Directory as parameter which provides all needed data to feed the operation called by the ProcessStep. At execution time, if the execution is not possible it is canceled.

When the Task is executed it changes its status from *assigned* to *ongoing* and fills the startDate attribute of the Task. When the Process execution is finished, the status of the Task is changed to *completed*, if the ProcessType of the underlying Process is set to *once-off* or *assigned* if it is a *recurring* Process. Then the endDate attribute is filled to keep track of Task duration.

When an Actor a tries to execute in the scope of a Project p a Task t involving an Activity operation $ac.\lambda$, one has to make sure a has still the Right to call it. Thus the following rule has to be enforced when executing the Task:

$$\text{execute}(o_{Activity}) \quad \Leftrightarrow \quad \text{allowed}(a, p, r, \pi, ac, \lambda)$$

Task Status Change The status of a Task can be changed using the changeTaskStatus operation. This operation can be called automatically or manually. Task execution can change the status to *ongoing, assigned, paused* its status or *completed* automatically The only possible manual changes are to set the status to *paused*, *ongoing* or *abandoned*. Note that if a Task is *completed* or *abandoned* its status cannot be changed anymore. In such a case, the Rights corresponding to the underlying Process steps have to be revoked from the Actor in order to avoid Process execution outside of the scope of the Task.

Position in Process At any time, current position in the Process can be retrieved as a ProcessStep using the currentStepOfTask operation and providing the Task as parameter. This information is useful in order to keep track of Task completion.

5.2 Model Properties

In this last section we discuss the PRM model presented in this chapter. In a first step we provide an overview of the strengths and advantages the model offers. However, the model also implies both

organizational and technical constraints that need to be highlighted. Indeed, projects have to create new roles for managing the PRM and have to adapt the tools they are using to take advantage of the features of the model. Thus, in a second step, we precise these constraints.

5.2.1 Model Strengths

The PRM Model provides multiple benefits for building distributed Information Systems bound to F/OSS environment constraints. We detail them in this section and Table 5.22 summarizes them.

Strength	Description
Data uniformity	Artifacts-Attribute to express any kind of information and resource.
Information manipulability	Flexible lookup, combination, substitutability of Artifacts and Artifact Directories.
Information integrity enforcement	Integrity enforced through Artifacts and Integrity Rules
Activity centered approach	Information System description through Activities and available operations.
Process streamlining	Description of processes, their sequence
Events Management	Provision of a synchronization and a communication means for Activities implementation and Processes
Transversal Metrics management	Model includes Metrics handling designed for distributed Information Systems
Core activities handling	Means to handle core activities involving Actors and Projects.
Community Management	Handling of contributors' interests, knowledge and competencies.
Extensibility	Information System extension through Activity and Artifact addition
Interoperability enabler	Interoperability through common Artifacts, Activity interfaces and Processes.

Table 5.22: Summary of PRM Model's strengths

Data uniformity The PRM provides through Artifacts a generic and a uniform means for representing and describing any kind information. Indeed, Artifacts can be used to represent content, community resources or any other type of resource. Artifacts can be matched in order to compare them. Artifact differentiation depends on their type, on the type of their Attributes and the list of their Attributes. Artifacts provide thus a powerful means for information sharing on distributed Information Systems.

Information manipulability Powerful means for manipulating Artifacts are provided by the PRM Model. For instance, they can be combined through Attributes. Artifact Directories and Artifact expressions enable the definition of expressive data structures. The substitutability relation goes beyond the usual comparison relation and provides extended flexibility to the PRM Model, which relies on it in order to provide a flexible artifact lookup of Artifacts.

Information integrity enforcement Both Artifacts integrity and Activity integrity are enforced by the PRM model. Indeed, as Artifacts are types, they only accept values or Attributes of the specified type. This ensures that only understandable information is stored in an Artifact. Further, mandatory Attributes of Artifacts have to be specified, which ensures information availability. Finally, as the PRM Model enables the definition of integrity rules and their association to the different operations of registered

Activities, this provides a means for ensuring that once created, Artifacts' integrity is ensured during their lifetime.

Activity centered approach The PRM Model proposes to handle distributed Information Systems as sets of Activity interfaces. These interfaces provide a list of operations they are able to handle. They can be implemented, shared along with the Artifacts they use.

Core activities handling The PRM Model handles all core activities needed for building F/OSS Information Systems: community, project, process, rights, roles, metrics, tasks, events, log, artifact and activity management.

Process streamlining consists of defining how processes have to be integrated in a global view. The benefits of streamlining processes is to see how they impact on each other, discover bottlenecks, and know where effort has to be put to improve the F/OSS Process. To achieve this, the PRM provides the Process Artifact to express F/OSS project processes. These Processes model how project activities are linked, i.e. how calls to their operations have to be organized. Processes can be chained and synchronized using Event elements. This synchronization can be achieved inside a project but also on a cross-project manner. Indeed outside Projects can ask to be notified of the end of a specific process run by a partner project to start their own Processes. For instance, a testing community may wish to be automatically informed of beta releases produced by different projects to help them test their products.

Events Management The PRM model provides through Events a generic means for inter-Process communication and synchronization. It can be used by core PRM activities as well as by user defined activities.

Transversal Metrics management Metrics allow to describe measures to be taken within F/OSS projects. F/OSS project activities can be distributed with no central point of control. In this context, metrics can involve multiple activities. Being able to evaluate transversal metrics involving these distributed activities is needed to achieve thorough analysis of F/OSS information systems, evaluate their processes and thus be able to improve them.

Community Management The PRM goes beyond usual community handling which lists the different contributors and provides common information about them. The PRM aims at considering community members interests, knowledge and competences when enrolling contributors, attributing tasks to members, etc. Further, the PRM provides a means to know exactly which Actors contribute to a project, and know what are their tasks, obligations and rights.

Extensibility PRM extensibility is ensured by the possibility to declare new Artifacts, Activities, Processes, Events and Metrics. For instance, a new Activity dealing with e-learning could provide an e-course Artifact and operations for defining the topics of the course, for associating a community member as teachers, and could embed an integrity rule defining the competencies threshold for accepting a community member as a teacher for the course. We describe how the PRM can be extended to adapt to specific domains in Chapter 6.

Interoperability enabler Interoperability depends on data format agreement, data understanding and data handling. The PRM provides a means for ensuring F/OSS Project interoperability by design through the use of Artifacts, Activities and Processes. These elements can provide a common ground of understanding to F/OSS projects as they model the data used by projects, the different ways this data can be

handled and the processes using it. Projects can agree on common artifacts, compare their activities along with involved integrity rules and Processes. As activity implementations are independent components and as they provide the same features, two implementations of the same activity are substitutable. Thus, projects can choose indifferently among existing implementations of activities, which can help adapting to external issues such as the ones projects providing external implementations of activities can be subject to.

Thus the PRM model provides a means to fulfill both information and process management requirements listed in Chapter 4.

5.2.2 PRM usage implications

In order to benefit of the strengths the PRM model offers, some organizational, i.e. new roles to be handled, and technical constraints have to be considered.

Presentation of new PRM roles

As the model introduces new elements and forces the description of previously unclear ones, new roles, tasks and responsibilities have to be defined in order to handle them. Indeed, in order to use efficiently the PRM model, projects must have contributors undertaking particular support roles. Table 5.23 lists these new roles, provides information about who has to undertake them and describes them briefly. Involved people include experts of the different fields, or even the community itself.

Role	Who	Description
Artifact Manager	Board of Experts	Defines the structure of used Artifacts
Activity Manager	Board of Experts	Defines Project's Activity interfaces
Process Manager	Board of Experts	Describes Project's Processes
Roles Manager	Process Expert	Defines the Different Roles used in the Project
Activity Dependency Manager	Activity Implementor	Describes the needs of each Activity in terms of other Activities' operations
Project Describer	Expert	Description of Project for improved contributor match up
Community Describer	Everyone	Description of contributors
Task Manager	Expert	Associates tasks to community members
Task Evaluator	Expert	Evaluates how contributors do their job

Table 5.23: New Roles introduced by the PRM

Artifact Manager For instance, the Artifacts to be used within the project and the information provided by each Artifact must be chosen. Mandatory and optional attributes must be selected, and substitutability rules must be defined. A specific role undertaking this task must be defined, and the contributors in charge of it must be aware of the different situations in which the Artifact can be involved. Further, this role must also control Artifact's conformity with project needs, as well as conformity with users needs and correct or complete Artifacts accordingly to observed issues. As Artifact definition is central to Project management and Activities, the Role should be supported by a Board of Experts of the different Activities involved.

Activity Manager This role needs information about the different activities of the project. Thus a role describing them is needed. It has to gather information about the different operations offered to

the community by the project, and then create the corresponding Activity interfaces. The operations of these interfaces have to use the different Artifacts chosen together with the Artifacts description role. This activity description also covers the definition of the integrity rules related to each operation of these activities, their association to the operations and referencing for information purpose. This role should be undertaken by a Board of Experts. Further it should be supervised by the Board of Directors of the project, if any, as it is their role to define the different activities a project is involved in.

Process Manager Another Role has to describe precisely the different Processes of the Project. These processes define how the Project is conducted, and serve as the background for defining and associating tasks to the different contributors. Further, having a clear description of the different processes helps keeping knowledge, and ease the introduction of new contributors as they can see how their task integrate with the whole process. The Process definition Role has also to verify if Project's processes are correctly connected and thus that the F/OSS process is streamlined. If issues are detected, the Processes must be modified accordingly. This is a key role for PRM enabled projects, which requires from contributors undertaking it a complete knowledge of Project's Processes and Activities.

Role Manager As Processes are bound to Competences, a role has to describe project Roles, which are needed for executing the different processes of the project. This implies a thorough analysis of each Process and the evaluation of needed competencies in order to be able to handle the process, and react accordingly if something goes wrong.

Activity Dependency Manager For each activity implementation, a contributor has to define if operations provided by other Activities are needed. The implementations of these Activities can be local or external to the Project. This information can help finding substitute projects if the Activity implementation used by the Project is part of a stalled Project, or if there is no more support for this Activity. In such a situation, another Project offering the same Activities can be found.

Project describer The PRM considers Projects as a central Artifact. Other information such as Tasks are directly bound to them. The participation of community members to a Project depends greatly on the Topics covered by the Project. In order to foster participation, Projects and Tasks need thus to be precisely described. While Process Managers describe the Processes the Tasks are related to, a role describing the Project and keeping this information up to date is needed. This Role is traditionally mixed with the role of webmaster. Indeed webmasters often include Project descriptions on websites. Nevertheless, in order to feed the PRM and ensure precise match up with Actors, this Role should be extracted. Further it should be undertaken by a community member aware of Projects details, orientation and Processes.

Finally, the community, as a resource, needs also to be managed. This implies the following three roles.

Community Describer In order to match Actors with Tasks and Projects, the PRM also needs information about the interests and knowledge of the community members. This information should be provided by the contributors themselves.

Task Manager Tasks corresponding to the different Processes of the Project must be associated to the different community members. While the PRM helps finding adequate Actors, a responsible person must select them depending on its knowledge of the underlying Processes, their complexity and its knowledge of the community and its members. Thus the Actor having this Role must be an expert in these fields.

Task Evaluator Then, when a task is done, another role has to evaluate what has been done in order to update information about the Competences of the different contributors. As such, depending on the Task's nature, the evaluator should be an expert, mastering the underlying Process or able to evaluate.

Some of these roles are not usual for the F/OSS community as they are describing the structure of the project. As such, this implies that these roles as well as the strengths of the model must be clearly explained to the community in order to foster motivation and thus find contributors willing to participate in these tasks. Next chapter will cover some examples of how the model can be used and bring benefit to the Projects using it.

Technical Constraints

PRM usage also implies some technical constraints which are summarized in Table 5.24.

Constraint	Description
Tools adaptation	Need to implement Activity interfaces, respect Artifacts format
Information gathering	Implies community members involvement and automation
Cross project activity sharing	Activity interfaces and Artifact formats sharing and discovery

Table 5.24: Summary of PRM Technical constraints

Tools adaptation. A first technical constraint is the need to adapt tools in order to make them use the PRM. This includes the need to implement Activity interfaces in order to provide the different operations while respecting their signature. Further it also implies the need to respect the format of Artifacts used by the Project, and integrate Artifacts and the PRM model with these tools.

Information gathering. Using the PRM Model and Artifacts, implies that information that may have been hidden may have to be gathered in order to complete Artifacts and ensure their integrity. Indeed, mandatory fields of Artifacts cannot be left empty, so, this information must be gathered. This task can be undertaken through the involvement of community members and roles such as described in the previous section or automatic information gathering and Artifact completion when applicable. Again, automation involves the adaptation of existing tools.

Cross project activity sharing. Finally, in order to easily share Activities among Projects, Projects need to have a means to publish their Activity implementations along with their Activity interfaces and corresponding Artifacts. Similarly, they should have a means to discover Activities provided by other Projects in order to integrate them if needed. For instance imagine a Project aiming at providing a translation service for other projects. Any project willing to integrate this external activity to its process should then be able to find this activity and call the operations it provides. To build such cross project activity sharing, multiple solutions can be considered such as centralizing PRM's core Activities or providing them as a web service. Approaches inspired from Service Oriented Architectures such as UDDI [135] or other registry services such as RMI [178] are possible.

Chapter 6

Extending the PRM Model

As defined previously, the PRM model is meant to be generic and extensible. The main reason for this feature being that in the context of totally decentralized information systems such as F/OSS information systems, there must be a way to extend the process with new activities and integrate them seamlessly with existing elements. These new activities have to be able to handle new types of resources. Multiple questions may arise concerning the operations involved in this new activity, the resources involved, the new resources to be created, the information to be left accessible to other activities or the integration of the activity with existing ones. In this chapter we present first the problems which may be encountered when extending the model if not doing it properly. Then we propose a simple method for extending the PRM model with new Activities and Artifacts, highlighting the *does and don'ts*.

6.1 Extension shelves

While the PRM model provides all needed tools to extend it to handle any activity and resource, multiple problems can occur if additional Activities and Artifacts are not correctly defined.

Scoping A first possible problem is the lack of a clear definition of the purpose of Activities, and of their domain, which can lead to Activity and thus responsibility mixing. Each Activity has to handle one specific domain and provide the resources and operations to achieve this task. The borders with other related activities have thus to be well known and described.

The same problem applies to Artifact definition as Artifacts have to represent a selected resource. The attributes of each Artifact have to be strictly related to this Artifact and no information should be merged. A clear scoping of Artifacts is needed for information responsibility management and for the ease of information retrieval. If information is mixed, errors or inconsistencies are possible, especially in a totally decentralized environment.

Thus the scoping of Activities and Artifacts must be precisely defined in order not to be too large nor too tight.

Information famine . The goal of an Activity is to provide all needed Information to Activity users through operations as well as to provided all needed means to act in the scope of the Activity. Providing too little information or limited means to use provided information makes the Activity useless as it tempers the ability to interact with it.

Similarly, Artifacts have to provide meaningful and adequate information through their attributes. Again, a severe lack of information (attributes) can temper the usefulness and usability of an Artifact.

Imagine for instance an Artifact used to represent Linux content units (or packages) which would lack description information. There would be no use for such an Artifact.

Thus the information famine problem is directly related to the scoping problem seen above as both Activities and Artifacts must provide the information related to the purpose of their existence.

Integration limitation Activities and Artifacts are not meant to be used alone. Activities are supposed to communicate through Processes and Artifacts are meant to be used by other Activities to benefit from existing information.

In this context, a possible problem which may appear when defining these elements is to limit provided information and behaviors to a minimal set the creator *thinks* that it has to be provided, forgetting to externalize information that users, be they Activities of Actors *may need* to access. Again, the example of Linux content units applies here: not providing dependency information, makes the testing activity unable to do its work as this information is required to evaluate the consistency of a package, by checking the non existence of cyclic dependencies.

Lack of integrity enforcement While providing integrity rules is not mandatory for the PRM to function correctly, the lack of integrity enforcement implies information poorness that may temper the use of both Activities and Artifacts. One of the goals of the PRM is to ensure information integrity. Information which verifies integrity rules is useful and guarantees its quality. Inversely, information which has not passed even a single integrity check may be difficult to use in a context where this integrity is mandatory. An example of simple integrity rule is that only content units which have passed all checks can be distributed.

Thus the information that such a rule has been enforced has to be explicit to Activity and Artifact users. It has to guide their choice of PRM elements they are using. Further, for each Activity behavior and each Artifact, an analysis has to be done in order to ensure that in the scope of the element and of its integration with other Activities, no key integrity rule is forgotten.

6.2 Extension method

The previous section highlighted possible problems that may appear if the definition of Activities and Artifacts is inaccurate. This implies that guidelines need to be provided to PRM users to correctly achieve their PRM model extensions and thus avoid further problems. The method to ensure that these extensions handle required and useful information include the steps listed in Table 6.1:

Step	Description
1.	Activity domain definition
2.	Activity integration
3.	Highlight of Activity dependencies
4.	Selection of information provided by the Activity
5.	Selection of resources involved in the Activity
6.	Selection of Integrity rules
7.	Activity refinement

Table 6.1: Steps involved in PRM extension.

Activity domain definition. The initial step of the method consists of the provision of a clear definition of the Activity domain. This definition has to indicate the purpose of the Activity along with the

behaviors the Activity is supposed to enable and the behaviors which are out of its scope. This definition has thus to provide the background and limits which will help achieving the next steps of the method.

Activity integration Once the definition of the Activity domain is done, the second step consists of integrating the new Activity with other existing and related Activities. The aim of this step is to highlight the information flows existing between the different Activities or which may exist with them. Thus, a fundamental part of this step is to imagine the potential of the new Activity. Projections are needed, then when these information flows are detected, they should be submitted to the contributors of the other Activities, to be sure that the ways communication is predicted is correct and that nothing is missing.

Definition of Activity dependencies . When registering the newly created Activity for a Project, Activity dependencies detected during the previous step must be available for the Project. Then, these dependencies indicate where the information comes from, and thus indicates the integrity rules which have to be enforced. This enables Activity analysis to detect for instance Activity incompatibility which may result from incompatible domains being handled or to detect problems due to incomplete sets of integrity rules, which may involve slight adjustments.

Selection of information provided by the Activity. The information flows which have been highlighted in the Activity integration part of the method not only help choose the Activities the new Activity is depending on, but also indicate the information which has to be provided to other Activities. Based on this analysis the operations of the Activity have to be selected.

Selection of resources involved in the Activity The different operations provided by an Activity as well as the Activities on which the Activity is dependent suggest the resources to be used, i.e. Artifacts, which are involved in the Activity. Some of these are provided by other Activities, but some resources may have to be provided by the Activity itself. In such a case, corresponding Artifacts have to be defined and correct attributes have to be chosen in order to ensure that all needed information to have a meaningful Artifact is provided. As for Activities, the scope of each Artifact has to be defined as well as the corresponding limits. In general, only information directly bound to the description and use of the Artifact should be set as an attribute.

Selection of integrity Rules The next step consists of choosing adequate integrity rules for both Activity operations and Artifacts. Activity related rules must ensure the integrity of the information held and provided by the Activity accordingly to the previously defined domain, and to the various information flows which have been highlighted.

Activity refinement. Once all previous steps are done and the Activity and Artifacts are ready to be registered, the whole Activity should be analyzed again, to be sure that nothing has been forgotten.

6.3 Additional elements registration

Once additional Artifact types and Activities are defined they must be registered for the project which will have to use them. The operation is simple. Each Artifact Type T must be registered using the registerArtifactType PRM operation and each Activity A must be declared using the declareActivity PRM operation.

Note that in order to register an Activity A_i if any integrity rule ir_i of an operation $A_i.\lambda_i$ needs access to an operation $A_j.\lambda_j$ of an Activity A_j, A_j must be registered first for dependency reasons.

Chapter 7

PRM In Action

Now that we have presented the PRM, it is needed to show how this model can be used in order to support the F/OSS process. Mainly the PRM is meant to provide means to solve existing issues by enabling behaviors and actions previously difficult or not possible to formalize. It enables the declaration of Processes, the definition of Roles having to handle them, their execution through Tasks and their measurement through Metrics but it also supports the creation of Projects and highlights the Activities and resources to be considered.

Process management highly depends on the maturity of considered projects. These can be small, established or optimizing. Small projects often adopt an *ad hoc* process management, with no clearly defined roles, etc. While such an approach is attractive in the beginning, as process management is time consuming and requires specific competences, it is not viable in the long run. The imperatives and goals of each maturity level differ thus as follows:

- **Small projects.** This level includes all small and starting projects, where process description, and thus process management, is incomplete. At most the processes are performed, mainly in an *ad hoc* manner. The aims of such projects are to grow fast, to involve more people, and present new ideas.

- **Established projects.** This level gathers all established and widely used projects. They include process management to some extent, the roles of community members are clearly defined. Such projects aim at stabilizing their product, at involving more people, and especially more users.

- **Optimizing projects.** The last level consists of large and advanced projects. Their processes are clearly defined and managed and so is their community. They focus on analyzing processes, optimizing them and achieving predictions in order to improve their efficiency. Such projects, aim at becoming *de facto* standards, and at reaching the enterprise environment, and more precisely, at being included in enterprise processes.

Projects never start big, they need guidelines helping them to be prepared for the tasks to be endorsed. As projects grow in size, new challenges appear and more attention has to be given to information management. A central question for newly created F/OSS projects is thus to know how to grow efficiently, i.e. how to become an established project, then an optimizing one. This is required to be efficient and to provide high quality assurance at each level. The PRM helps structuring F/OSS projects by providing guidelines and enabling project-wide reasoning and process improvement. In this chapter we illustrate how the PRM can be used to this extend.

The following scenario illustrates how the PRM can be used for structuring a F/OSS project. It highlights the different steps involved in the creation of a project, such as the declaration of the project, the declaration of the activities and artifacts to be handled, the registration of involved people as well

as the definition of involved processes, creation and distribution of roles. Then we illustrate how such a structure supports project's evolution from both organizational and activity evolution viewpoints.

7.1 Scenario Overview

In the context of a object oriented programming class, a group of teaching assistants – Alex, John, Alice and Shelly – has prepared a set of courses for presenting the OO paradigm and design patterns. As they consider these courses as potentially useful to other people, they decide to propose them as a complete object oriented course to the open source community. They have ideas to improve them but the small team needs man force and competencies to implement these new ideas. Indeed, they want other people to contribute correcting existing courses, improving them, and also proposing new topics, etc. They decide thus to create a project around their idea and release these courses under a F/OSS license. They hope people will join them to create a community around their project and will share knowledge.

The creation of such a Project may seem simple at first sight, however the small team wants this project to grow and be able to face organizational changes easily as teaching assistants change relatively often. Further they also want to be able to extend their project with new Activities. For instance they want to be able to propose exercises related to the courses they propose.

Imagine that Alex, John, Alice and Shelly decide to use the PRM to guide them in the definition of their Project. Their work will make them declare an Activity for handling course Artifacts along with related Lessons. Tables 7.1, 7.2, and 7.3 describe these Artifacts and Activity. They will also have to declare the processes and distribute roles among project's participants. In the following sections we will overview all the different steps they have to follow for creating such a Project from scratch. These steps will help them organize their small community, declare existing processes, select the activities and artifacts to be involved, creating required roles for the project and assign them to contributors having been first registered as community members of the project.

7.2 Step 1: Actors Registration

As a first step, the team needs to be registered with the PRM in order to indicate that these people exist and can interact with the PRM, can receive tasks, etc. Thus the following calls are done to the PRM Activity managing Actors: *ActorActivity*.

$$\begin{aligned}
&\text{ActorActivity.registerActor(Directory}_{Alex}) -> \text{Alex}\\
&\text{ActorActivity.registerActor(Directory}_{John}) -> \text{John}\\
&\text{ActorActivity.registerActor(Directory}_{Alice}) -> \text{Alice}\\
&\text{ActorActivity.registerActor(Directory}_{Shelly}) -> \text{Shelly}
\end{aligned}$$

The directories Directory_{Alex}, Directory_{John}, Directory_{Alice} and $\text{Directory}_{Shelly}$ contain the information needed to build an Actor Artifact for each participant to the project. As defined in section 5.1.2, each of these directories contains a ContactInformation Artifact and three sets of Topic Artifacts representing the knowledge, interests and competencies of the corresponding Actor. As the Actor has been just registered, his competencies have not been evaluated yet. For instance Directory_{Alex} contains the following information:

$$\begin{aligned}
&\text{ActorActivity.getContactInformation("`Alex"', "`"', "`alex@opencourses.org"') -> ContactInformation}_{Alex}\\
&\text{Directory}_{Alex} <<_{contactInformation} = \text{ContactInformation}_{Alex}\\
&\text{Directory}_{Alex} <<_{interests} = \{\text{"`Project Management"'}\}\\
&\text{Directory}_{Alex} <<_{competencies} = \{\}\\
&\text{Directory}_{Alex} <<_{knowledge} = \{\text{"`Project Management"', "`Teaching"', "`Object Oriented Programming"'}\}
\end{aligned}$$

7.3 Step 2: Creating the project

The second step consists in creating a Project through the PRM by interacting with the Project Management Activity *ProjectActivity*. As the initiator of the idea, Alex registers thus the Project *OpenCourses* with the PRM as follows:

```
ProjectActivity.declareProject(   "'Open Courses"',
                                  "'OC"',
                                  "'A Project offering Courses under an Open Source License"',
                                  Alex, Course,
                                  "'http://www.opencourses.org/"',
                                  null) -> OC
```

Having registered the Project, Alex becomes its owner. This means that he receives the core roles which will enable him to build the Project through the PRM.

7.4 Step 3: Core Processes, Roles and Tasks

In order to define how the PRM has to be used, core Processes and Roles are required. The aim here is to have the procedures indicating how people can interact with the PRM in the scope of the Project. Every Project is free to define the details of theses processes and roles. We propose here a possible definition.

7.4.1 Core Processes

Change Ownership. This Process enables the change of ownership of the Project if the owner plans to quit the project.

$$\pi_{ownership} = ProjectActivity.setContact(Actor)$$

Project Modification. This process allows changing information about the project.

$$\pi_{projectModification} = (ProjectActivity.setTopics(\mathbb{P}\, Topic) \,||\, ProjectActivity.setDescription(Text) \,||\, ProjectActivity.setContact(Actor));$$

Artifact Declaration. This Process is used to register new types of Artifacts for further use within Activities.

$$\pi_{artifactDeclaration} = ArtifactActivity.registerArtifactType(TypeDirectory, ValuationDirectory, \mathbb{P}\, I\mathcal{R}, Project);$$

Activity Declaration. This Process creates a new Activity as a set of operations bound to IntegrityRules.

$$\pi_{activityDeclaration} = ActivityActivity.declareActivity(Name, \mathbb{P}(\lambda, \mathbb{P}\, I\mathcal{R}));$$

Activity Association. This Process binds a created Activity to a Project. This allows to share Activities among Projects.

$$\pi_{activityAssociation} = ActivityActivity.associateActivity(Activity, Project);$$

Process Definition. The Process of defining Processes consists in the creation of the different ProcessSteps, the declaration of the Process, then the sequencing of the different steps. Note that a Process can be used as a ProcessStep to chain different Processes.

$$\begin{aligned}\pi_{processStepDeclaration} \quad = \quad & (ProcessActivity.declareProcessStep \;||\\ & ProcessActivity.declareLoopStep \;||\\ & ProcessActivity.declareConditionStep);\end{aligned}$$

$$\begin{aligned}\pi_{processDeclaration} \quad = \quad & \pi^{+}_{ProcessActivity.processStepDeclaration};\\ & ProcessActivity.declareProcess(Name, ProcessStep_{first}, Directory_{interests},\\ & Directory_{knowledge}, Directory_{competences}, ProcessType);\\ & ProcessActivity.setProcessStepSequence(ProcessStep_{previous}, ProcessStep_{next})^{-};\end{aligned}$$

Role Definition. Processes indicating what the responsibilities of a Person are gathered as Roles.

$$\pi_{roleDeclaration} \quad = \quad RoleActivity.declareRole(Text_name, Text_description, \mathbb{P}\, Process);$$

Actor Registration. This Process allows the registration of new Actors with the PRM.

$$\pi_{actorRegistration} \quad = \quad ActorActivity.registerActor(Directory_{Actor});$$

Actor Invitation. Registered Actors can be invited to become community members of Projects thanks to this Process consisting of proposing a Role to an Actor.

$$\pi_{actorInvitation} \quad = \quad ActorActivity.proposeRole(Role, Actor, Project);$$

Role Acceptance. Actors can retrieve Roles having been proposed to them. Then they can accept or reject them. This is illustrated by the following Process.

$$\begin{aligned}\pi_{invitationAcceptance} \quad = \quad & ActorActivity.getRolesProposed(Actor, Project);\\ & ActorActivity.answerRoleProposition(Role, Actor, Project, Boolean);\end{aligned}$$

Contribution Proposition. Actors can apply for a Role through this Process. This allows Actors to be proactive while searching for Projects where they could help.

$$\pi_{contributionProposition} \quad = \quad proposeRoleContribution(Role, Actor, Project);$$

Actor Binding. This Process registers an Actor as a Contributor of a Project.

$$\pi_{actorBinding} \quad = \quad ProjectActivity.registerContributor(Actor, Project);$$

Task Attribution. Once a Role has been accepted by an Actor or once an Actor has proposed himself for a Role, the Role can be assigned to him. Role assignment results in the creation of corresponding Tasks. Further the Role can be unassigned as any time, which results in the deletion of corresponding Tasks.

$$\begin{aligned}\pi_{taskAssignment} \quad = \quad & RoleActivity.assignRole(Actor, Project, Role) \;||\\ & RoleActivity.unassignRole(Actor, Project, Role);\end{aligned}$$

Actor De/activation. As Actors can leave the project or can become inactive while still being a member of the community of a Project, this Process allows to declare Actors as being active or inactive.

$$\pi_{actorActivation} = (ProjectActivity.activateActor(Actor, Project) \;||\; ProjectActivity.deactivateActor(Actor, Project));$$

7.4.2 Core Roles

Bootstrap. This is the central Role of the Project. When the project is created it includes all other core Roles. The Actor creating the Project receives thus this Role, which enables him to bootstrap the use of the PRM by registering people, assigning tasks etc.

Project Manager. This role enables the person it is assigned to to modify contact information about the Project, it includes the process $\pi_{projectModification}$.

Community Manager. This role enables the Project to keep control over its community, by baning undesirable Actors or inviting need ones. Involved Processes are: $\pi_{actorBinding}$ and $\pi_{actorActivation}$.

Activity Manager. The Activity management role handles the declaration and association of new Activities and Artifacts to Projects. This Role is central, as it enables Projects to evolve and to propose new features to the community. It involves two processes: $\pi_{artifactDeclaration}$, $\pi_{activityDeclaration}$ and $\pi_{activityAssociation}$.

Roles Manager. This role is responsible for the definition of the processes of the Project and for their gathering as new Roles. The two key processes it has to execute are thus: $\pi_{processStepDeclaration}$, $\pi_{processDeclaration}$ and $\pi_{roleDeclaration}$.

Tasks Manager. Defined Roles have to be distributed to the different community members but they also need to be removed from a community member once he or she retires. These are the processes the Task Manager Role has to handle: $\pi_{actorInvitation}$, $\pi_{taskAssignment}$.

User. Any user can receive Roles propositions from Task Managers. This involves the following two Processes: $\pi_{invitationAcceptance}$, $\pi_{invitationAcceptance}$. Note that any Person registered on the PRM obtains this Role.

7.4.3 Core Tasks

To be usable, the PRM requires that these core Roles be assigned to the community members of the Project. Here follows how they are dispatched in our scenario.

As Alex created the Project, he obtains the Bootstrap Manager Role. It allows him to associate the other Roles to the other members of the Project. Alex wants to keep control over the information which is published about the Project, he assigns thus the Project Manager Role to himself. He also wants to have contact with the community and be able who will participate in the Project. Thus he also keeps this role.

As John is a PRM specialist Alex assigns the Activities Manager Role to him. Roles Management role is given to Alice because she has expertise in process management and as Shelly is good at evaluating people's competencies and at organizing teams, she receives the Tasks management Role.

As a bootstrap manager, Alex executes the processes $\pi_{actorBinding}$ and $\pi_{taskAssignment}$ in order to create the core community of the Project and assign to it the core tasks. This results in the following calls to the PRM :

ProjectActivity.registerContributor(John, OC)
ProjectActivity.registerContributor(Alice, OC)
ProjectActivity.registerContributor(Shelly, OC)
RoleActivity.assignRole("‘Project Manager"’,Alex, OC);
RoleActivity.assignRole("‘Community Manager"’,Alex, OC);
RoleActivity.assignRole("‘Activities Manager"’,John, OC);
RoleActivity.assignRole("‘Roles Manager"’,Alice, OC);
RoleActivity.assignRole("‘Tasks Manager"’,Shelly, OC);

7.5 Step 4: Project Specificities

Once the Project is created and core Roles have been assigned, specific Artifacts and Activities can be added. In the case of our scenario, the Course management Activity needs to be declared together with the Artifacts it has to handle, i.e. courses and lessons. John is responsible for this Task, the next sections describe the Artifact types and Activities he creates.

7.5.1 Artifacts Definition

Any course provided by the Project is described using two different Artifacts types: Courses and Lessons. A Course is an envelope gathering Lessons around a set of Topics. Table 7.1 details the Attributes of the Course Artifact. Each Course has a name, a short name, a description, a list of Topics it is related to and a URL where additional information can be found. For instance, a Course can be named "‘Object Oriented Programing"’, its short name can be "‘ProgObj"’, its description can be "‘Introduction to object oriented programming, and object oriented architectures and methodologies"’, the Topics may include "‘Java"’, "‘programing"’, "‘modeling"’, etc.

Attribute Name	Attribute Type	Description
Title	Text	Title of the Course
ShortName	Text	Short name of the Course
Description	Text	Description of the Course
Topics	$\mathbb{P}$ Topic	Topics of the Course
URL	URL	URL of the Course

Table 7.1: Course Artifact Attributes

Each course is made of multiple Lessons. The Lesson Artifact represents a single lesson given in the scope of a Course. Table 7.2 details the Attributes of the Lesson Artifact. Each Lesson has a title, a description providing information about it. It focuses on a set of Topics. The Lesson also holds information about its authors and the URL where the materials related to the Lesson can be found on the Web.

Attribute Name	Attribute Type	Description
Title	Text	Title of the Lesson
Description	Text	Description of the Lesson
Topics	$\mathbb{P}$ Topic	Topics of the Lesson
Authors	$\mathbb{P}$ Actor	Authors of the Lesson
URL	URL	URL where the Lesson’s materials can be found

Table 7.2: Lesson Artifact Attributes

7.5.2 Activities Definition

In the scope of the scenario, only one additional Activity has to be defined. This Activity has to handle the Courses as well as the Lessons related to these Courses. The Course Management Activity enables allowed Actors to propose Courses and Lessons, evaluate them and then approve them before making them available to the public. Lessons and Courses can be replaced or removed. They can also be activated or deactivated if it is needed to hide a Course or a Lesson to the public. This Activity also provides different operations for searching for Courses and Lessons. Table 7.3 lists the different operations provided by this Activity.

Name	Description
propose(Course):void	Proposes a Course
propose(Lesson, Course):void	Proposes a Lesson for a Course
getProposedCourses():ℙCourse	Returns proposed Courses
approveCourse(Course):void	Approves a proposed Course and registers it
rejectCourse(Course):void	Rejects a proposed Course
getProposedLessons(Course):ℙLesson	Returns proposed Lessons for a Course
approveLesson(Lesson):void	Approves a proposed Lesson and registers it
rejectLesson(Lesson):void	Rejects a proposed Lesson
courses():ℙ Course	returns registered and active Courses
lessons(Course):ℙ Lesson	returns registered and active Lessons for a Course
evaluate(Course):Boolean	Returns the evaluation for a Course
evaluate(Lesson):Boolean	Returns the evaluation for a Lesson
activate(Course):void	Activates a Course
deactivate(Course):void	Deactivates a Course
activate(Lesson):void	Activates a Lesson
deactivate(Lesson):void	Deactivates a Lesson
remove(Course):void	Removes a Course
replaceCourse(Course, Course):void	Replaces a Course with a modified one
replaceLesson(Course, Lesson, Lesson):void	Replaces a Course Lesson with a newer one
remove(Lesson, Course):void	Removes a Lesson for a Course
findCourse(Directory):ℙArtifact	Searches for Course Artifacts
findLesson(Directory):ℙArtifact	Searches for Lessons Artifacts
findLesson(Course, Directory):ℙArtifact	Searches for Lessons Artifacts in the scope of a Course
exists(Course):Boolean	Checks if a Course exists
exists(Lesson, Course):Boolean	Checks if a Course exists

Table 7.3: Operations of the Course Management Activity.

7.5.3 Registration

Once all these new Artifacts types and Activities are created, John can register them using the following calls related to the processes $\pi_{artifactDeclaration}$ and $\pi_{activityDeclaration}$. Each $I\mathcal{R}$ represents an integrity rule bound to the ArtifactType or to the Activity being created.

ArtifactActivity.registerArtifactType(Directory$_{CourseType}$, Directory$_{Valuation}$, $\mathbb{P}\, I\mathcal{R}$, OC)
ArtifactActivity.registerArtifactType(Directory$_{LessonType}$, Directory$_{Valuation}$, $\mathbb{P}\, I\mathcal{R}$, OC)
ActivityActivity.declareActivity("'CourseActivity"', $\mathbb{P}(\lambda, \mathbb{P}\, I\mathcal{R})) -> CA$
ActivityActivity.bindActivity(CA, OC)

7.5.4 Specific Processes

The following processes are made from the operations listed in Table 7.3. These processes allow the creation, evaluation, modification, and usage of Courses and Lessons.

Course Proposition. The creation of a Course has to undergo a given number of steps. The first one of these steps is the proposition of a Course. When proposing a Course a evaluation request is raised.

$$\begin{aligned}\pi_{courseProposition} \quad = \quad & CourseActivity.propose(course);\\ & EventActivity.raise(CourseEvaluationRequest));\end{aligned}$$

Course Evaluation. The Course evaluation process is triggered by the raise of a $CourseEvaluationRequest$ Event. Depending on the result of the evaluation of the Course, the Course is activated and a $PositiveCourseEvaluation$ event is raised or the Course is deactivated and a $CourseModificationRequest$ Event is raised.

$$\begin{aligned}\pi_{courseEvaluation} \quad = \quad & EventActivity.observe(CourseEvaluationRequest);\\ & CourseActivity.evaluate(course)?\\ & (CourseActivity.activate(course);\\ & EventActivity.raise(PositiveCourseEvaluation)) :\\ & (CourseActivity.deactivate(course);\\ & EventActivity.raise(CourseModificationRequest));\end{aligned}$$

Course Approval. The Course Approval process is triggered by the observation of a $PositiveCourseEvaluation$. When this Event is received, the Course correctness is verified, however it is still not provided by the Project. To make a Course available through the Activity, it must be approved after having been evaluated.

$$\begin{aligned}\pi_{courseApproval} \quad = \quad & EventActivity.observe(PositiveCourseEvaluation);\\ & (CourseActivity.approveCourse(course) \,||\, CourseActivity.rejectCourse(course));\end{aligned}$$

Course Modification. If a Course is not considered complete enough for being distributed, or if errors have been detected, a $CourseModificationRequest$ Event is raised. When observed, a Course replacement must be then provided, then the replacement must be evaluated.

$$\begin{aligned}\pi_{courseModification} \quad = \quad & EventActivity.observe(CourseModificationRequest);\\ & CourseActivity.replaceCourse(course, course);\\ & EventActivity.raise(CourseEvaluationRequest);\end{aligned}$$

Lesson Proposition. As for Courses, Lessons can be proposed through the Course Management Activity. The proposition triggers the evaluation of the Lesson being proposed.

$$\begin{aligned}\pi_{lessonProposition} \quad = \quad & CourseActivity.propose(Lesson, Course);\\ & EventActivity.raise(LessonEvaluationRequest);\end{aligned}$$

Lesson Approval. Once a Lesson is considered as correct, it can be approved or rejected depending on how it integrates with other Courses.

$$\begin{aligned}\pi_{lessonApproval} \quad = \quad & EventActivity.observe(PositiveLessonEvaluation);\\ & (CourseActivity.approveLesson(lesson, course) \,||\\ & CourseActivity.rejectLesson(lesson, course));\end{aligned}$$

Lesson evaluation. Again, as for Courses, the evaluation of a Lesson is triggered by a proposition. The evaluation can be positive or negative. If positive, the Lesson is activated, and a $PositiveLessonEvaluation$ Event is raised. If not, the Lesson is deactivated, and modifications are requested through the raise of a $LessonModificationRequest$

$$\begin{aligned}\pi_{lessonEvaluation} \quad = \quad & EventActivity.observe(LessonEvaluationRequest);\\ & CourseActivity.evaluate(lesson)?\\ & (CourseActivity.activate(lesson);\\ & EventActivity.raise(PositiveLessonEvaluation)) :\\ & (CourseActivity.deactivate(lesson);\\ & EventActivity.raise(lessonModificationRequest));\end{aligned}$$

Lesson Modification. This process waits for $LessonModificationRequest$ Events then replaces a Lesson with a modified version of it, which is supposed to be correct. Once the replacement is done, a evaluation of the new Lesson is requested by raising a $LessonEvaluationRequest$.

$$\begin{aligned}\pi_{lessonModification} \quad = \quad & EventActivity.observe(LessonModificationRequest);\\ & CourseActivity.replace(course, lesson, lesson);\\ & EventActivity.raise(LessonEvaluationRequest);\end{aligned}$$

Course Check. The check process triggers the evaluation of a Lesson or the evaluation of a Course. After the check the **Course Approval** or **Lesson Approval** Processes are run, which allows to take a decision about the approval or rejection of the Course or Lesson.

$$\begin{aligned}\pi_{checkCourse} \quad = \quad & (EventActivity.raise(courseEvaluationRequest) \;||\\ & EventActivity.raise(LessonEvaluationRequest));\end{aligned}$$

Course Visibility. This process takes care of making Courses and Lessons visible to the Project's Community.

$$\begin{aligned}\pi_{visibilityCourse} \quad = \quad & (CourseActivity.activate(course) \;||\\ & CourseActivity.deactivate(course) \;||\\ & CourseActivity.activate(lesson) \;||\\ & CourseActivity.deactivate(lesson));\end{aligned}$$

Courses and Lessons search. There are multiple means to find a Course or a Lesson. This Process lists them.

$$\begin{aligned}\pi_{searchCourse} \quad = \quad & (courses() \;||\\ & CourseActivity.lessons(Course) \;||\\ & CourseActivity.findCourse(Directory) \;||\\ & CourseActivity.findLesson(Directory) \;||\\ & CourseActivity.findLesson(Course, Directory));\end{aligned}$$

7.5.5 Specific Roles

Course Creator. The Course Creator Role is responsible for proposing Courses and Lessons. It is also the Creator who has to modify the proposed Courses if any problem occurs. Thus the processes part of this Role are the following: $\pi_{courseProposition}$, $\pi_{courseModification}$, $\pi_{lessonProposition}$ and $\pi_{Lessonmodification}$.

Course Manager. While the Creator who proposes and corrects Courses and Lessons, the Course Manager decides whether a Course or a Lesson is worth being proposed by the project. He also decides if a Course or a Lesson needs to be checked and be made invisible for any reason. The processes he undertakes are $\pi_{courseApproval}$, $\pi_{lessonApproval}$, $\pi_{checkCourse}$ and $\pi_{visibilityCourse}$.

Course Evaluator. The Course Evaluator Role is responsible for evaluating Courses and Lessons every time the Course Manager requests it, or when a Course Creator proposes a new Course or Lesson. This involves two different Processes: $\pi_{courseEvaluation}$ and $\pi_{lessonEvaluation}$.

Course User. Finally, the last Role defined by the Project team is Course User. As its name highlights it, this Role has access to the Courses and Lessons proposed by the Project. The related Process is: $\pi_{searchCourse}$.

7.5.6 Specific Tasks attribution

The specific Roles defined in previous section have been dispatched as follows between the different participants to the Project.

RoleActivity.assignRole("'Course Creator"',Alex, OC);
RoleActivity.assignRole("'Course Creator"',Shelly, OC);
RoleActivity.assignRole("'Course Manager"',John, OC);
RoleActivity.assignRole("'Course Evaluator"',Alice, OC);

Course Creator. Alex and Shelly are chosen as Course and Lesson Creators for the Project. Their Task will be to propose courses and modify them if needed.

Course Manager. John receives the Role of Program Manager, thus he becomes responsible for the final choice of proposed Courses and Lessons. He can also switch the visibility of a Course or of a Lesson if needed.

Course Evaluator. Alice is chosen as the evaluator of the project. She will have to evaluate proposed Courses and Lessons and ask for modifications if needed.

Course User. Any registered Actor registered with the Project as a Contributor gets this role and thus can search for Courses and Lessons. This allows anyone to retrieve the content proposed by the project

7.6 Project Evolution

7.6.1 Organization Evolution

Over time some changes can occur in any Project. The first type of possible changes changes any Project has to face is also the most probable one. Organizational changes with people movement occur often in F/OSS Projects. The difficulty they imply is that they require tasks reassignment. Some of such changes can be small and punctual as the illness of Alex during one month. During this time, a replacement person has to be found within the project. Multiple criteria can direct the choice. It can be done on a required competencies basis or on a trust basis. We can for instance imagine that Alex wants John to be his replacement, and thus wants him to become responsible for the roles he is usually handling. Some other changes can be more important such as Shelly pregnancy. In such a situation she may decide to

leave the Project for a long time as she prefers to invest more time in her family. This implies that a replacement having matching competencies must be found.

Such organizational changes can be handled through the PRM. In the first case, the Roles of Alex are retrieved and are assigned to John by Shelly, the responsible person for tasks assignment, for the Period during which Alex is missing, while keeping in mind current Roles of John:

RoleActivity.getRoles(John, Project) $->$ johnRolesOld
RoleActivity.getRoles(Alex, Project) $->$ alexRoles
RoleActivity.assignRoles(John, Project, alexRoles)

After this assignment, John would be able to see his new Tasks by querying the PRM as follows:

RoleActivity.getRoles(John, Project) $->$ johnRolesNew

Once Alex returns, the Roles having been added can be easily removed from John by Shelly to recover the initial situation.

rolesToRemove = alexRoles - johnRolesOld
RoleActivity.unassignRoles(John, Project, rolesToRemove)

In the second use case, as Shelly indicates that she leaves the Project, the first step is to find a replacement person having similar competencies. For instance, we can imagine that Mike is a member of the Project having the same competencies as Shelly.

As the Task Manager, she can then associate her Role, **Task Manager** to Mike.

RoleActivity.assignRoles(Mike, Project, "'Task Manager"');

Alex can then deactivate Shelly. Indeed she is not anymore an active member of the Project and thus she should not be able anymore to act as such. However, she has not been banned from the project. Thus as she can rejoin the team anytime, information about her duties can be kept. She also holds knowledge which can be useful to the community if needed.

ProjectActivity.deactivate(Shelly, Project)

7.6.2 Activities Evolution

Another type of evolution Projects are subject to is related to the Activities of Projects. As Project grow, they may wish to add new Activities to complete the services they offer. In the case of our scenario, imagine that the team decides to provide exercises related to the Courses and Lessons already provided. Such an addition, can be easily expressed through the PRM. It implies the creation of a corresponding Artifact type, an Activity for handling the new Artifact as well as for integrating the Artifact with existing Activities, the definition of Processes involving the new Activity, Roles and the assignment of these Roles as Tasks.

Artifacts and Activities

Exercise Artifact definition. The Exercise Artifact indicates where exercises questions with corresponding answers can be found on the web. Table 7.4 gives more information about the Attributes of the Exercise Artifact. All these attributes are mandatory, and no special integrity rule has to be specified. The newly created Artifact Type must be then registered with the PRM for the project by John, the Activity Manager, as follows:

ArtifactActivity.registerArtifactType(Directory$_{ExerciseType}$, Directory$_{Valuation}$, $\mathbb{P}\,\mathcal{IR}$, OC)

Attribute Name	Attribute Type	Description
Title	Text	Title of the Exercise
Authors	$\mathbb{P}$ Actor	Authors of the Exercise
Description	Text	Description of the Exercise
Topic	$\mathbb{P}$ Topic	Exercise Topics
$\text{URL}_{questions}$	URL	URL the questions can be found
$\text{URL}_{answers}$	URL	URL the answers can be found

Table 7.4: Exercise Artifact Attributes

Exercise handling Activity definition. Once the Exercise Artifact is registered for the Project, an Activity for managing Exercises needs to be defined and registered for the Project. As for Courses, the team wants Exercises to be proposed then evaluated before being approved or rejected. Alex and his friends also wants a possibility to disable Exercises if needed. This leads them to declare the operations listed in Table 7.5 for the Exercise Management Activity. Among integrity Rules to be enforced, note for instance that an Exercise can be approved or rejected only if it has first been proposed and positively approved. The registration of the Activity is then done by John as follows:

ActivityActivity.declareActivity("'ExerciseActivity"', $\mathbb{P}(\lambda, \mathbb{P}\, I\mathcal{R})) -> \text{CM}$
ActivityActivity.bindActivity(CM, OC)

Name	Description
propose(Exercise, Course, Lesson):void	Proposes an Exercise for a Lesson of a Course
getProposedExercises(Course, Lesson):$\mathbb{P}$Course	Returns proposed Exercises for a Lesson of a Course
approveExercise(Exercise):void	Approves a proposed Exercise and registers it
rejectExercise(Exercise):void	Rejects a proposed Exercise
exercises():$\mathbb{P}$ Exercise	returns registered and active Exercises
exercises(Course):$\mathbb{P}$ Exercise	returns registered and active Exercises for a Course
exercises(Course, Lesson):$\mathbb{P}$ Exercise	returns registered and active Exercise for a Course's Lesson
evaluate(Exercise):Boolean	Returns the evaluation for an Exercise
activate(Exercise):void	Activates a Exercise
deactivate(Exercise):void	Deactivates a Exercise
remove(Exercise):void	Removes a Exercise
replaceExercise(Course, Lesson, Exercise, Exercise):void	Replaces a Course Lesson's Exercise with a newer one
findExercise(Directory):$\mathbb{P}$Artifact	Searches for Exercise Artifacts
findExercise(Course, Directory):$\mathbb{P}$Artifact	Searches for Exercise Artifacts in the scope of a Course
findExercise(Course, Lesson, Directory):$\mathbb{P}$Artifact	Searches for Exercise Artifacts in the scope of a Course and a Lesson
exists(Exercise):Boolean	Checks if an Exercise exists
exists(Course, Lesson, Exercise):Boolean	Checks if an Exercise exists for a Course's Lesson

Table 7.5: Operations of the Exercise Management Activity.

Processes.

The following Processes are specific to the management of the Exercise Activity, as such they must be registered on the PRM.

Exercise Proposition. The creation of an Exercise has to undergo a given number of steps. First the Exercise needs to be proposed. When proposing a Course a evaluation request is raised as the following process illustrates it:

$$\begin{aligned}\pi_{exerciseProposition} \quad = \quad & ExerciseActivity.propose(exercise);\\ & EventActivity.raise(ExerciseEvaluationRequest));\end{aligned}$$

Exercise Evaluation. The evaluation of an Exercise is triggered by the raise of a $ExerciseEvaluationRequest$ Event. Depending on the result of the evaluation, the Exercise is activated and a $PositiveExerciseEvaluation$ event is raised or the Exercise is deactivated before a $ExerciseModificationRequest$ Event is raised.

$$\begin{aligned}\pi_{exerciseEvaluation} \quad = \quad & EventActivity.observe(ExerciseEvaluationRequest);\\ & ExerciseActivity.evaluate(exercise)?\\ & (ExerciseActivity.activate(exercise);\\ & EventActivity.raise(PositiveExerciseEvaluation)) :\\ & (ExerciseActivity.deactivate(exercise);\\ & EventActivity.raise(ExerciseModificationRequest));\end{aligned}$$

Exercise Approval. The observation of a $PositiveExerciseEvaluation$ triggers the Process of Exercise Approval. When this Event is received, while the Exercise correctness is considered has having been verified, to make an Exercise available to the Project's community through the Activity, it must be approved.

$$\begin{aligned}\pi_{exerciseApproval} \quad = \quad & EventActivity.observe(PositiveExerciseEvaluation);\\ & (ExerciseActivity.approveExercise(exercise) \;||\\ & ExerciseActivity.rejectExercise(exercise));\end{aligned}$$

Exercise Modification. If an Exercise is not considered complete enough for being distributed, or if errors have been detected, an $ExerciseModificationRequest$ Event is raised. When such an Event is observed, a replacement fixing detected issues must be provided, then the replacing Exercise must be evaluated.

$$\begin{aligned}\pi_{exerciseModification} \quad = \quad & EventActivity.observe(ExerciseModificationRequest);\\ & ExerciseActivity.replaceExercise(course, lesson, exercise, exercise);\\ & EventActivity.raise(ExerciseEvaluationRequest);\end{aligned}$$

Exercise Check. The check process triggers the evaluation of an Exercise. After the check the **Exercise Approval** Process is run, which allows to take a decision about the approval or rejection of the Exercise.

$$\pi_{checkExercise} \quad = \quad EventActivity.raise(exerciseEvaluationRequest);$$

Exercise Visibility. This process handles Exercises visibility by the Project's Community.

$$\begin{aligned}\pi_{visibilityExercise} \quad = \quad & (ExerciseActivity.activate(exercise) \;||\\ & ExerciseActivity.deactivate(exercise);\end{aligned}$$

Exercise Search. There are multiple means to find a Exercise. This Process lists them.

$$\begin{aligned}\pi_{searchExercise} = \; & (ExerciseActivity.exercises() \;||\\ & ExerciseActivity.exercises(course) \;||\\ & ExerciseActivity.exercises(course, lesson) \;||\\ & ExerciseActivity.findExercise(Directory) \;||\\ & ExerciseActivity.findExercise(Course, Directory) \;||\\ & ExerciseActivity.findExercise(Course, Lesson, Directory));\end{aligned}$$

Roles.

Exercise Creator. The Exercise Creator Role is responsible for proposing Exercises and modifying them if any problem occurs. Thus the processes part of this Role are the following: $\pi_{exerciseProposition}$, $\pi_{exerciseModification}$.

Exercise Manager. The Exercise Manager decides whether a Exercise is worth being proposed by the project for a Course Lesson. He also decides if an Exercise needs to be checked and be made invisible for any reason. The processes he undertakes are $\pi_{exerciseApproval}$, $\pi_{checkExercise}$ and $\pi_{visibilityExercise}$.

Exercise Evaluator. The Exercise Evaluator Role is responsible for evaluating Exercises every time the Exercise Manager requests it, or when an Exercise Creator proposes a new Exercise. This involves the following Process: $\pi_{exerciseEvaluation}$.

Exercise User. Finally, the last Role bound to Exercise Management, is Exercise User. As its name indicates it, this Role has access to the Exercises proposed by the Project. The related Process is: $\pi_{searchExercise}$.

Tasks.

As for Course Management, the Roles defined for Exercise Management have been dispatched as follows between the different participants to the Project.

```
RoleActivity.assignRole("‘Exercise Creator"’,Alice, OC);
RoleActivity.assignRole("‘Exercise Creator"’,John, OC);
RoleActivity.assignRole("‘Exercise Manager"’,Alex, OC);
RoleActivity.assignRole("‘Exercise Evaluator"’,Alex, OC);
RoleActivity.assignRole("‘Exercise Evaluator"’,Alice, OC);
RoleActivity.assignRole("‘Exercise Evaluator"’,John, OC);
RoleActivity.assignRole("‘Exercise Evaluator"’,Shelly, OC);
```

Exercise Creator. As Alex and Shelly are already the Course creators, Alice and John are chosen as Exercise Creators for the Project. Their Task will be to propose Exercises and modify them if needed.

Exercise Manager. Alex receives the Role of Exercise Manager, thus he becomes responsible for the final choice of proposed Exercises for Courses' Lessons.

Exercise Evaluator. As this Role is highly time consuming the whole team receives the role of Exercise Evaluator. They will have to evaluate proposed Exercises and ask for modifications if needed.

Course User. Any Actor registered with the Project as a Contributor gets this role and thus can search for Exercises.

7.7 Scenario Wrap up

This simple scenario has showed how the organization of a small Project can be handled through the PRM. Involved Processes, Roles as well as artifact types, Activities and Metrics are information which can be shared among projects in an open source manner. They can be made accessible to other projects to help their management as well as to the people of the project to better explain how an activity is run, what roles it consists of, etc.

The PRM forces projects to think about what they do, about why they do it that way. Further it provides a means to describe and thus explain how a project is managed. When creating new Activities and Artifact Types, one can explain why he has simplified or improved an existing Activity or Artifact. Thus new users can decide whether or not they want to use the modified version.

Understanding the Process is the first step toward Process improvement. The PRM provides a means to describe the process as a whole and includes the ability to express how new activities have to be included. Gathered information about the process is meant to be available to everybody. Sharing Process Management information is a step toward F/OSS project management. Such an approach goes beyond the usual F/OSS concept by not only sharing the content but also by sharing working methods. Any participant to the F/OSS process can benefit from such an evolution:

- **Small projects.** They get guidelines for growing, and get a means for finding and including competencies not available internally by reusing Activities, Artifact Types, Processes, Metrics modeled by larger Projects. This helps them focus on their core work.
- **Large projects.** They get increased control over the information. The global view provided by the PRM enables transversal measurement as well as more efficient Project (re)organization.
- **All projects.** They can get a way to better explain what they are doing, the competencies they are looking for. Their visibility gets thus increased, which can help finding new contributors and competencies which may usually difficult to get.
- **Community.** Contributors can benefit from the PRM awareness of their interests, competencies and knowledge. Further, their integration with a project may be simpler as they can get information about involved processes, and thus can see how the particular task they have to undertake integrates with the global picture of the project.
- **Managers.** They get a better understanding of the project, they can easier handle organizational changes occurring within the Project, search for competencies, handle their management tasks. Further, through the use of Metrics they can build analysis on top of described projects and processes in order to improve them.
- **All.** The PRM models a means to easily find information about registered Projects.

Part III

Application

Chapter 8

PRM Implementation and usage

8.1 PRM Implementation

To enable the implementation of scenarios such as the one presented in Chapter 7, the PRM has been implemented as a Java API. The goal of this API is to allow people to build complete applications using PRM properties, such as a web portal, and content management systems designed for project and process management. The API offers the following features:

- **PRM mechanisms.** An implementation of PRM mechanisms such as Artifact matching, Artifact substitutability, Artifact lookup, Event management and Metrics management is provided. Further, data structures these mechanisms are depending on are also provided. This includes Artifact Directories and Artifact Sets. Artifact Expressions are built using Artifact Directories in conjunction with Artifact Filters, which enable boolean expressions as well as the use of wildcards to make the lookup process more flexible.

- **Activities.** All core PRM Activities defined in Section 5.1.4 have been provided as abstract classes ready to be implemented by PRM users. New Activities can be declared and added through the Activity Management Activity.

- **Artifacts.** All Artifacts related to the core PRM Activities have been implemented as well. They are provided along with the substitutability relation they have to follow, and integrity rules they have to enforce. In order to support the creation of new Artifact Types, the API offers different abstract types of Artifacts for handling the substitutability of equivalent artifacts, textual artifacts (both case sensitive and case insensitive), compatible artifacts, numbers, partially and totally ordered Artifacts.

8.2 The EDOS Project testbed

The PRM model has been extended in the scope of the EDOS (Environment for the development and Distribution of Open Source software) Project to formalize activities related to the production of a GNU/Linux distribution. EDOS is a research project funded by the European Commission as a STREP project under the IST activities of the 6th Framework Programme. The project involves universities - Paris 7, Tel Aviv, Zurich and Geneva Universities -, research institutes - INRIA - and private companies - Mandriva, Caixa Magica, Nexedi, Nuxeo, Edge-IT and CSP Torino.

As the number of users and of available packages grows constantly, the complexity of the production of a F/OSS distribution grows as well. This process consists of the packaging and assembling of a large number of programs and applications into a single tested product, together with a set of tools to install

and administer the system. Distribution editors such as Mandriva, Debian, Red Hat, Ubuntu or Caixa Magica have to handle the following two processes:

- **Producing a new version of a distribution:** starting from a complete version of the distribution, the editor adds, deletes, reconfigure and updates packages with respect to the external sources. This implies the integration of external code, the run of multiple tests, debugging, etc.
- **Customizing a distribution for a specific user:** starting from a complete version of the distribution, the editor adds, deletes, updates and configures packages to make the distribution fit specific needs. Multiple constraints can shape the resulting distribution.

The usual approach to execute these processes is to use collections of ad hoc scripts and a lot of manual intervention. However, as the number of packages grows very quickly, the underlying complexity grows as well and this approach becomes increasingly inadequate. The EDOS Project focuses on the problems existing in the production of F/OSS operating systems on a large scale and its main objective is to develop technology and tools to support and improve these two processes.

Four activities are considered as central by the EDOS Project in relation with these processes: distribution management, quality assurance management, dependencies management and metrics management. Each of them focuses on a specific aspect of the F/OSS process and needs to be improved. As cross-issues exist, information needs to be shared in order to efficiently address each of these tasks in a global manner. Content distribution cannot be addressed without also considering dependencies as well as content testing and bug reporting. Similarly, process measurement cannot be achieved if access to required information is impossible. All of these topics are inter-related, so each influences the way the others should be tackled.

In this context, the PRM (formerly known as the Project Management Interface (PMI)) has been used as a means for reasoning in a transversal manner about the processes involved in the production of a GNU/Linux distribution [42, 115, 116]. The idea was to define a transversal task for linking these activities all together. The PRM has defined guidelines to which a project can refer in order to implement its activities, to integrate them, to expose them as well as to indicate any information it relies on. To fulfill the specific needs of the EDOS project, an extension of the PRM had to be defined.

The elements – Artifacts, Activities as well as integrity rules and substitutability rules – involved in the F/OSS Process of developing a linux distribution have been thus integrated as the PRM extension for EDOS [115] (EDOS-PRM.) The later is available in Appendix B. This specification is meant to be used as the foundation for a new kind of integrated development environment specifically tailored to F/OSS development, diffusion and testing by distributed people.

Using the PRM had the advantage to provide a complete specifications, while easing the addition of new activities. The PRM can be implemented in different ways such as a web infrastructure or as a rich client infrastructure depending on implementors choice. In fine, the resulting tools aims at providing support for:

- changing rapidly RPMs during the development stage while ensuring global integrity
- checking the dependencies
- getting live metrics on the evolution of the processes
- testing the distribution

The EDOS-PRM and the PRM have been used in different contexts which are further explored in the next chapters of this Thesis.

Chapter 9

From Process Measurement to Decision Making

Process measurement is central to process improvement. The PRM provides features to handle process-wide measurement, however, the adopted approach is only a bootstrap allowing further extensions. In its form presented in chapter 5, it does not allow complex actions such as the definition of schedules, objectives, and thresholds which are needed for advanced performance measurement and strategy definition. Such features were desirable in the context of the EDOS project. Thus, in order to improve the measurement capabilities offered by the PRM, an extension, the PRM Process Measurement Improvement (PRM-measurement), handling such features has been designed.

This chapter presents this extension. It enables the usage of advanced metrics and key performance indicators (KPI). The definition KPIs as a combination of metrics with thresholds presenting and explaining possible values gives more expressiveness to the metrics Artifacts. This allows thus reasoning about the results obtained after metric calculation. Expressing project objectives as combinations of KPIs can be used to analyze the process and take decisions to improve it. Among other features, this extension is meant to provide the following ones:

- **Metrics refinement.** The PRM-measurement extension separates single measurements done through the PRM, from the metrics they are involved in. This added granularity allows to reuse single measurements in different contexts, while providing a description of the returned value.
- **Key performance indicators handling.** Metrics only provide numeric values not specifying what value should be reached. The PRM-measurement adds the concept of key performance indicators (KPI) and allows to associate targets and thresholds to single metrics. This enables the qualification of metrics depending on the context of their usage and enables performance analysis.
- **Objectives handling.** The PRM-measurement provides a means to define the objectives of a project in terms of KPIs, i.e. in terms of performance to reach.
- **Schedule based execution of metrics.** A scheduler for executing metrics on a time or Event basis is provided by the PRM-measurement extension.

An implementation providing an interface for handling metrics is under development [1]. The PRM-measurement extension will be completed by a graphical user interface. Its aim is to be an interactive dashboard allowing to handle metrics as Table 9.1 shows it.

[1] This work is part of the master thesis in management and technologies of Information Systems of Othmar Heini, a student from the University of Geneva

Feature	Description
Metrics declaration	The GUI has to enable the creation of new Metrics, their composition, etc.
Metrics usage	The Metrics can be used by different Projects within different KPIs. These KPIs can be used themselves in the scope of different objectives.
Metrics persistence	The metrics management tool has to make created metrics persistent and available to other projects.
Metrics retrieval	Any Project should be able to access a declared Metric trough the GUI tool.
Metrics execution	The Metrics can be executed on demand by the users or on schedule.
Metrics monitoring	The graphical tool has to display the result of executed metrics. It has also to display the KPIs for each project, along with the objectives defined for each project.

Table 9.1: PRM-measurement extension GUI features

9.1 Process Measurement PRM Extension in details

The extension of the measurement part of the PRM model involved the modification of some Artifacts and the addition of new ones. Figure 9.1 lists the main Artifacts added by this extension, Figure 9.2 provides an UML diagram of them. The Process measurement PRM extension also defines a set of new Activities as presented in Figure 9.3. We detail these activities as well as their operations and artifacts in the following paragraphs.

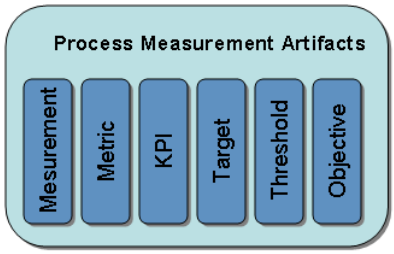

Figure 9.1: Artifacts of the Process Measurement Extension

MeasurementManagement Activity allows the creation and management of Measurements. A **Measurement** is a numeric value which represents a particular fact at a specific moment in time such as the size, length, or amount of something, as established by measuring. A Measurement could for instance be the number of bugs reported for package p during the week w. Measurements are typically the building blocks of Metrics. It is through their combination in the Metric's formula that complex, transversal Metrics can be expressed. Table 9.2 details the attributes of this Artifact.

Attribute Name	Attribute Type	Description
Name	Text	Name of the Measurement
Description	Text	Description of the Measurement
Operation	Operation	Operation that delivers the value of the Measurement

Table 9.2: Measurement Artifact Attributes

A Measurement is composed of a name and a description. Its operation takes a certain number of parameters and returns a numeric value as result. The parameters of an operation can be static data,

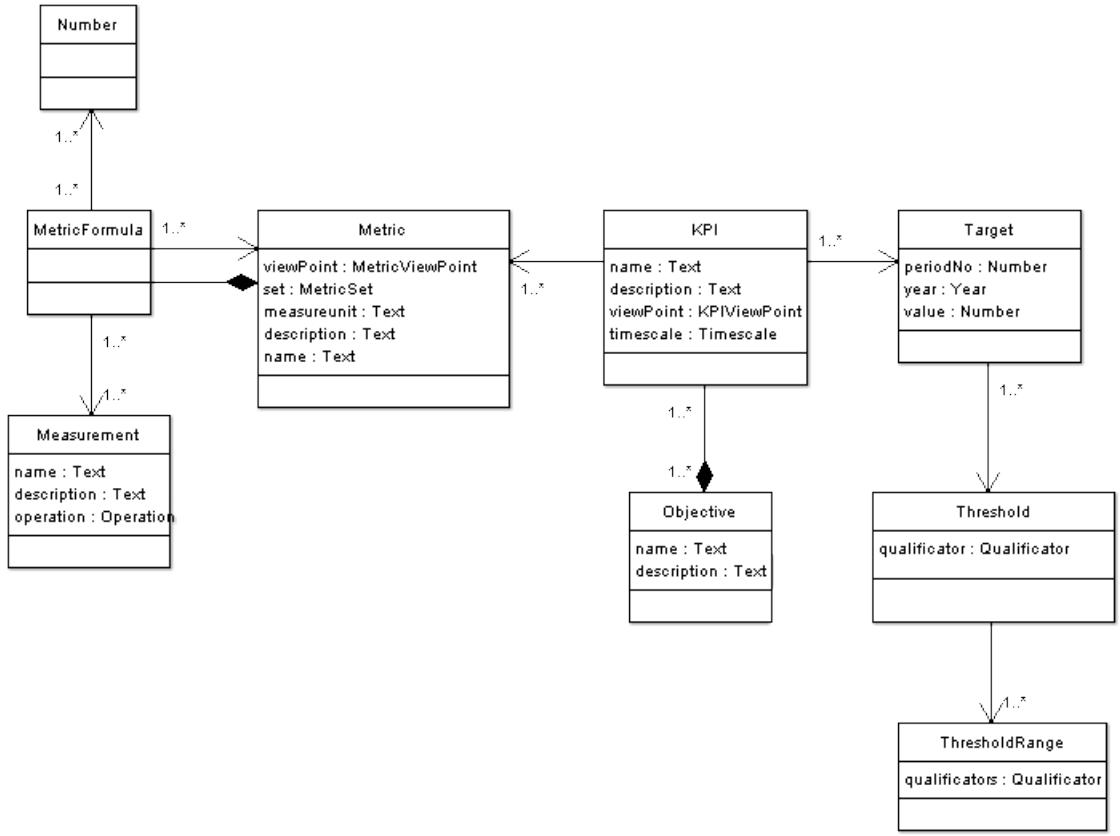

Figure 9.2: PRM Extension for Process Measurement

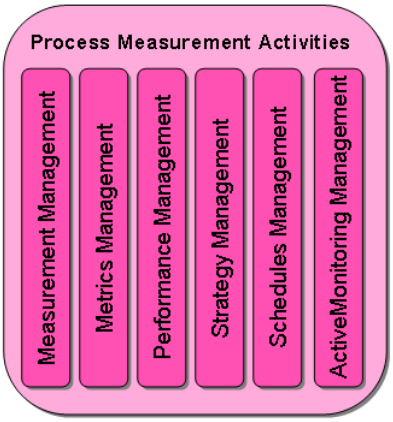

Figure 9.3: Activities of the Process Measurement Extension

the result of another operation or parameters that are defined at the moment of execution. Time-related parameters, such as "yesterday", "during January" or "in the last 10 days", are typically part of this last category. Having a set of undefined parameters allows a Measurement to be used and reused for a number of purposes. Table 9.3 provides the operations bound to the Measurement Activity.

Operation Name	Parameters and Return Type	Description
declareMeasurement	(Text$_{name}$, Text$_{description}$, ArtifactType, λ):Measurement	Creates a new Measurement
registerMeasurement	(Measurement, Project):void	Associates a Measurement to a Project
unregisterMeasurement	(Measurement, Project):void	Dissociates a Measurement from a Project
evaluateMeasurement	(Measurement, Directory$_{parameters}$):Number$_{result}$	Evaluates a Measurement, the parameters are those required by the operation to be run

Table 9.3: Measurement management operations.

Metrics Management Activity. The Metrics Management Activity allows the creation and management of Metrics. A Metric is "a quantitative measure of the degree to which a system, component, or process possesses a given attribute". A Metric can be based on a single Measurement, more often however, a Metric summarizes a set of data. A typical metric would for instance be the percentage of resolved bugs vs announced bugs for a package p. Metrics are built from Measurements, other Metrics, and/or constant values. The Metric's formula permits the combination of these building blocks in a flexible manner, allowing thus Metrics to be aggregated over several levels of granularity. A Metric possesses a name and a description, its viewpoint indicates a particular domain it belongs to. Furthermore, a metric is part of a metricset, which is a set of logically related Metrics. The formula is an arithmetic function composed of operands and operations as shown in Figure 9.4. Finally, the measureunit indicates the unit of measurement of the Metric's output, for example "minutes", "dollars" or "percent".

Attribute Name	Attribute Type	Description
Name	Text	Name of the Metric
Description	Text	Description of the Metric
Viewpoint	MetricViewPoint	Domain the Metric belongs to
Set	MetricSet	Logical set of Metrics the Metric belongs to
Formula	MetricFormula	Formula which expresses the Metric
Measureunit	Text	Unit of measurement for the metric's output

Table 9.4: Metric Artifact Attributes

The extended Metric formula is defined as standard arithmetical expressions in Figure 9.4. Table 9.5 lists the operations bound to the Metrics management activity.

Performance Management Activity. This particular activity allows project performance monitoring through the creation and management of key performance indicators (KPIs), Targets and Thresholds. The combination of these three artifacts allows projects to use metrics qualifying the result they return in order to provide managers information about what the expected values are, what they mean, in other words how this value has to be interpreted. Table 9.6 lists the operations bound to the Performance management activity.

$$
\begin{array}{lll}
formula & ::= & expression \\
expression & ::= & operand \mid expression\ operation\ expression \mid \text{"("}\ expression\ operation\ expression\ \text{")"} \\
operand & ::= & Constantvalue \mid Measurement \mid Metric \\
operation & ::= & \text{"}+\text{"} \mid \text{"}-\text{"} \mid \text{"}*\text{"} \mid \text{"}/\text{"} \mid \text{"}\%\text{"}
\end{array}
$$

Figure 9.4: Extended Metric Formula definition

Operation Name	Parameters and Return Type	Description
declareMetric	(Text$_{name}$, Text$_{description}$, MetricViewpoint, MetricSet, MeasurementUnit, *formula*):Metric	Creates a new Metric
registerMetric	(Metric, Project):void	Associates a Metric to a Project
unregisterMetric	(Metric, Project):void	Dissociates a Metric from a Project
evaluateMetric	(Metric, Directory$_{parameters}$):Number$_{result}$	Evaluates a Metric, the parameters of this operation are those required by the different Measurements being evaluated during the evaluation of the Metric

Table 9.5: Metrics management operations.

Operation Name	Parameters and Return Type	Description
declareThreshold	(ThresholdRange, Qualificator):Threshold	Creates a new Threshold
declareTarget	(Number$_{periodNo}$, Year, $\mathbb{P}$Threshold)):Target	Creates a new Target
declareKPI	(Text$_{name}$, Text$_{description}$, KPIViewpoint, TimeScale, $\mathbb{P}$Target):KPI	Creates a new KPI
registerKPI	(KPI, Project):void	Associates a KPI to a Project
unregisterKPI	(KPI, Project):void	Dissociates a KPI from a Project
getKPI	(Project):$\mathbb{P}$KPI	Gets KPIs for a Project
getTargets	():$\mathbb{P}$Target	Gets Targets
getThresholds	():$\mathbb{P}$Threshold	Gets Thresholds
evaluateKPI	(KPI, Directory$_{parameters}$):Directory$_{result}$	Evaluates a KPI, the resulting Directory contains the value of the Metric, the target value and the qualificator of the Metric's Value

Table 9.6: Performance management operations.

A **KPI** is "a metric that embeds performance targets so projects can chart progress toward goals". It associates a Metric with a number of Targets that provide a context for the Metric. A Target represents a value a project would like to reach over a specific period of time. It is composed of a name, a description and a viewpoint the KPI belongs to. The attribute metric designates the Metric which is used by the KPI. The timescale specifies the granularity of the time periods for which the Targets are set, for example daily, monthly, quarterly or yearly. Finally, the KPI holds a number of targets. These Attributes are listed in Table 9.7.

A **Target** represent a value a project would like to reach over a specific period of time. For the Metric "unresolved bugs per month" for example, one could fix a target $t1$ of 20 for the period $p1$, a target $t2$ of 15 for the period $p2$, etc. Thresholds are used to qualify the gap between the actual value calculated by the KPI's Metric and the targeted value.

Attribute Name	Attribute Type	Description
Name	Text	Name of the KPI
Description	Text	Description of the KPI
Viewpoint	KPIViewPoint	Domain the KPI belongs to
Metric	Metric	Metric that calculates the value of the KPI
Timescale	Timescale	Granularity of the time periods of the targets
Targets	Target	Set of targets that are associated with the KPI

Table 9.7: KPI Artifact Attributes

A target indicates the year and the number of the period for which it applies. The period $p3$ of a monthly KPI would for example designate the month of March. The attribute value indicates the value which should be reached and thresholds holds a set of Threshold that help qualify an achieved value in respect to the targeted value. A list of these Attributes is provided by Table 9.8.

Attribute Name	Attribute Type	Description
periodNo	Number	Period for which the target applies
Year	Year	Year for which the target applies
value	Number	Value that should be achieved
thresholds	$\mathbb{P}$Threshold	Set of thresholds that help qualify the gap between an achieved and a targeted value

Table 9.8: Target Artifact Attributes

A **Threshold** qualifies the values of a specific value range. For example, the threshold t1 could stipulate that values between 50 and 100 are considered as "good", whereas threshold t2 could stipulate that values between 30 and 50 "need improvement".

A Threshold is composed of a $valuerange$ and a $qualificator$. $Qualficators$ could for instance be based on letters (A, B, C, D, E), score names (Exceptional, Very Good, Good, Needs Improvement, Unacceptable) or colours (green, yellow, red). The complete list of Threshold artifact's attributes are listed in Table 9.9.

Attribute Name	Attribute Type	Description
valuerange	ThresholdRange	Range of values of the threshold
qualificator	Qualificator	Qualification of the value range

Table 9.9: Threshold Artifact Attributes

StrategyManagement Activity. This activity allows the creation and management of Objectives of a project. An **Objective** is "a thing aimed at or sought; a goal" part of project's strategy. A typical Objective could for example be to increase the number of downloads of a package by 10%. In order to support the Objective, critical success factors are identified and corresponding KPIs are designed then assigned to the objective. In our case, KPIs such as number of bugs per package or package popularity could be strategic drivers which support the achievement of the objective. An Objective has a name and a detailed description. It further features one or several KPIs that act as strategic drivers. Its attributes are listed in Table 9.10.

Project Objectives can be defined in terms of KPIs by project managers. Information built can be

Attribute Name	Attribute Type	Description
Name	Text	Name of the Objective
Description	Text	Description of the Objective
Kpis	$\mathbb{P}$KPI	KPIs that support the achievement of the Objective

Table 9.10: Objective Artifact Attributes

interpreted and used in order to define a strategy map for the project. This map can be used to refine and tune the adopted strategy, add weights to the different objectives in order to define their importance. KPIs related to different Objectives can be found and then activities related to them can then be extracted in order to indicate where effort should be focused within the project. The operations provided by the strategy management activitiy is listed in Table 9.11.

Operation Name	Parameters and Return Type	Description
declareObjective	(Text_{name}, $\text{Text}_{description}$, Project):Objective	Creates a new Objective for a Project
addKPI	(KPI, Objective):void	Adds a KPI to an Objective (KPI must be associated with the project of the Objective)
removeKPI	(KPI, Objective):void	Removes a KPI from an Objective
getKPIs	(Objective):$\mathbb{P}$KPI	Retrieves all KPIs associated with an Objective
influencesObjectives	(Project, KPI):$\mathbb{P}$Objectives	Retrieves all Objectives depending on a KPI within a Project
setCriticality	(Objective, Number):void	Sets the criticality of the Objective

Table 9.11: Strategy management operations.

EventsObserver Activity. In order to enable Event-driven PRM programming and process management and handle PRM calls scheduling and PRM calls monitoring, an EventObserver Activity has been added. This Activity is meant to be implemented by different Activities needing to be notified of Events occurring within the PRM. As the name of the activity suggests it, the adopted approach is based on the Observer design pattern [47]. Table 9.12 lists the different operations provided by this Activity.

Operation Name	Parameters and Return Type	Description
notify	(Event):void	Notifies the Activity that an Event occurred.
addEventTypeToObserve	(EventType):void	Adds an EventType the EventObserver has to observe.
removeEventTypeToObserve	(EventType):void	Removes an EventType the EventObserver has to observe.
getEventTypes	():$\mathbb{P}$EventType	Returns all the EventTypes the EventObserver is observing.

Table 9.12: Event Observing management operations.

Further, in order to be able to notify Activities implementing the EventObserver Activity through the PRM Event Management Activity, the latter has been completed with operations enabling the registering and unregistering of EventObservers. EventObservers can declare themselves as willing to be notified each time an EventType occurs. Table 9.13 presents the operations having been added to the Event Management Activity proposed in the PRM core.

Operation Name	Parameters and Return Type	Description
registerEventObserver	(EventObserver):void	Registers an EventObserver to be notified of Events occuring.
unregisterEventObserver	(EventObserver):void	Unregisters an EventObserver to be notified of Events occuring.
addEventTypeToListenTo	(EventObserver, EventType):void	Adds an EventType the EventObserver has to be observing.
removeEventTypeToListenTo	(EventObserver, EventType):void	Removes an EventType the EventObserver has to be observing.
getEventTypes	(EventObserver):$\mathbb{P}$EventType	Returns all EventTypes an EventObserver is observing.

Table 9.13: Extended Event Management operations.

SchedulesManagement Activity. As a Metrics scheduling feature to the PRM required a mean to schedule Metrics evaluation, instead of hardcoding such a feature which would only be usable for Metrics scheduling, we decided to extend the PRM with a Schedules Management Activity to schedule any PRM operation call and which can be used by any other Activity needing it. Table 9.14 lists the operations offered by this Activity. Schedules

Operation Name	Parameters and Return Type	Description
createSchedule	(Date, Activity, λ, Directory$_{parameters}$):Schedule	Creates a Date-based Schedule
createSchedule	(Date$_{start}$, Date$_{stop}$, Number$_{frequency}$, Activity, λ, Directory$_{parameters}$):Schedule	Creates a frequency-based Schedule
createSchedule	(Event, Activity, λ, Directory$_{parameters}$):Schedule	Creates an Event-based Schedule

Table 9.14: Schedules management operations.

Such schedules management Activity is used as the base for the Metrics Scheduling Activity. It offers the ability to create Metrics schedulers per Project and per Actor. Then Metrics can be scheduled in the scope of a MetricsScheduler. When a Metric is evaluated for a MetricsScheduler, the result is then sent as a MetricsEvent containing information about the MetricsScheduler having triggered the evaluation. If the result has to be captured, the capturing Activity has to be registered as an observer for this Event. Table 9.15 lists the operations provided by the ScheduledMeasurement Activity.

Operation Name	Parameters and Return Type	Description
createMetricsScheduler	(Text$_{name}$, Project, Actor$_{owner}$):MetricsScheduler	Creates a MetricsScheduler for a Project and defines an Actor as its owner
getSchedulers	():ℙMetricsScheduler	Returns all MetricsSchedulers
getSchedulers	(Project):ℙMetricsScheduler	Returns all MetricsSchedulers registered for a Project
getSchedulers	(Actor):ℙMetricsScheduler	Returns all MetricsSchedulers owned by an Actor
scheduleMetric	(MetricsScheduler, Metric, Schedule):Schedule	Schedules the execution of a Metric
unscheduleMetric	(MetricsScheduler, Schedule):void	Removes the schedule of a Metric
getSchedule	(Metric, MetricsScheduler):ℙSchedule	Gets all Schedules for a Metric
getScheduledMetrics	():ℙMetric	Gets all scheduled Metrics
getScheduledMetrics	(MetricsScheduler):ℙMetric	Gets all scheduled Metrics for a MetricsScheduler

Table 9.15: ScheduledMeasurement management operations.

Chapter 10

Testing framework for J2ME applications

The PRM, and EDOS-PRM, has also been extended in order to describe the activities of a simple collaborative testing framework for Java Micro Edition (J2ME) [176] applications [1]. The goal of this framework is to provide means to application developers and users to detect if an application will be runnable on a specific mobile phone device and to share this information with other users using the same model of device.

Each mobile phone device running a java virtual machine holds a set of Java Specification Requests (JSR) [90] defining the J2ME configuration of the mobile device. The minimal set of JSRs' a device runs consists of a given version of CLDC [20] (version 1.0 "Jsr30" [98] or version 1.1 "Jsr139" [97]) and a version of MIDP [122] (version 1.0 "Jsr37" [99] or version 2.0 "Jsr118" [96]). Other JSR packages can be added to the J2ME platform by mobile phone producers such as for instance the Java API for Bluetooth 1.0 "Jsr82" [100].

The issue faced by application developers is that the list of available additional JSRs, and even the list of core JSRs such as the CLDC and MIDP cannot be retrieved through any mean provided by current virtual machines. This leads to situations where needed packages and APIs are not available on the device. Such situation can not be detected prior to running the application, and the latter cannot self test themselves to check if they are runnable on the device.

As code introspection is not available under Java Micro Edition, existing testing frameworks such as Junit cannot be run on it. Thus, The testing framework for J2ME is a framework based on the PRM is made available to achieve these tests trying to provide the best possible workaround to the introspection issue. It provides means to create tests needed to detect a configuration, run them on a mobile phones and to detect if a given user-defined application (or any JSR) will be runnable on the provided platform depending on the results obtained. While performance Testing is not the main purpose of this framework, it has been designed keeping in mind possible extensions to achieve such tests.

10.1 Tackling the configuration retrieval issue

The framework enables information retrieval concerning the configuration of a mobile phone device running a J2ME virtual machine. Gathered information can be split in two sets:

- The properties of the available Virtual Machine
- The list of available JSR's along with their properties

The workaround to the introspection issue is to get the Java configuration of the mobile phone device by checking the availability of all JSR's as well as their configuration. To achieve this, the testing

[1] This work has been part of the master thesis of Nicolas Riou and Simon Pachy, two students from the University of Savoie

frameworks provides various Micro Edition Tests extending EDOS-PRM Tests. Two types of tests are available, the ones testing the availability of a JSR, and the ones testing its configuration.

The availability of a JSR is tested by running try/catch blocks retrieving specific classes that should be provided by the JSR using the java `Class.forName()` method. If a class is not available, the corresponding exception is catch and decision can be taken in order to stop the application, modify the way it has to run (by trying to detect an alternative implementation for instance) or just display the result of the test.

```
try {
// retrieval trial of a specific class of jsr120
Class.forName("javax.wireless.messaging.MessageConnection");
}
catch(ClassNotFoundException){
// jsr120 does not seem to be supported
...
}
```

The Java virtual machine configuration can be retrieved using the `System.getProperty()` method providing specific parameters. This enables the retrieval of the version of the CLDC being run and of the MIDP as well. Further it also provides useful information about the platform (available memory, total memory, region, etc.) If a JSR is available, some information about its configuration can also be retrieved using the `System.getProperty()` method or by using a method specific to the underlying API as the `Graphics3D.getProperties()` method for the "Jsr184".

10.2 J2ME Testing PRM Extension

The J2ME testing framework has been designed around the PRM and EDOS-PRM which have served as guidelines for Test management and for describing the J2ME testing Activity. Figure 10.4 depicts how the PRM has been extended, Figure 10.1 lists the main Artifacts added by this extension and the related activity is presented in Figure 10.2. We detail these elements in the following paragraphs.

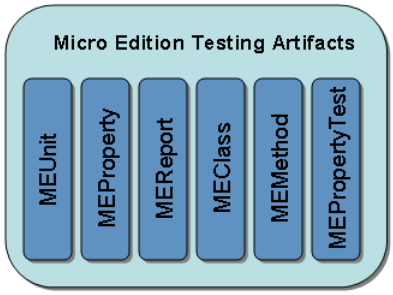

Figure 10.1: Artifacts of the Micro Edition Testing Extension

Micro Edition Testing Activity. This activity is meant to handle the testing of applications (be it a JSR or a user application) through the push of applications and tests on the server running the ME Testing Activity, the retrieval of these elements by different devices running different configurations, the registration of the results on the server and the retrieval of them for analysis. The testing part itself is running on the device, and thus is external to the PRM extension. Table 10.1 lists the operations related to the METesting Activity.

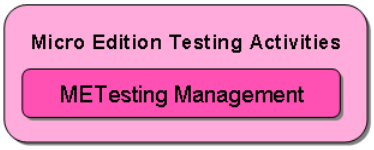

Figure 10.2: Activities of the Micro Edition Testing Extension

Operation Name	Parameters and Return Type	Description
register	(MEUnit, $\mathbb{P}$MEProperty$_{MEUnit}$)	Registers a MEUnit application along with corresponding MEProperties to be tested.
retrieveTests	(MEUnit):$\mathbb{P}$MEProperty	Retrieves all properties to be tested for a MEUnit.
registerResults	(MEUnit, PlatformConfiguration, MEReport):void	Registers a test report for a MEUnit running on a PlatformConfiguration.
isRunning	(MEUnit, PlatformConfiguration):Boolean	Indicates whether a MEUnit is running on a PlatformConfiguration or not.
retrieveConfigurations	(MEUnit, Boolean):void	Retrieves all configurations on which a MEUnit is running / failing.
retrieveApplications	(PlatformConfiguration, Boolean):void	Retrieves all applications running / failing on a configuration

Table 10.1: Micro Edition Testing management operations.

A **MEUnit**, is any application to be distributed in order to be run on a J2ME platform, and which can be tested through the METesting Activity. For instance, JSRs and user applications are specializations of the MEUnit Artifact.

Each MEUnit involves a set of **MEClass**, and of **MEProperty** which can be tested. A MEClass lists the MEMethods which have to be available for the MEUnit to be runnable. A MEProperty, is a property the applications developper wishes to test, such as available memory, a system MEUnit configuration or any other property, such as the configuration of a given API.

MEProperties can be compound of SimpleMEProperties. A set of **MEPropertyTest** is associated to each MEProperty; it represents the tests to be run corresponding to it. The **MEUnitTest** is a specific MEPropertyTest verifying if the associated MEMethod is available on the tested device.

Once all tests for a MEUnit are run, a **MEReport** is generated indicating which MEPropertyTests succeeded and which ones failed.

Figure 10.3 illustrates the architecture involving the METesting Activity. A server running this activity offers services to application developers, testers and users. Application developers register applications on the server, and testers can test them on a given configuration. The resulting report is then registered on the server through the activity and can be retrieved by any user. Furthermore, application developers are able to retrieve all results concerning their application to find out which configurations cannot run them. Similarly, users can retrieve a list of all applications that can be run on their device.

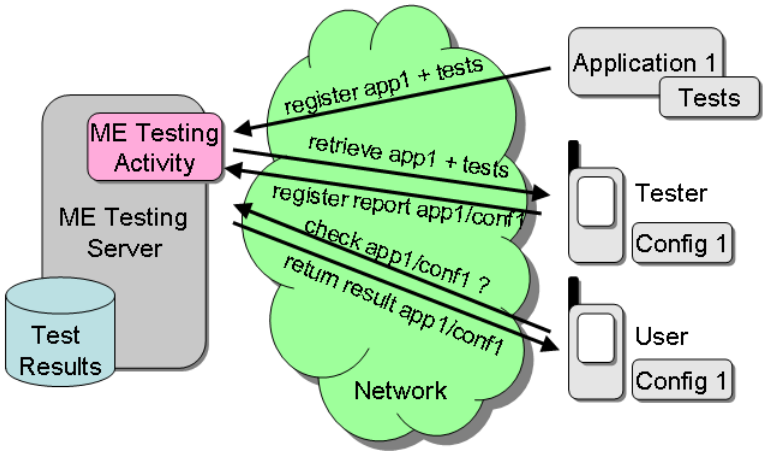

Figure 10.3: PRM J2ME Testing Framework scenario

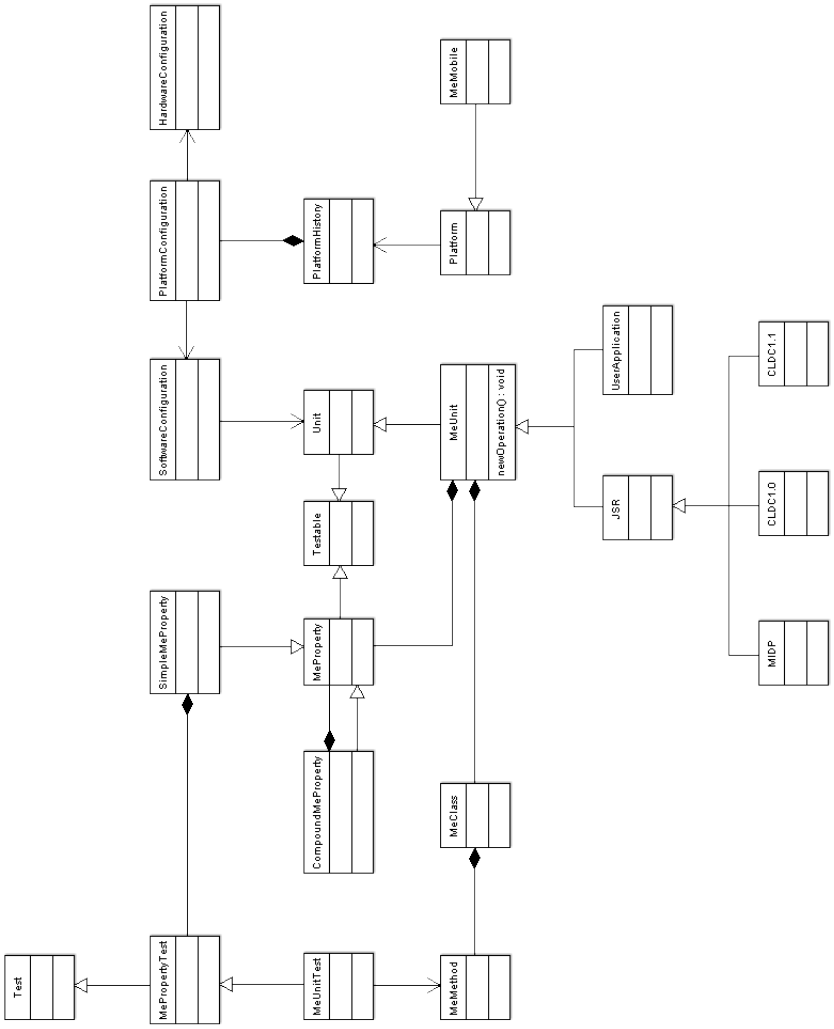

Figure 10.4: PRM Extension for the J2ME Testing Framework

Part IV

Conclusion and Appendices

Chapter 11

Conclusion

We highlighted in this Thesis the evolution the F/OSS environment has been subject to during the past decades, from the very first idea of *Open Source* to the current ongoing integration of F/OSS with the Enterprise environment. The F/OSS environment is a philosophy and methodology providing an incredible source of different competences. Enthusiastic community members contribute to the development of F/OSS projects undertaking responsibilities in every part of the so called F/OSS Process, the process embedding different activities bound to a F/OSS project, such as production, testing, recruitment, design, debugging, administration, distribution, and any other activity that may be added over time.

While such a methodology is an example of cooperation and collaborative work, as project grow and as time passes, the F/OSS environment is subject to specific constraints. These constraints are mainly bound to the high distribution of the F/OSS environment, contributors and activities, and to the constant need to foster interest in the projects, to involve new contributors for new tasks or to replace departing ones. Further, as there is no common ontology for the F/OSS environment, buzzwords like project, test, package, debugging, distribution, can have multiple meanings which are not documented and which thus make difficult to compare projects and make them interoperate.

Projects start small, and grow big. Their informational needs change over time, their activities also do so. The bigger a project is, the more it needs to interact with other projects, the more it requires a means for optimizing the usage of its resources, understanding how the different processes are interdependent. The complexity induced by such a distributed environment makes difficult to manage the process, measure it and provide efficient means for assisting projects' decision makers. Projects need structures helping them grow, and providing guidelines making them avoid known obstacles which when considering global knowledge about F/OSS could be predictable if it was any way to fix it.

Main current challenges the F/OSS environment is facing have been outlined in this Thesis and related requirements have been extracted. Based on this analysis, we proposed a way to structure F/OSS information in order to be able to streamline and manage the F/OSS process, considering the particular constraints implied by this environment. This work resulted in the definition of F/OSS Process Reference Model (PRM), a model for describing and structuring the F/OSS process. Apart from the descriptive power of the PRM, two other aspects are worth being mentioned.

The first aspect is bound to a central keyword in the F/OSS environment context: *involvement*. The PRM uses contributors involvement as a driver for supporting the F/OSS process. To achieve this, community members' interests, knowledge and competencies are used to achieve a match-up between Projects, Roles, Processes, Tasks and people willing to contribute.

The second aspect of interest is *sharing*. This word often remembers the action of *sharing sources*. While this meaning is correct, the PRM enables more complex sharing. The structuring power of the PRM, enables PRM users to describe precisely the resources they use, describe how and when they can be substituted by other resources. Similarly the activities can also be described, together with the different integrity rules related to them. And it is also the case for process and roles descriptions and

even metrics which are all reusable. Indeed with the PRM, these descriptions can be shared among projects to ease their integration, communication, and understanding of what they are effectively doing. By providing access to this information it allows Projects to detect then replicate best practices.

All these aspects make the PRM a step toward F/OSS project management where not only produced content is considered as open source, but so are human resources and organizational decisions.

11.1 Contributions

The contributions of this Thesis are the following:

F/OSS process management panorama. This Thesis has provided a common ground of understanding of the F/OSS problem area and of the challenges emerging in the context of modern and growing F/OSS projects. We highlighted the evolution this environment has been subject to during the past decades also considering its current integration with the Enterprise environment. Main problem areas have been outlined and related requirements have been extracted. Based on this analysis, we defined the elements which are required to structure F/OSS information keeping as our objective having the ability to streamline and manage the F/OSS process, considering the particular constraints implied by the F/OSS environment.

A Model for F/OSS. Based on these requirements, we formalized and presented a process reference model (PRM) to model the activities, roles and resources of the F/OSS process. This model allows F/OSS activities to be more efficiently handled. The goal of the PRM is to define the key content and community artifacts of the F/OSS process and to formalize the relations between these. We believe that this precision allows inefficiencies in F/OSS processes to be detected and eliminated, and F/OSS-oriented Information System (FIS) to be designed. The PRM permits the abstraction of bazaar observers to be implemented with consideration for the F/OSS environment and its evolution. It provides a flexible way to structure information through Artifacts and manipulate them through basic primitives provided by Activities. Processes can be built on top of these activities and be measured even if no central point of control is available. The model is extensible as both new Artifacts and Activities can be declared. Processes can be built on top of these activities and be measured even if no central point of control is available. The model is extensible to support any kind of F/OSS activity.

Model Application. The PRM model has been implemented in the context of the EU 6th Framework project EDOS (Environment for the development and Distribution of Open Source software.)[1] The model has been extended to fit the particular constraints of the EDOS project (EDOS-PRM). Particular focus has been done on the following aspects: handling of transversal metrics able to help F/OSS project managers in their decision making, and the usage of the PRM in a communitarian testing environment.

11.2 PRM Strengths

The PRM provides through Artifacts a generic and uniform means for representing and describing any F/OSS related information. Indeed, Artifacts can be used to represent content, community resources or any other type of resource. Artifacts can be matched in order to compare them. Artifact differentiation depends on their type, on the type of their Attributes and the list of their Attributes. Artifacts provide thus a powerful mean for information sharing on distributed Information Systems. The substitutability relation goes beyond the usual comparison relation and provides extended flexibility to the PRM Model, which relies on it in order to provide a flexible artifact lookup of Artifacts.

[1] grant number FP6-IST-004312.

The PRM Model proposes to handle distributed Information Systems as sets of Activity interfaces. These interfaces provide a list of operations they are able to handle. They can be implemented, shared along with the Artifacts they use. The PRM Model handles all core activities needed for building F/OSS Information Systems: community, project, process, rights, roles, metrics, tasks, events, log, artifact and activity management.

The PRM provides a means for ensuring F/OSS Project interoperability by design through the use of Artifacts, Activities and Processes. These elements can provide a common ground of understanding to F/OSS projects as they model the data used by projects, the different ways this data can be handled and the processes using it. For instance, Projects can agree on what a Test exactly is what is the information needed to express it. Further, Projects can also agree on what operations a Testing activity has to provide, and on the integrity rules to be enforced.

The PRM approach enables the integration of activity processes in a global view. This enables project managers to see how they impact on each other, discover bottlenecks, and know where effort has to be put to improve theF/OSS Process. PRM Processes can be chained and synchronized using Event elements. This synchronization can be achieved inside a project but also on a cross-project manner. Indeed outside Projects can ask to be notified of the end of a specific process run by a partner project to start their own Processes. For instance, a testing community may wish to be automatically informed of beta releases produced by different projects to help them test their products.

Metrics allow to describe measures to be taken within F/OSS projects. F/OSS project activities can be distributed with no central point of control. In this context, metrics can involve multiple activities. Being able to evaluate transversal metrics involving these distributed activities is needed to achieve thorough analysis of F/OSS information systems, evaluate their processes and thus be able to improve them.

Considering the management of F/OSS human resources, the PRM goes beyond usual community handling. While the latter lists the different contributors and provides common information about them, the PRM aims at considering community members interests, knowledge and competences when enrolling contributors, attributing tasks to members, etc. Further, the PRM provides a means to know exactly which Actors contribute to a project, and know what are their tasks, obligations and rights.

Finally, the PRM is extensible. New Artifacts, Activities, Processes, Events and Metrics can be added to adapt the model to the needs of the projects using it. For instance, a new Activity dealing with e-learning could provide an e-course Artifact and operations for defining the topics of the course, for associating a community member as teachers, and could embed an integrity rule defining the competencies threshold for accepting a community member as a teacher for the course.

Having such a structured common ground of understanding enables the creation of meaningful and efficient F/OSS dashboards. With such tools, project managers can analyze processes, used and available resources, and compare processes with other projects' processes. The extensibility enables projects to grow and to adapt to the environment, while the possibility to share, measure and analyze projects' structures enables *in fine* process improvement.

11.3 PRM Constraints

However, while the PRM promises interesting benefits, some organizational, and technical constraints have to be considered.

Organizational Constraints. As the model introduces new elements and forces the description of previously unclear ones, new roles, tasks and responsibilities have to be defined in order to handle them. Indeed, in order to use efficiently the PRM model, projects must have contributors undertaking particular support roles. Among other roles which have to be assigned when using the PRM one can find Artifact manager, Activity Manager, Process Manager, Role manager, Project describer, Community describer, Task manager and Task evaluator. For instance, the Activity Manager role has to gather information

about the different operations offered to the community by the project, and then create the corresponding Activity interfaces. The operations of these interfaces have to use the different Artifacts chosen together with the Artifacts description role. This activity description also covers the definition of the integrity rules related to each operation of these activities, their association to the operations and referencing for information purpose. Another example of new role is the Task Evaluator role, which when a task is done, has to evaluate what has been done in order to update information about the Competences of the different contributors in order to better assign future tasks.

Technical Constraints. PRM's efficiency is tightly coupled to the availability and precision of information. To benefit from the for-mentioned strengths, the PRM has to be able to get the right information at the right time. Moreover, information correctness must be ensured to have reliable information Systems built on top of the PRM. This raises questions about how information has to be gathered. This task can be undertaken through the involvement of community members and roles such as described in the previous paragraph or automatic information gathering and Artifact completion when applicable. Such an automation involves the adaptation of existing tools. This includes the need to implement Activity interfaces in order to provide the different operations while respecting their signature. Further it also implies the need to respect the format of Artifacts used by the Project, and integrate Artifacts and the PRM model with these tools. Finally, projects must be able to publish their Activity implementations along with their Activity interfaces and corresponding Artifacts to share them with other projects. Similarly, they should have a means to discover Activities provided by other Projects in order to integrate them if needed.

Some of these constraints are not usual for the F/OSS community. Indeed these new roles are implying further effort from contributors which may seem to be too heavy. As such, this implies that these roles as well as the strengths of the model must be clearly explained to the community in order to foster motivation and thus find contributors willing to participate in these tasks.

11.4 Perspectives

We end this Thesis by describing some possible domains which can benefit from the PRM. We also discuss possible improvements that can be made to the model.

F/OSS project management. The PRM brings all the means needed for enabling efficient F/OSS project management. Imagine a framework gathering activities, artifact types, process descriptions as well as roles and metrics and putting them in a repository available to project managers. Such a repository could serve for instance as a basis for comparing the processes used by various F/OSS distributors such as Caixa Magica, Debian and Mandriva linux. Project managers could pick activities, artifacts and processes from these repositories to build their own project, they could use existing processes as is or improve them to fit particular needs. They could do the same with metrics. With time, these repositories would provide information to fit many types of situations, be it on projects' activity architecture definition level, the process definition level, community management level or process measurement level. Such global sharing of project management means and knowledge could improve the way F/OSS projects are conducted, and could give access to small projects or projects with no real in-house management competencies to good practices.

Toward a F/OSS ontology. A promising perspective for the PRM is to turn it into a ontology for F/OSS. The description of the different aspects of the F/OSS process is needed in many situations. From the perspective of new projects such an ontology could serve as the design of their information system, highlighting the information they have to handle in the perspective of growing, but also defining the

vocabulary to use to understand other projects and to be globally understood. Large projects could use this ontology as a common ground of understanding they could use to communicate with other projects, integrate their processes, etc. From a research perspective, having a F/OSS ontology will help research to be globally understood. This would also make sure that the scope of the research is understood as the limits of what can be done is described by the ontology. For instance, if the ontology defines that there is no possibility for reverting the execution of a process, none can expect such feature to be possible. In a more global perspective, such an ontology would support the build of information systems and would define the integrity rules that are to be respected for all F/OSS operations.

Content aggregation. With projects using the PRM, the collection of information and presentation of aggregated content can be made easier. Indeed the common ground of understanding the model provides, as well as the mechanisms enabling the comparison and substitutability of F/OSS artifacts, make the PRM an excellent support for aggregating F/OSS information trough infomediaries, or project-related and cross-project dashboards. Unlike usual aggregation of content like RSS feeds, the PRM provides an understanding of handled information, which enables its comparison independently of its nature be it a resource, or a role, process, etc.

We can thus imagine specialized infomediaries analyzing and comparing how different project manage processes, which activities they use, comparing projects with each other on a process basis or community involvement basis, trying to extract best practices as well as errors to avoid. Such PRM based infomediaries could also provide a real time survey of the F/OSS environment highlighting hot topics. It would provide useful information about the areas needing more community involvement, highlighting the reasons making this involvement required in terms of impact on depending projects, and thus in term of potential global risks for large projects and thus companies using them.

Credentials' based rights management. An interesting improvement to be made to the PRM could be to manage security through credentials. By expressing Rights given when assigning a Task to an Actor as credentials and using integrity rules to check them, only Actors being given the correct credentials could access an activity operation. This would be particularly useful in the case of distributed processes and especially in the case of cross-project processes. Indeed, instead of having to handle the rights for actors external to the project, projects could delegate to other projects the right to create credentials to access some activities in the context of cross-project processes.

Such a security scheme would ease the way the PRM manages rights. For instance, transversal metrics calculation implies that the actor executing the metrics has the right to access to the different operations of the PRM. Being able to give a special credential to an Actor having to create and execute metrics calling any activity operation would avoid having to give him new rights everytime new Activities appear.

Using credentials could also open interesting perspectives in the field of compliance and enterprise policy management [93]. Regulatory frameworks such as the Sarbanes-Oxley Act (SOX) or Basel II require processes to enforce given rules and to adapt their execution depending on particular events. For instance, SOX defines that some actions cannot be done as long as the Chief Financial Officer (CFO) has not approved the action, it also defines that the CFO must be kept informed of other action. Such policies could be integrated to PRM Activities' operations as integrity rules. One can imagine that operations could return credentials ensuring that a particular task has been done, keeping track of what has been executed, and that operations would trigger events when needed.

Exceptions handling is another aspect where credentials can be useful. A strict security management scheme leaves no place for unexpected situations where access to a resource is legitimate, but was not anticipated when assigning the rights to the different actors [92]. However, depending on the sensibility of the information to be accessed or modified, and depending on the presumed ability of the person trying to do the action, the action might be accepted in some situation if the severity of a bad action is

considered as a less worst than doing nothing. Imagine a critical bug detected in a library and nobody to commit the fix as the official committer is gone on vacation. If the Actor is competent enough (for instance, he's a committer for another project and has all the competencies for taking this decision), one might want to authorize the action and log it as exceptional.

PRM and Trust. When talking about distributed environments where anyone may contribute to a project, and where anyone is invited to do so, trust problems appear. Indeed, projects need information about newcomers to better evaluate them and propose task more adapted to their competencies. Thus, projects may want to access information concerning the involvement, competencies of the actor in other projects. However, while some projects can be perfectly managed, other, may be poorly managed, and contributors be poorly evaluated. Similarly, some projects may boost the information concerning friendly contributors. Another aspect concerns the trust the community has in the Project itself, and in the contributors evaluating other contributors and Projects.

All these aspects in a distributed environment based on information sharing highlight the need for trust features. The PRM could be extended with such features enabling the evaluation of projects and actors, but also of activities' implementation, processes, metrics and roles. Mechanisms concerning Artifacts' substitutability could also be evaluated. Similarly, the trust in the topics handled by the different projects could also be defined in order to detect current F/OSS environment trends. Having the ability to build such a Trust image of F/OSS environment is essential for easing project management.

11.5 Free and Open Source Process Management

The application area of the Process Reference Model is large. A key aspect of this model is that it structures the F/OSS environment at different levels providing thus support for developers, application architects, but also project managers, analysts, decision makers and last but not least simple community members. Unlike usual approaches placing produced content in the middle of the talk, the PRM highlights the key role of projects' activities, and then of processes built on top of them, resources which are involved and contributors. All these elements are described precisely by the PRM in order to reach one goal: better coordinate activities, better integrate the community with projects, and thus better understand the F/OSS process.

The PRM is a means for gathering knowledge about the F/OSS environment and process management. Repositories can be built to keep this knowledge and make it globally available to the F/OSS community. Such an approach meets the F/OSS philosophy. The aims of F/OSS are to share code to make it available to people, while enabling improvement by the community. Creating and sharing repositories containing F/OSS process management knowledge goes beyond this initial principle. It does not only apply the F/OSS approach to the content, but also to all aspects related to the management of the F/OSS process. Sharing such information can help guide projects, better organize them as well as their community, better detect problems and better streamline the process. The structures the PRM provides are a first step toward what we can call Free and Open Source Process Management.

Appendix A

Working Method Employed

We describe in this Appendix the approach that we have been using to highlight the Activities, Artifacts and Roles appearing in the F/OSS process. The technique is known as a *conceptual map* [49] and results in graphs such as the one presented in Figure 5.3. This is a graphical technique where we start by placing the key concepts of the process being modeled, and then identifying those which are linked. In a second phase, we add further concepts to qualify the meaning of these links, for example, the concept of ProjectRole expresses the roles that a project adopts for its management. In a third phase, we draw *information zones* – represented by colors in the graph – that identify the concepts that are closely related based on the information flows and roles appearing in the zone. The conceptual map can then be used as a foundation for building UML Use Case Diagrams, Class Diagrams, etc. or any other type of representation be it textual, graphical or formal. For example, there is an information zone for managing Roles used within a project, check required competences, in order to assign Processes to Actors in the scope of a Project. This particular information zone is presented in Figure A.1. The combination of the information zones depicts the knowledge of the field being modeled (F/OSS in our case).

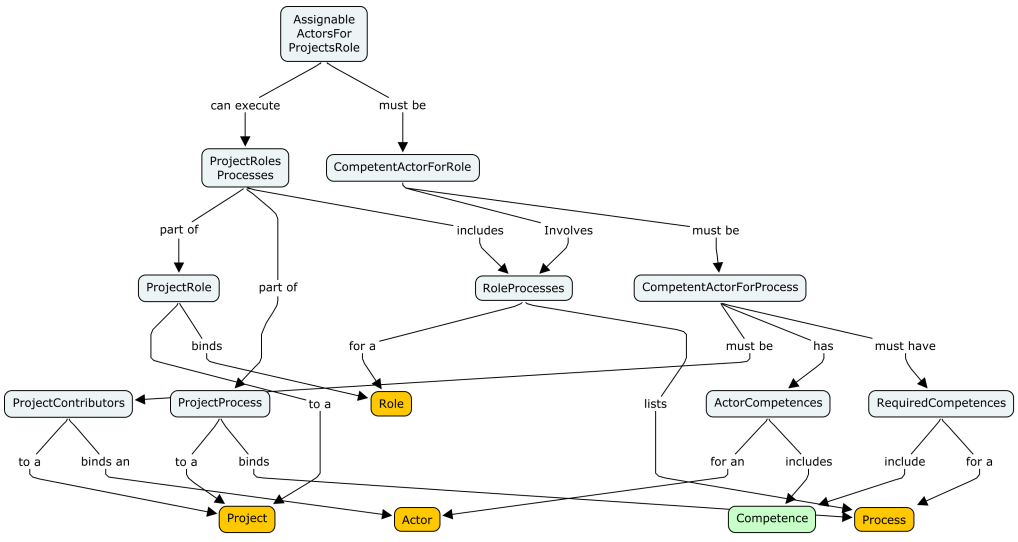

Figure A.1: Conceptual Map of Core PRM Artifacts

The key to this use is that each information zone can be refined independently of other zones. The separation of concerns is thus already present in the global design phase, and remains present through the whole development, implementation and operation phases. This aspect was particularly important in our case where we are dealing with a highly distributed environment.

Indeed, there are three principal reasons which make this approach particularly adapted to describe the activities involved in the F/OSS process.

1. **Boundaries and description.** First, as stated before, each information zone is associated with a set of resources, roles, with particular responsibilities for executing related tasks. Further each information zone is associated with information flows, which describe the way resources made available can be used within the zone and outside it. Describing the concepts involved in each information zone shows exactly what information is needed for handling the resources, what are the roles involved, and what the boundaries of these roles are.

2. **Overlaps.** Second, the map shows clearly that some information zones overlap, and that some concepts of the zone are completely independent from the rest of the map. Overlapping information zones imply that concepts in the intersection need to be analyzed as they affect more than one zone. If overlapping zones cannot be avoided in the design, then one of the intersecting zones has to be attributed ownership of the involved concepts and thus resources – or a new information zone must be defined to cover the intersection. The owning zone is the only one in which refinement and development of concepts takes place. Other information zones are simply "clients" of these concepts and have to negotiate modifications if needed. In Figure 5.3 for instance, the concept of process is used by several zones such as role management, task management or rights management. Ownership is attributed to the Process management zone whose aim is to globally manage the design, implementation and operation of the processes independently of the zones that use this process concept. In contrast, non-overlapping parts are internal to each information zone, and so can be managed with no risk to the development of others.

3. **Extension impact.** Third, the addition of a new concept – and its interaction with existing concepts – can be easily modeled. An addition may imply modifications of existing information zones or even the creation of a new specialized zone. This feature is probably the most important as it allows at any moment to evaluate the global impact of the addition of a new concept to the whole model. This change cannot easily be modeled by formalisms that insist on use case scenarios as this is too focused.

Appendix B

Extending the PRM for handling Linux distributions

The process of distributing Linux Distribution implies the extension of the PRM core. Such an extension involves the addition of new activities related to the different aspects which are specific to this process as well as the addition of corresponding Artifacts. We applied here the methodology for extending the PRM model with adequate Activities and Artifacts, which was proposed in Chapter 6.

The Activities corresponding to the linux distribution use case are summarized in Table B.1 and related Artifacts are summarized in Table B.2. This appendix focuses on detailing these Activities and indicating which Artifacts each Activity provides and is responsible for. The substitutability relations of these Artifacts and required integrity rules which have to be enforced have been declared in [115]. Some of them are presented in Appendix C which illustrates some of the benefits resulting from this PRM extension.

Production Management	Distribution Management
Defect Management	Dependencies Management
License Management	Platform Management
Security Management	Test Management

Table B.1: PRM Model extension: F/OSS Activities Summary

Platform Management. A Platform is an Artifact representing a machine running hardware devices and software applications. Each Platform has a single Administrator, and multiple users as well as multiple PlatformAddresses. The PlatformAddress is a user defined Artifact which can be an IPv4 address, an IPv6 address, a MAC address or any other type of address. Administrators install Units as well as HardwareDevices on Platforms, nad have to provide a corresponding configuration. Units installed on a Platform can be retrieved and shared with other Platforms. The Platform Management Activity provides a means for HardwareDevices and Units installation, uninstallation as configuration on Platforms. Other operations enable the retrieval of past configurations and the revert to these past configurations. Table B.3 provides the list of operations provided by the Platform Management Activity, while Table B.4 provides a list of Platform Artifact's attributes.

The PlatformConfiguration Artifact describes the complete configuration of HardwareDevices and Units installed on a Platform. This Artifact is used to keep track of installation details of both Hardware-Devices and Units, in order to help analyzing reports in case of Defects detection or Tests failures. This can help building a knowledge base, which can be useful for debugging, for future development as well

Artifact	Description
Bundle	Collection of Units
Comment	Opinion, input made and signed by an Actor
Defect	Error in Testable Artifacts leading to unexpected behavior
DefectStatus	Defect Snapshot providing all information about the state of a Defect at a given time
DefectStatusType	Position of a Defect in its life cycle
DefectSeverity	Estimated severity of a Defect
DefectPriority	Indicated priority of the Defect for fixing it
Functionality	Functionality provided by F/OSS content
HardwareConfiguration	Set up of a Hardware device
HardwareType	Hardware device type
License	Usage rights and obligations associated to a F/OSS content
Patch	Unit designed for correcting a defective one.
Platform	Computer made of different Hardware devices and on which different software are installed
PlatformConfiguration	Description of the sets of hardware and related software configurations installed on a Platform
PlatformAddress	Any mean to locate a Platform on the network
Signature	Assertion that a F/OSS action has been done by an Actor
SoftwareConfiguration	Description of a Unit's set up, defining chosen installation options, etc.
Test	Manual or Automated check that have to be run on a Testable artifact in order to assess QA
Testable	Artifact for which Tests can be defined and run; can be a Unit, a Test, etc.
TestReport	Result of the application of a Test listing what happened during its execution
TestResult	Test result
TestType	Test execution type
Unit	Description of a F/OSS content unit indicating various information such as the UnitLocation, existing dependencies or conflicts or covered topics within the F/OSS Process.
UnitLocation	Artifact indicating where on the network the content described by a Unit can be retrieved from
UnitStatus	Description of Unit development maturity
UnitType	Indication of Unit content type, be it source code, binary code, documentation, etc.
UnitVersion	Any type of versioning information

Table B.2: PRM Model extension: F/OSS Artifacts Summary.

Operation	Description
declare(Platform):void	Declares a Platform
addHW(Platform, HardwareConfiguration):Boolean	Adds a HardwareDevice to a Platform
removeHW(Platform, HardwareConfiguration):Boolean	Removes a HardwareDevice from a Platform
install(Platform, Unit):Boolean	Installs a Unit on a Platform
uninstall(Platform, Unit):Boolean	Uninstalls a Unit from a Platform
install(Platform, Channel):Boolean	Installs a Channel on a Platform
uninstall(Platform, Channel):Boolean	Uninstalls a Channel from a Platform
notify(Platform, Channel_{old}, Channel_{new}):void	Notifies a Platform of a Channel update
setAdministrator(Platform, Actor):void	Sets the administrator of a Platform
addUser(Platform, Actor):void	Adds a Platform's User
removeUser(Platform, Actor):void	Removes a Platform's User
getConfigurationHW(Platform, HardwareDevice):HardwareConfiguration	Returns the configuration of a HardwareDevice
getConfigurationSW(Platform, Unit):SoftwareConfiguration	Returns the configuration of a Unit
configureHW(Platform, HardwareConfiguration):Boolean	Configures a HardwareDevice on a Platform
configureSW(Platform, SoftwareConfiguration):Boolean	Configures a Unit on a Platform
getAdministrator(Platform, Date):Actor	Get the Administrator of a Platform, based on a Date
getUsers(Platform, Date):$\mathbb{P}$Actor	Get the set of users of a Platform, based on a Date
getConfiguration(Platform, Date):PlatformConfiguration	Gets the configuration of a Platform, based on a Date
revertConfiguration(Platform, Date):Boolean	Reverts the configuration of a Platform, based on a Date
getUnits(Platform):$\mathbb{P}$Unit	Returns all Units installed on a Platform
getPlatforms(Unit):$\mathbb{P}$Platform	Returns all Platforms having installed a Unit

Table B.3: Platform Management Activity operations

Attribute Name	Attribute Type	Description
addresses	$\mathbb{P}$PlatformAddress	Addresses of the Platform
administrator	Actor	Admnistrator of the Platform
users	$\mathbb{P}$Actor	Users of the Platform
configuration	PlatformConfiguration	Configuration of the Platform

Table B.4: Platform Artifact Attributes

as for Channels build and dependency declaration. The attributes of the PlatformConfiguration Artifact are presented in Table B.5

Attribute Name	Attribute Type	Description
platform	Platform	Platform to which the configuration is associated
hardware	$\mathbb{P}$HardwareConfiguration	HardwareConfigurations of the Platform
software	$\mathbb{P}$SoftwareConfiguration	HardwareConfigurations of the Platform

Table B.5: PlatformConfiguration Artifact Attributes

The HardwareDevice Artifact represents a hardware component. It can be configured through a HardwareConfiguration Artifact. Hardware devices can be of any type indicated as a HardwareType Artifact. Possible types are for instance Ethernet and wifi cards, printers, sound cards, scanners, etc. Other available information include the vendor company having produced the device, the serial number, the model and a short description. This Artifact also provides information about possible configuation options, andproposes a standard configuration. All thes attributes are listed in Table B.6.

Attribute Name	Attribute Type	Description
description	Text	HardwareDevice description
vendor	Text	HardwareDevice vendor
serial	Text	HardwareDevice serial number
model	Text	HardwareDevice model
type	HardwareType	HardwareDevice type, i.e. “Printer”, “NetworkCard”, “Memory”, etc.
optionsConfig	Directory	Configuration options
standardConfig	Directory	Standard configuration

Table B.6: HardwareDevice Artifact Attributes

The HardwareConfiguration artifact defines how a HardwareDevice installed on a Platform is configured. For instance we can imagine a that the HardwareConfiguration of a Printer could define the print quality to be used and could indicate if the economy mode has to be used as well as its IP address if it is a network printer. Table B.7 lists the attributes of this Artifact.

Attribute Name	Attribute Type	Description
device	HardwareDevice	HardwareDevice being configured
config	Directory	Configuration

Table B.7: HardwareConfiguration Artifact Attributes

Similarly, the SoftwareConfiguration Artifact describes the way a Unit is configured. Available configuration options and the stadard configuration of the Unit are defined in the Unit Artifact. These options can includes for instance a list of ports to listen to, virtual hosts which are handled, etc. Table B.8 lists existing methods to retrieve attributes held by this artifact.

Attribute Name	Attribute Type	Description
unit	Unit	Unit being configured
config	Directory	Configuration

Table B.8: SoftwareConfiguration Artifact Attributes

Production Management. Production Management provides all operations for content creation. This Activity is a prerequisite for other Activities such as Testing, Defect Management and Distribution. It includes operations for the declaration of units of Content which are put into production, i.e. which have been created and are ready for further manipulation, their compilation and bundling with other units of content. Table B.9 lists the operations related to this Activity,

A Unit is the artifact representing content resources produced within a Project. These resources can be binary content, source code, documentation or utility content such as scripts. The set Unit represent the set of created Units. Each Unit is bound to a directory of attributes on its creation. This directory

Operation	Description
add(Unit):void	Adds an Unit into production
compile(Unit):void	Compiles an Unit which is already in production
createBundle(ℙ Unit):Bundle	Creates a Bundle from a set of Units
getUnits():ℙUnit	Returns all Units in production
getUnits(Bundle):ℙUnit	Returns all Units part of a Bundle
getBundles():ℙ Bundle	Returns all Bundles in production
getBundles(Bundle bundle):ℙ Bundle	Returns all Bundles part of a Bundle
getTimeToRebuildFromSource(ℙ Unit):Number	Returns the time needed to rebuild a Unit in seconds
getVariablesCount(ℙ Unit):Number	Returns the number of variables of the unit
getFunctionsCount(ℙUnit):Number	Returns the number of functions of the unit
getMacrosCount(ℙ Unit):Number	Returns the number of Macros of the Unit
getAverageSourceSize(ℙ Unit):Number	Returns the average source size of the unit

Table B.9: Production Management Activity operations

characterizes the Unit and defines mandatory Attributes that have to be provided in order to have a well formed Unit. The functionalities provided by an unit are described using functional attributes. They are also used for describing runtime and compile-time dependencies and conflicts between units. In particular, *Functionality* Artifacts are used to abstractly denote the so-called virtual packages in dependency management. Table B.10 lists the attributes of this Artifact.

A UnitLocation Artifact gives precise information about where a unit can be retrieved and how it is named. The artifact UnitLocation is independent from the distribution mechanism employed and provides means for verifying the integrity of retrieved F/OSS content. It can evolve over time to continuously provide accurate information on unit localization.

A Bundle is a special Unit representing a collection of Units. As such it possesses the same attributes a Unit does, and adds information about the Units it gathers as a collection. As other Units, Bundles possess their own content and also possess their own dependencies be they requirements or conflicts. This allows Bundles creators to specify collection related known issues and requirements. Table B.12 lists the attributes which are specific to the Bundle Artifact, attributes of the Unit Artifact are omitted.

Dependencies Management. A key feature of the EDOS-PRM is to provide a means for ensuring a complete control over F/OSS content dependencies. The Dependency Management Activity aims at achieving this goal by providing operations for verifying dependencies for Units and Bundles, providing operations for measuring them and for measuring potential dependency related issues which may appear. Finally, note that as dependencies are declared at Unit creation time, this Activity does not provide a means to declare new dependencies. Thus detecting new dependency issues should result in the creation of a new Unit Artifact. The operations related to this Activity are listed in Table B.13.

License Management. Every F/OSS content must respect usage policies defined by the license that is associated with it. Licenses have to define explicitly the context in which a content can be used, modified, copied or integrated in other creations. The License artifact presented in Table B.15 provides a means to describe Licenses and the License Management Activity provides the operations needed manipulate them. For instance, it allows to identify them precisely, retrieve their text, contributors, and known compatibility issues with other Licenses. These operations ar listed in Table B.14.

Test Management. The Test Management Activity provides all needed operations for handling tests. These operations allow users to declare tests for Units, run them and create reports afterward. Further,

Attribute Name	Attribute Type	Description
description	Text	Human readable description of the Unit
project	Project	Project in whose scope the Unit is produced
contentType	UnitType	Code source, binary, documentation, application or utility content. A Unit can be of multiple types.
status	UnitStatus	Maturity of the content: Alpha, Beta, RC
unitLocation	UnitLocation	The physical space where the Unit can be retrieved from
functionalities	$\mathbb{P}$Functionality	Functionalities of the Unit
license	License	License the Unit is bound to
version	UnitVersion	Version of the Unit
compatibility	$\mathbb{P}$Unit	Backward compatibility with other versions of the Unit
requiredSWInstalltime	Directory$_{UnitLocation}$	Sets of Software Units required at install time
requiredSWRuntime	Directory$_{UnitLocation}$	Sets of Software Units required at runtime
conflictSWInstalltime	Directory$_{UnitLocation}$	Sets of Software Units conflicting at install time
conflictSWRuntime	Directory$_{UnitLocation}$	Sets of Software Units conflicting at runtime
replaces	$\mathbb{P}$UnitLocation	Set of Units the Unit replaces once installed
requiredHWInstalltime	Directory$_{HardwareConfiguration}$	Sets of HardwareConfigurations required at install time
requiredHWRuntime	Directory$_{HardwareConfiguration}$	Sets of HardwareConfigurations required at runtime
conflictHWInstalltime	Directory$_{HardwareConfiguration}$	Sets of HardwareConfigurations conflicting at install time
conflictHWRuntime	Directory$_{HardwareConfiguration}$	Sets of HardwareConfigurations conflicting at runtime
optionsConfig	Directory	Unit's configuration options
standardConfig	Directory	Unit's standard configuration

Table B.10: Unit Artifact Attributes

Attribute Name	Attribute Type	Description
Unit	Unit	Unit the UnitLocation refers to
Size	Number	Size of the content
CRC	Text	CRC to check the integrity of the content
Locations	$\mathbb{P}$URL	The different locations where a Unit can be retrieved from

Table B.11: UnitLocation Artifact Attributes

Attribute Name	Attribute Type	Description
Content	$\mathbb{P}$Unit	Content of the Bundle

Table B.12: Bundle Artifact Attributes

this Activity provides a means for retrieving any information related to the testing activity as well as any information required for measuring it. Table B.16 lists the operations it provides.

This Activity handles Testable and Test Artifacts. A Testable is an Artifact that can be tested. It provides information about existing Tests that can be run and methods to run them. Known Testable

Operation	Description
check(Unit unit):Boolean	Checks Unit's dependencies
getDependencies(Unit unit):ℙUnit	Extracts all Units an Unit is depending on
getTimeToComputeDependencies(Unit unit):Number	Computation time in seconds
getTimeToCreateMinimalClosedPackageSet(Unit[] requiredUnits):Number	Time to create minimal closed package set in seconds
getTimeToEvaluateMonotonicity(Unit unit):Number	Time to evaluate monotonicity in seconds
getTimeToEvaluateTrimmed(Unit[] units):Number	Time to evaluate trim in seconds
getTimeToEvaluateUpgradability(Unit[] units):Number	Time to evaluate upgradability in seconds
getUnmetDependenciesCount(Unit unit):Number	Number of unmet dependencies
getReplaceTypeInDegreeDependenciesCount(Unit unit, Unit[] units):Number	Number of replace in degree dependencies
getReplaceTypeOutDegreeDependenciesCount(Unit unit, Unit[] units):Number	Number of replace out degree dependencies
getRunTypeInDegreeDependenciesCount(Unit unit, Unit[] units):Number	Number of run in degree dependencies
getRunTypeOutDegreeDependenciesCount(Unit unit, Unit[] units):Number	Number of run out degree dependencies
getInstallTypeInDegreeDependenciesCount(Unit unit, Unit[] units):Number	Number of install in degree dependencies
getInstallTypeOutDegreeDependenciesCount(Unit unit):Number	Number of install out degree dependencies
getSATTemperature(Unit[] unit):Number	SAT Temperature
getDependencyKernelSize(Unit[] unit):Number	Size of kernel dependencies
getDependencyCount(Unit[] unit):Number	Number of dependencies

Table B.13: Dependencies Management Activity operations

Operation	Description
getLicenseName():Text	Returns the License name
getLicenseAcronym():Text	Returns the License acronym
getLicenseAbstract():Text	Returns a short description of the License
getLicenseVersion():UnitVersion	Returns the version of the License
getPreviousVersion():License	returns previous version of the License
getLicenseDate():Date	Returns the release date of the License
getLicenseUrl():URL	Returns the URL where the full text of the License can be found
getLicense():License	Returns the License under which the License is released
getLicenseAuthors():ℙ Actor	Returns the Actors having created the License
compatibleWithLicense():ℙℙ License	Returns the sets of Licenses with which the License is compatible
incompatibleWithLicense():ℙℙ License	Returns the sets of Licenses with which the License is incompatible

Table B.14: License Management Activity operations

Attribute Name	Attribute Type	Description
Name	Text	Name of the License
Acronym	Text	License Acronym
Abstract	Text	License human readable description
Version	UnitVersion	License version number
Previous	License	Previous version of the License
Location	URL	URL where the full text of the license can be found
Authors	$\mathbb{P}$Actor	Actors of the License
Date	Date	License release Date
License	License	License the License is bound to
Compatibility	$\mathbb{P}\mathbb{P}$License	Compatible Licenses
Incompatibility	$\mathbb{P}\mathbb{P}$License	Incompatible Licenses

Table B.15: License Artifact Attributes

Artifacts include Units and Tests which can be both tested. The Test Artifact gathers information about how an existing Artifact has to be verified for Defects. Tests are then applied to Testable Artifacts. Tests can be made of other Tests in order to build Test suites. Once a Test has been run, a TestReport Artifact is produced. The TestReport Artifact represents the report which is produced each time a Test is run. It provides information about the execution result and context of the Test, and indicates if any problem has been detected. Table B.17 lists Testable Attributes, while Test Attributes are listed in Table B.19 and TestReport Attributes are listed in Table B.18.

Defect Management. The Defect Management Activity provides operations needed to efficiently handle defects. This Activity gathers the different Defects which may exist for Testable content and keeps an history of changes made to these Defects. It provides a means to declare Defects, search for them, change their status, get information about them, add PlatformConfigurations affected by the Defect, propose and choose Patches, but also to apply these Patches on a given Platform. It also provides a lookup feature which allows to search for Defects sharing similarities. Every Defect status changes, Patch addition or change of the set of affected PlatformConfigurations by a Defect is historized. Table B.20 provides the list of this Activity's operations.

Defect Artifacts are used to indicate a malfunction or bug in a Testable artifact that has been detected by a community member. This artifact holds different attributes providing information such as the Testable targeted by it, available Patches which can be used to remove the Defect, the status of the Defect. The status of the Defect is represented as a DefectStatus Artifact. Table B.21 lists these Attributes. Defect can be fixed using Patches. A Patch is a special Unit whose installation on a Platform has to fix a Defect. A Patch is a Unit and thus can require other Units to be installed. Patch dependency information is thus used to find Patches that need to be installed before another Patch is installed. Table B.22 lists the Attributes of the Patch Artifact, omitting the ones inherited from the Unit Artifact.

Distribution Management. Once Units have been produced, tested and debugged they can be distributed. The purpose of the Distribution Management Activity is to make Units available to the community and remove them from distribution channels if needed. Corresponding operations are listed in Table B.23.

Operation	Description
declareTestable(Project, Testable):void	Declares a Testable for a Project
addTest(Test, Testable):void	Adds a Test for a Testable to its list of Tests
addTest(Test, Testable$_{added}$, Test$_{previous}$):void	Adds a Test for a Testable after a given Test
test(Testable):void	Launches all Tests for a Testable using the chain order
test(Testable, Test):void	Runs a Test toward a Testable
getTests():ℙTest	Returns all existing Tests
getTests(Project):ℙTest	Returns all existing Tests
getTests(Testable testable):ℙTest	Returns all Tests for a Testable classified by Test order
getTests(PlatformConfiguration):ℙTest	Returns all Tests that can be run on a given Configuration
getTests(Actor):ℙTest	Returns all Tests created by a given Actor
getTestablesUnderTest():ℙTestable	Returns all Testables having at least a Test associated to them and having been Tested
getTestablesNotTested():ℙTestable	Returns all Testables having at least a Test associated to them and having never been Tested
getTestables():ℙTestable	Returns all Testables having at least a Test associated to them
getTestables(Project):ℙTestable	Returns all Testables of a Project having at least a Test associated to them
getRuns(Test):Number	Returns the number of times a given Test has been run.
isValid(Test):Boolean	Returns the validity of the test. Allows to declare a Test as not valid.
setValid(Test, Boolean$_{validity}$):void	Sets the validity of the Test.
addReport(TestReport, Test):void	Adds a TestReport for a given Test.
getReports(Test):ℙTestReport	Returns all TestReports issued from a given Test

Table B.16: Test Management Activity operations

Attribute Name	Attribute Type	Description
Tests	ℙTest	The set of Tests applicable to a Testable Artifact.

Table B.17: Testable Artifact Attributes

Attribute Name	Attribute Type	Description
Name	Text	Name of the Test
Description	Text	Brief description of the Test
Target	Testable	Target Testable to be tested
Configuration	PlatformConfiguration	PlatformConfiguration a Test is designed for
Functionalities	ℙFunctionality	Set of Functionalities a Test is testing
Type	TestType	Type of the test
Author	Actor	Actor having created the Test
Content	ℙURL	URL where the content of the Test can be retrieved from
Subtests	ℙTest	Other Tests a Test embeds (test suite case)

Table B.18: Test Artifact Attributes

Attribute Name	Attribute Type	Description
Author	Actor	Actor having generated the TestReport
Target	Test	Test to which the report is bound
Description	Text	Content of the TestReport
Configuration	PlatformConfiguration	Exact PlatformConfiguration on which the Test has been run
Defects	$\mathbb{P}$Defect	The Defects having been detected if any

Table B.19: TestReport Artifact Attributes

Operation	Description
declareDefect(Testable, Defect):void	Declares a Defect for a Testable element.
getDefectStatus(Defect):DefectStatus	Gets the DefectStatus of a Defect object.
proposePatch(Defect, Patch, PlatformConfiguration):void	Proposes a Patch for a Defect and for a PlatformConfiguration.
getProposedPatches(Defect):void	Proposes a Patch for a Defect.
getProposedPatches(Defect, PlatformConfiguration):void	Proposes a Patch for a Defect and for a PlatformConfiguration.
changeDefectStatus(Defect, DefectStatus):void	Changes the DefectStatus of a Defect.
changeDefectStatus(Defect, DefectStatus, Patch):void	Changes the DefectStatus of a Defect proposing a Patch.
addAffectedConfiguration(Defect, PlatformConfiguration):void	Adds a Configuration that leads to the Defect to be reproduced.
setAsSibling(Defect$_{main}$, Defect$_{sibling}$):void	Declares a Defect as the sibling of another Defect
getMainDefect(Defect):Defect	Returns the main Defect of a sibling Defect
getSiblingDefects(Defect):$\mathbb{P}$Defect	Returns sibling Defects of a Defect
getPatches(Defect):$\mathbb{P}$Patch	Retrieves all Patches for Defect.
getHistory(Defect):$\mathbb{P}$Defect	Returns a list of Defects representing the history of the provided Defect
getDefects():$\mathbb{P}$Defect	Returns all existing Defects.
getDefects(Testable):$\mathbb{P}$Defect	Returns all existing Defects for a given Testable element.
find(Directory$_{template}$):$\mathbb{P}$Defect	Returns all existing Defects corresponding to specific criteria.
apply(Platform, Patch):void	Applies a Patch, correcting the issue of the target Unit and setting the new Unit ID.
isApplied(Platform, Patch):Boolean	Verifies if a Patch has been applied.
revert(Platform, Patch):Boolean	Removes a Patch from a Platform and restore previous state.

Table B.20: Defect Management Activity operations

Attribute Name	Attribute Type	Description
targets	Testable	Get the Testable the Defect affects
description	Text	Description of the Defect
patches	$\mathbb{P}$Patch	Available Patches to fix a Defect
configuration	PlatformConfiguration	Platform configuration the Defect has been detected on
status	DefectStatus	Defect status, depending on the life cycle of the defect

Table B.21: Defect Artifact Attributes

Attribute Name	Attribute Type	Description
target	Unit	the target Unit to be patched.

Table B.22: Patch Artifact Attributes

Operation	Description
distribute(Unit):void	Distributes an Unit
remove(Unit):void	Removes an Unit from Distribution
isDistributed(Unit):Boolean	Verifies if a Unit is distributed
getUnits():$\mathbb{P}$Unit	Returns all distributed Units
getBundles():$\mathbb{P}$Bundle	Returns all distributed Bundles
getUnits(Directory$_{template}$):$\mathbb{P}$Unit	Returns distributed Units matching specified characteristics.
getBundles(Directory$_{template}$):$\mathbb{P}$Bundle	Returns distributed Bundles matching specified characteristics.

Table B.23: Distribution Management Activity operations

Appendix C

EDOS-PRM: benefits

This Appendix presents some benefits the PRM brought to the EDOS project. We focus here on some fine grained examples of improvements the PRM can bring to F/OSS process, be it on the information control level in Section C.1, information retrieval level in Section C.2, information enrichment level in Section C.3

C.1 Information Control

The PRM model offers multiple means to achieve this goal respect to integrity enforcement, information substitutability. We explore here some examples involving these two capabilities of the PRM.

C.1.1 Integrity enforcement

As previously defined, a goal of the PRM is to ensure F/OSS information integrity by expressing integrity rules and enforcing them. Such rules are defined for three different levels and are accordingly applied.

The first set of rules ensures that created Artifacts are well formed. A good example of an integrity rule which has to be enforced at Artifact instantiation time, is the dependency control which has to be done when creating a new content Unit. Indeed when creating a new Unit no internal dependency conflicts are allowed. This includes the following rules, where points 2 to 4 have to be applied recursively to all dependencies in order to ensure that there will be no dependency collision in the whole Unit dependency chain:

1. No conflicts between the different hardware and software requirement dependencies of the Unit and its hardware and software conflict dependencies are allowed
2. No conflicts between the different hardware and software requirement dependencies of the Unit and the hardware and software conflict dependencies of the Units it is depending on are allowed
3. No conflicts between the different hardware and software conflict dependencies of the Unit and the hardware and software requirement dependencies of the Units it is depending on are allowed
4. The License of the created Unit cannot conflict with the License of any of the Units on which the Unit depends.

The second level integrity rules ensure that the internal state of Artifacts remain well formed through manipulations. While some attributes are immutable, some other can be modified over time. Artifact manipulation integrity rules cover thus all the rules ensuring that a modification of artifacts attributes won't tamper its integrity. For instance, when declaring a conflict with some functionality, one must be

sure that this conflicting functionality is not provided by any of the Units the modified Unit is depending on. Finally, Process interaction integrity rules ensure that transversal integrity is also enforced. These rules are focused on Artifacts manipulation in the context of F/OSS processes and ensure that activities cannot put the PRM in an unstable state. For instance, the PRM declares that the following three rules have to be enforced when calling the $distribute$ method of the Distribution Management process interface:

1. Only Units which are officially in production (registered as part of a project) can be distributed.
2. Minimal tests which have been defined for the Unit to be distributed must all have been passed successfully
3. Only Units whose software dependencies can be satisfied (i.e. which can be retrieved from the environment) can be distributed.

C.1.2 Substitutability

Another interesting aspect of the PRM is that it provides information about Artifacts substitutability. The PRM defines substitutability rules for each type of Artifact. It considers both straightforward information, such as Artifacts attributes matching, as well as transversal information, related to the use of the Artifacts within the F/OSS Process. We present here four examples illustrating how substitutability is defined and how its use can benefit the F/OSS Process.

Unit substitutability When a Unit which has to be installed is not available, or if one of its dependencies conflicts with other installed Units, finding a replacement Unit that will seamlessly provide the same features is mandatory. Unit substitutability depends on many criteria. Thus, a Unit u_1 to be substitutable by a Unit u_2 the following rules have to be respected:

1. u_1 functionalities must be substitutable by u_2 functionalities.
2. u_1 license must be substitutable by u_2 license

Indeed, both Units have to do the same job, and in a F/OSS context, comply with compatible Licenses, Further some rules depending on the Unit usage context have to be respected in order to verify the substitutability of u_1 by u_2. If u_1 is used in the context of a set of Installed Units $U_{context}$ hardware and software issues appear. Thus the following rules have to be added:

1. u_2 or its software and hardware required dependencies cannot be part of the software and hardware conflict dependencies of any Unit of $U_{context}$ set of Units
2. u_2 software and hardware conflict dependencies cannot include any of the Units belonging to the $U_{context}$ set of Units or their software
3. u_2 License cannot conflict with the License of any of the Units belonging to the $U_{context}$ set of Units which are depending on u_2 or on which u_2 depends.

Platform substitutability When searching for a Platform having a particular set of properties in order to run some tests, or to deploy an application, minimizing the effort and related risk is wished.

Platform substitutability is then useful, it ensures that all Units that can be installed on a platform p_1 can be also installed on p_2. For this reason, the Configuration of Platforms has to be verified for substitutability and not only matched. Indeed, matching the configuration of p_2 to p_1's Configuration, would allow p_2's Configuration to be broader than p_1's. The resulting issue being that other hardware

devices or Units available on p_2 can create potential conflicts with Units at installation time on p_2 while creating none when installed on p_1.

Thus, a Platforms p_1 can be substituted by a platform p_2 if the Configuration of p_1 is substitutable by p_2's Configuration and if p_2 hosts the Units p_1 is hosting. This ensures that the two Platforms are providing the same hardware and software features.

Patch substitutability Patch substitutability depends on the target Unit to be Patched and the Defects corrected by the Patches. As the latter do not change attributes associated to an Unit, the only difference between the application to an Unit u of two different Patches $p1$ and $p2$ which target the same defect d, is the newly created Unit representing the result of the application of $p1$, $p2$ to u. Thus as long as $p1$ and $p1$ are targeting the same unit u and if $p2$ corrects at least the Defects $p1$ corrects then $p1$ can be substituted by $p2$. However, if any installed Unit depends on the resulting Unit from the application of $p1$, this dependency has to be updated.

License substitutability As the description of License attributes declared it, Licenses are related to each other through a compatibility relation. This compatibility relation is used to define the substitutability of Licenses. To substitute a License l_1, another License l_2 has to be compatible with l_1, but this is not sufficient. Indeed, as Units using different Licenses can be combined, depending on the content of the licenses, situation may occur where if l_1 is used along with a set of given Licenses, l_2 becomes incompatible with l_1. We define thus, that a License l_2 can substitute a License l_1 if the following rules are respected:

1. l_2 is declared compatible with l_1
2. l_1 is not declared incompatible with l_2
3. l_2 is declared compatible with the set of Licenses l_1 is part of.
4. l_1 is not declared incompatible with the set of Licenses l_2 is part of.

C.2 Information Retrieval

Another key mission for F/OSS Information Systems is to ease information retrieval. To this extend, all PRM Activities provide different operations giving access to the Artifacts they are handling. The lookup operation has to be provided by all Activities. Its goal is to enable the search for Artifacts depending on user defined criteria. The latter are provided as an Actor attribute directory d. Thus d contains fields $d.\_i$ corresponding to the different attributes which can be retrieved using attr_i query methods. When calling the lookup method with d as parameter, for each Artifact a an Activity A is responsible for, the lookup method verifies if $a.\mathsf{attr}_i$ matches $d.\_i$. Then the method returns a set of Artifacts verifying this rule.

Examples of information retrieval means and more precisely of the lookup operation can be given to illustrate their support for Role, Task, Project and Community Management. These means can help Project managers find contributors and assign them roles, or help potential contributors find projects matching their interests and where their skills can be useful.

Actor Lookup The lookup operation provided by the Actor Management Activitiy allows PRM users to search for Actors based on any Actor related Attribute. For instance, $d.interests$ of the directory d passed as parameter can contain a set of Topics to indicate that searched Actors must be interested in the specified Topics. Thus, in such a case, for each Actor a of the set of Actors returned by the

lookup operation, m(attr$_{interests}(a)$,$d.interests$). Any other attribute can be included in the search such as Actor's competences, knowledge or even affiliation in order to get a hand on the Actors best matching needs Project managers may have.

Project lookup Similarly, Projects can be searched for instance based on the Topics they are working on. In this case, looking-up Projects with a directory d containing $d.topics$, the set of topics an Actor a is interested in, returns a set of Projects, where each Project p matches the request such as m(attr$_{topics}(p)$,$d.topics$).

Contributor Lookup Further, the PRM offers means to retrieve other information which may be usefull in order to support F/OSS Process management. Such information includes for instance, a list of contributors to a each Project which can be retrieved using the getContributors operation of the Project Management Activity and providing a Project as parameter.

Role Lookup Then, for each of these Actors, assigned Roles in the scope of the Project can be known using the Role Management Activity getRoles operation, providing the Actor and the Project. Similarly, all Actors being assigned a Role for a Project can be retrieved using the getContributors operation of the same Activity providing a Role and Project as parameter. These operations enable thus project managers to know exactly and easily what each contributor is doing in the scope of a Project.

Task Lookup In order to ease the work of F/OSS project contributors, there should be a way to retrieve all the tasks a contributor is in charge of. The PRM Activity operations getTasks enable Task retrieval for Actors in the scope of a Project and also more precisely for a specific Role the Actor has been assigned.

C.3 Information Enrichment

The PRM can be applied to multiple F/OSS domains in order to cover the specific needs they have. The PRM helps enriching the information in order to give it more expressiveness. We explore in this section how this feature has been exploited in the scope of the EDOS project which is using the PRM: the differentiation between patching and versioning and advanced software dependency management based on software and hardware features.

C.3.1 Patching and versioning

The PRM uses the notion of Unit artifact to represent installable software content resources such as packages. Units are described using multiple attributes, indicating for instance the functionalities provided by the Unit, its version or patching information. The PRM makes indeed a distinction between versions and patches.

While new versions of a unit denote major changes made to its predecessor such as design change for the content, dependencies changes, addition or removal of functionality, or any other behavioral change to the unit, patches are used to fix issues (which are denoted by defect artifacts) in specific units. All units are uniquely identified by a localization attribute indicating all possible places where they can be retrieved from. Patches are designed for a single unit, and their application results in the creation of a new unit with a new identifier. Two versions of a unit have thus two different identifiers and so does a unit to which two different patches have been applied.

This distinction allows to clarify version and patching trees, thus helping managing dependencies. Indeed, a dependency may require a specific version with a patch applied to it. As patched units keep track of all previously applied patches for their version it is always possible to know if such a dependency

is available. Further, as patches provide all information for being rollback-ed, it is always possible to revert to a previous state and apply another patch. Thus the PRM provides both expressiveness and flexibility for versioning an patching.

C.3.2 Advanced dependencies declaration

Unlike usual dependency declaration which are expressed as requirements for other content resources or conflicts with other content resources, the PRM provides multiple ways to express both software and hardware dependencies as well as conflicts. As such it follows the recommendations specified in [156].

The attribute based approach to describe attribute functionality adds expressive power to dependency description. Attributes and attribute expressions can be used to express generic properties of units. Then the PRM allows units to be matched to such sets of attributes. This is particularly useful to replace so called *virtual packages* as the latter are supposed to share a common set of properties. Further, as attributes define the properties of unit, and as attributes are substitutable, dependencies built on attributes ensure that as long as a unit offers the same attributes an older version was offering, the newer version can substitute the older one, if there is no dependency conflict with the local configuration.

As for patching and versioning, the PRM separates semantically different information to describe unit's properties, and thus unit's dependencies. Indeed, different attribute sets are provided to express different information. The elements of these sets are then separated depending on their usage context. Thus the PRM acts as a safe-guard ensuring a correct definition of dependencies.

Software Dependencies. Software dependency information is expressed as Unit's attributes. As mentioned in [156], there are three main classes of software dependencies.

The *required* dependencies of a unit can be described in terms of localization attributes of other units or in terms of functionality attributes. For instance, a unit can depend on a specific library provided by another unit, but can also depend on a particular functionality such as a SQL compatible database. In the latter case, the dependency is independent of any particular unit (package); this is referred to as a virtual package in [156]. Required dependencies are separated in two further subgroups, the first one indicating required dependencies at compilation time, and the second one describing required dependencies at execution time. As content may be installed without being run (as this is the case with latex files that can be compiled to dvi and then pdf and then distributed without ever opening the dvi file with a viewer), this separation is needed.

The second class of dependency information relates to installation and runtime *conflicts*. These can also be expressed in terms of localization attributes or in terms of functional attributes to be avoided in order to have a successful installation and execution. This implies that none of the listed units can be on the target computer at installation time, respectively at run time, and that none of the installed units should provide one of the specified functional attributes.

Finally, the third class of dependencies enumerates the units that have to be *replaced* after having installed the unit. This dependency is expressed in terms of localization attributes of the units that have to be replaced.

Hardware Dependencies. Similarly to software dependencies, hardware dependencies can be separated in two subgroups indicating hardware devices *required* for the Unit to work and hardware devices *conflicting* with the Unit's execution. This information is kept as sets of HardwareConfiguration artifacts which provide configuration related information for hardware devices. These can indicate for instance, the CPU type, the link type of a net connection, the print quality of a printer, etc or other information such as the IP address of an Ethernet card.

The functionalities provided by an unit are described using functional attributes. They are also used for describing runtime and compile-time dependencies and conflicts between units. In particular,

elements from $\mathcal{A}_{functionalities}$ are used to abstractly denote the so-called *virtual packages* in dependency management.

All dependencies are gathered using Directories. This allows to declare, through artifact expressions, alternatives for dependencies, be they hardware or software dependencies and unit or feature based dependencies or conflicts. For instance, it is possible to declare complex dependencies such as "Unit u_2 is depending on Unit u_1 having version comprised between v2 and v4 (preferred dependency), or any other Unit providing functionalities expressed by the set $s_{dependency}$ but not the set $s_{conflict}$ (alternative)". This approach provides flexibility for dependency and conflicts declaration, and allows their precise declaration in terms of units, functionalities and hardware.

Bibliography

[1] Serge Abiteboul, Roberto Di Cosmo, Stefane Fermigier, Stephane Lauriere, and al. Edos: Environment for the development and distribution of open source software. In *Proceedings of the First International Conference on Open Source Systems, Genova, Italy*, July 2005. 14

[2] Alioth. http://alioth.debian.org/, June 2006. 44

[3] Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. Gregorio robles, stefan koch and jesus m. gonzalez-barahona. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04). 26th International Conference on Software Engineering (Edinburgh, Scotland)*, May 2004. 40

[4] Apache agila. http://wiki.apache.org/agila/, January 2007. 32

[5] Apache ant: build tool. http://ant.apache.org/, October 2006. 32

[6] Apache Project. Apache maven: software project management and comprehension tool. http://maven.apache.org/, October 2006. 33

[7] Apache Project. Apache open for business (OFBiz). http://ofbiz.apache.org/, October 2006. 15, 32

[8] Apache Project. Jakarta turbine project. http://jakarta.apache.org/turbine/, January 2007. 33

[9] Microsoftand SAP BEA, IBM. Business process execution language for web services version 1.1. Technical report, IBM, May 2003. 31

[10] Objectweb bonita. http://wiki.bonita.objectweb.org/xwiki/bin/view/Main/WebHome, January 2007. 32

[11] Business process management initiative (BPMI). http://www.bpmi.org/, December 2006. 30

[12] Business process management language 1.0 (BPML). http://www.ebpml.org/bpml_1_0_june_02.htm, January 2007. 30

[13] Business process modeling notation (BPMN). http://www.bpmn.org/, January 2007. 32

[14] Business readiness rating - a framework for evaluating open source software. http://www.openbrr.org/wiki/index.php/Home, January 2007. 39

[15] Bugzilla. http://www.bugzilla.org/, June 2006. 35, 44

[16] Bugzilla test runner. http://www.willowriver.net/products/testrunner.php, June 2006. 44

[17] Caixa magica linux. http://www.caixamagica.pt/, October 2006. 4

[18] Co-ordination action for libre software engineering for open development platforms for software and services (CALIBRE) project. http://www.calibre.ie/, September 2006. 14, 41

[19] M. Carro. The amos project: An approach to reusing open source code. In *Proceedings of the CBD 2002 / ITCLS 2002 CoLogNet Joint Workshop*, pages 59–70. Facultad de Informatica, September 2002. 33

[20] Connected limited device configuration (CLDC). http://java.sun.com/products/cldc/, December 2006. 117

[21] Compiere. http://www.compiere.org/product/index.html, October 2006. 15

[22] Cospa project. http://www.cospa-project.org/, September 2006. 14

[23] Common public license (CPL) - v1.0. http://www.eclipse.org/legal/cplv10.html, September 2006. 14

[24] Concurrent versions system (CVS). http://www.nongnu.org/cvs/, October 2006. 26

[25] B. Dale and H. Bunney. *Total Quality Management Blueprint*. Blackwell Publishing, Incorporated, Oxford, UK, 1999. 34

[26] Debian Project. Debian developer locations. http://www.debian.org/devel/developers.loc, October 2006. 24

[27] Debian Project. Debian linux. http://www.debian.org/, October 2006. 6

[28] Debian Project. Debian new maintainer's guide. http://www.debian.org/doc/maint-guide/, October 2006. 26

[29] Debian Project. Debian social contract. http://www.debian.org/social_contract, October 2006. 25

[30] Nicolas Jullien Didier Demazière, François Horn. How free software developers work: The mobilization of distant communities. Technical report, CNRS,CLERSE, MARSOUIN, October 2006. 28

[31] Description of a project (DOAP). http://usefulinc.com/doap/, January 2007. 43

[32] Dogtail. http://people.redhat.com/zcerza/dogtail/, August 2007. 37

[33] Wayne W. Eckerson. Deploying dashboards and scorecards. Best practices report, The Data Warehousing Institute (TDWI), July 2006. 41, 42

[34] Eclipse project. http://www.eclipse.org/, September 2006. 1, 14, 26

[35] Eclipse Project. Eclipse project dashboards. http://www.eclipse.org/projects/dashboard/, June 2006. 44

[36] Environment for the development and distribution of open source software(EDOS) project. http://www.edos-project.org, June 2006. 37, 39

[37] EDOS Project. Edos portal for caixa magica linux. http://caixamagica.edos-project.org/xwiki/, January 2007. 39

[38] EDOS Project. Edos portal for debian linux. http://debian.edos-project.org/xwiki/, January 2007. 39

[39] EDOS Project. Edos portal for mandriva linux. http://mandriva.edos-project.org/xwiki/, January 2007. 39

[40] EDOS WP2 Team. Edos dependency management wp overview. http://www.edos-project.org/xwiki/bin/view/Main/Wp2Overview, 2005. 21

[41] EDOS WP3 Team. Deliverable d3.2.2: Second version of the qsd portal. Technical report, EDOS Project, 2006. 37

[42] EDOS WP5 Team. Deliverable d5.1.: Edos metrics definition. Technical report, EDOS Project, 2005. 26, 39, 106

[43] EDOS WP5 Team. Deliverable d5.2.: Results of the measurements applied to the current processes for cooker and caixa magica. Technical report, EDOS Project, 2006. 39

[44] EDOS WP5 Team. Deliverable d5.3.: First run of the edos metrics applied to the new processes. Technical report, EDOS Project, 2006. 39

[45] An empirical approach to software archeology. Gregorio robles, jesus m. gonzalez-barahona and israel herraiz. In *21st IEEE International Conference On Software Maintenance*, September 2005. 40

[46] J. Erenkrantz and R. Taylor. Supporting distributed and decentralized projects: Drawing lessons-from the open source community. In *Proceedings of 1st Workshop on Open Source in an Industrial Context, Anaheim, California.*, October 2003. 29

[47] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, 2005. 113

[48] Exim internet mailer. http://www.exim.org/, December 2006. 21

[49] Gilles Falquet, Michel Leonard, and Jeanne Sindayamaze. F2concept: a Database Extension Language for Managing Class Extensions and Intensions. Final report, University of Geneva, May 1993. 131

[50] Alain Fernandez. *Les nouveaux tableaux de bord des managers*. Edition d'Organisation, Paris, 2005. 42

[51] Stephen Few. *Information Dashboard Design: the effective visual communication of data*. O'Reilly, Sebastopol, CA, 2006. 41

[52] Fitnesse. http://www.fitnesse.org, June 2006. 35, 44

[53] Formalized linux knowledge (Flink). http://flink.dcc.ufba.br/en/, June 2006. 43

[54] Floss project. http://www.infonomics.nl/FLOSS, September 2006. 14

[55] FLOSS Project. Floss reports. http://www.infonomics.nl/FLOSS/report, September 2006. 14

[56] FLOSSMetrics. Free/Libre Open Source Software Metrics (FLOSSMetrics) Project. http://www.flossmetrics.org/, January 2007. 41

[57] FLOSSMole. Free/Libre Open Source Software Mole (FLOSSMole). http://ossmole.sourceforge.net/, February 2007. 40

[58] Friend of a friend (FOAF). http://www.foaf-project.org/, January 2007. 43

[59] Chris Widdows Frans-Willem Duijnhouwer. Open source maturity model (OMMM). Technical report, Capgemini, 2003. 39

[60] Free Software Foundation. Gnu is not unix (GNU) operating system. http://www.gnu.org/. 11

[61] Free Software Foundation. Overview of the gnu project. http://www.gnu.ai.mit.edu/gnu/gnu-history.html, December 1998. 11

[62] Free Software Foundation. The free software definition. http://www.gnu.org/philosophy/free-sw.html, October 2006. 9

[63] Free software foundation (FSF). http://www.fsf.org/, October 2006. 10, 11

[64] Free Software Foundation. Free software philosophy. http://www.gnu.org/philosophy/, October 2006. 9

[65] Free Software Foundation. Free software: Various licenses and comments about them. http://www.gnu.org/licenses/license-list.html, October 2006. 10, 20

[66] Free Software Foundation. Gnu general public license. http://www.gnu.org/copyleft/gpl.html, October 2006. 20

[67] Freebsd project. http://www.freebsd.org/, September 2006. 11, 24

[68] FreeBSD Project. Freebsd dashboards. http://people.freebsd.org/~bsd/prstats/, June 2006. 44

[69] freshmeat.net. http://www.freshmeat.net/, October 2006. 24

[70] Freshports - the place for ports. http://www.freshports.org/, October 2006. 24

[71] Gaim linux/unix instant messenger client. http://gaim.sourceforge.net/, October 2006. 1, 26

[72] Gforge. http://gforge.org/, June 2006. 4, 24, 44

[73] Debora Abdalla Guillaume Barreau. Semantic linux: a fertile ground for the semantic web, April 2005. 43

[74] Idabc open source observatory. http://europa.eu.int/idabc/en/chapter/452/, September 2006. 14

[75] Interoperability research for networked enterprises applications and software (INTEROP) network of excellence. http://www.interop-noe.org/, January 2007. 42

[76] ISO. 8402:1994- quality management and quality assurance – vocabulary. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20115&showrevision=y, December 2000. 34

[77] ISO/IEC. 9646-7:1995 - information technology – open systems interconnection – conformance testing methodology and framework – part 7: Implementation conformance statements. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=3084, June 2001. 36

[78] ISO/IEC. Iso/iec 9126-2:2003 - software engineering – product quality – part 2: External metrics. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=22750, July 2003. 38

[79] ISO/IEC. Iso/iec 9126-3:2003 - software engineering – product quality – part 3: Internal metrics. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=22891, July 2003. 38

[80] ISO/IEC. 15504-1:2004 - information technology – process assessment – part 1: Concepts and vocabulary. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=38932&ICS1=35&ICS2=80&ICS3=, November 2004. 30

[81] ISO/IEC. 15504-4:2004 - information technology – process assessment – part 4: Guidance on use for process improvement and process capability determination. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=37462, July 2004. 30

[82] ISO/IEC. Iso/iec 9126-4:2004 - software engineering – product quality – part 4: Quality in use metrics. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=39752, March 2004. 38

[83] ISO/IEC. Iso/iec 25000:2005 - software engineering – software product quality requirements and evaluation (square) – guide to square. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35683, July 2005. 38

[84] ISO/IEC. Iso/iec 9126-1:2001 - software engineering – product quality – part 1: Quality model. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=22749&ICS1=35&ICS2=80&ICS3, September 2006. 38

[85] ISO/IEC. Iso/iec 15939:2002 - software engineering – software measurement process. http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=29572, July 2007. 38

[86] ISO/IEC. Iso/iec 25020:2007 - software engineering – software product quality requirements and evaluation (square) – measurement reference model and guide. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35744&scopelist=PROGRAMME, May 2007. 38

[87] ISO/IEC. Iso/iec prf tr 25021- software engineering – software product quality requirements and evaluation (square) – quality measure elements. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35745&scopelist=PROGRAMME, July 2007. 38

[88] D. Reed J. Oikarinen. Ietf rfc 1459: Internet relay chat protocol (irc). http://www.ietf.org/rfc/rfc1459.txt, May 1993. 24

[89] Kevin Crowston James Howison, Megan Conklin. Ossmole: A collaborative repository for floss research data and analyses. In *Proceedings of the First International Conference on Open SourceSystems, Genova, Italy*, July 2005. 40

[90] Java specification requests. http://jcp.org/en/jsr/, December 2006. 117

[91] Jboss jbpm. http://www.jboss.com/products/jbpm/, January 2007. 32

[92] Michel Pawlak Jean-Henry Morin. A credential based approach to managing exceptions in digital rights management systems. In *4th pre-ICIS Academic Workshop AIM, ICIS 2005 - International Conference on Information Systems*, Las Vegas, USA, Dec 2005. 129

[93] Michel Pawlak Jean-Henry Morin. In F. Herrmann and D. Khadraoui, editors, *Advances in Enterprise IT Security*, Information Science Reference, chapter From digital rights management to enterprise rights and policy management: Challenges and opportunities., pages ?–? IDEA Group Publishing, Mar 2007. 129

[94] Nicolas Jullien Jean-Michel Dalle. Open-source vs. proprietary software. January 2002. 34

[95] C. Jensen and W. Scacchi. Automating the discovery and modeling of open source software development process. In *3rd. Workshop on Open Source Software Engineering, 25th. Intern. Conf. Software Engineering, Portland, OR*, may 2003. 22, 30

[96] Jsr 118: Mobile information device profile 2.0. http://jcp.org/en/jsr/detail?id=118, December 2006. 117

[97] Jsr 139: Connected limited device configuration 1.1. http://jcp.org/en/jsr/detail?id=139, December 2006. 117

[98] Jsr 30: J2me connected limited device configuration. http://jcp.org/en/jsr/detail?id=30, December 2006. 117

[99] Jsr 37: Mobile information device profile for the j2me platform. http://jcp.org/en/jsr/detail?id=37, December 2006. 117

[100] Jsr 82: Java apis for bluetooth. http://jcp.org/en/jsr/detail?id=82, December 2006. 117

[101] Project jxta. http://www.jxta.org/, September 2006. 14

[102] Andrew M. St. Laurent. *Understanding Open Source and Free Software Licensing*. O'Reily & Associates, Inc., 1 edition, August 2004. 20

[103] Frank Leymann. Web services flow language (WSFL 1.0). Technical report, IBM Software Group, 2001. 31

[104] Libresoft/GSyC. Libre software engineering tools. http://libresoft.urjc.es/Tools/index_html, Mar 2007. 40

[105] Libresource. http://dev.libresource.org/, June 2006. 44

[106] Linux desktop testing project. http://ldtp.freedesktop.org/wiki/, January 2007. 36

[107] Linux test project. http://ltp.sourceforge.net/, January 2007. 35

[108] Richard MacManus. Yellowikis - a case study of a web 2.0 business, part 1. http://blogs.zdnet.com/web2explorer/?p=58, November 2006. 10

[109] Malaysian public sector open source software initiative. http://opensource.mampu.gov.my, September 2006. 14

[110] Mandriva. http://www.mandriva.com/, October 2006. 4, 6

[111] Mandriva. Mandriva club. http://club.mandriva.com/, December 2006. 29

[112] Mandriva. Mandriva linux package statistics. http://mandriva.edos-project.org/xwiki/bin/view/Packages/PackageStatistics, Oct 2006. 1

[113] Mandriva. Mandriva quality assurance lab's contributor's corner. http://qa.mandriva.com/twiki/bin/view/Main/QaContributorsCorner, January 2007. 37

[114] Mediawiki free software wiki. http://www.mediawiki.org/, November 2006. 10

[115] Ciarán Bryce Michel Pawlak. Deliverable d5.5.1.: A project management interface for edos, first version. Technical report, EDOS Project, March 2006. 106, 133

[116] Ciarán Bryce Michel Pawlak. Deliverable d5.5.2.: A project management interface for edos, second version. Technical report, EDOS Project, November 2006. 106

[117] Martin Michlmayr. Managing volunteer activity in free software projects. In *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track*, pages 93–102, Boston, USA, 2004. 27

[118] Martin Michlmayr. Software process maturity and the success of free software projects. In Krzysztof ZieliÅĎski and Tomasz Szmuc, editors, *Software Engineering: Evolution and Emerging Technologies*, pages 3–14, KrakÃşw, Poland, 2005. IOS Press. 29

[119] Martin Michlmayr, Francis Hunt, and David Probert. Quality practices and problems in free software projects. In Marco Scotto and Giancarlo Succi, editors, *Proceedings of the First International Conference on Open Source Systems*, pages 24–28, Genova, Italy, 2005. 34

[120] Martin Michlmayr and Anthony Senyard. A statistical analysis of defects in debian and strategies for improving quality in free software projects. In JÃijrgen Bitzer and Philipp J. H. SchrÃűder, editors, *The Economics of Open Source Software Development*, pages 131–148, Amsterdam, The Netherlands, 2006. Elsevier. 34

[121] Microsoft. Microsoft shared source initiative. http://www.microsoft.com/resources/sharedsource/, January 2007. 14

[122] Mobile information device profile (MIDP). http://java.sun.com/products/midp/, December 2006. 117

[123] A. Monk and S. Howard. The rich picture: A tool for reasoning about work context. March-April 1998. 29

[124] Thomas P. Moran, Alex Cozzi, and Stephen P. Farrell. Unified activity management: supporting people in e-business. *Commun. ACM*, 48(12):67–70, 2005. 29

[125] Matthew B. Morgan, Barton F. Branstetter, Jonathan Mates, and Paul J. Chang. Flying blind: Using a digital dashboard to navigate a complex PACS environment. *Journal of Digital Imaging*, 19(1):69–75, March 2006. 41

[126] Mozilla project. http://www.mozilla.org/, September 2006. 14

[127] Bernard Munos. Can open-source r&d reinvigorate drug research? Technical report, Eli Lilly & Co., September 2006. 1, 10

[128] Navica Inc. The open source maturity model (OSMM). http://www.navicasoft.com/pages/osmmoverview.htm, January 2007. 39

[129] Neogia. http://neogia.labs.libre-entreprise.org/index.html, October 2006. 15

[130] J. Noll and W. Scacchi. Specifying process-oriented hypertext for organizational computing. *Network and Computer Application*, 24(1):39–61, 2001. 29

[131] Notation Working Group. Bpmn charter. Technical report, Business Process Management Initiative (BMPI), November 2001. 32

[132] NSTISSA. National information systems security (INFOSEC) glossary. Technical Report NSTISSI Nř 4009, NSTISSA, September 2000. 3

[133] OASIS. Opendocument: Oasis open document format for office applications community. http://opendocument.xml.org/, November 2006. 10

[134] OASIS BPEL Workgroup. Web services business process execution language version 2.0. Technical report, OASIS, May 2006. 31

[135] OASIS UDDI Spec Technical Committee. Uddi version 3.0.2. Technical report, OASIS, November 2004. 82

[136] Ohloh.net. Ohloh.net Project. http://www.ohloh.net/, January 2007. 41

[137] Open Source Initiative. Approved licenses. http://www.opensource.org/licenses/, October 2006. 10, 13, 20

[138] Open Source Initiative. Open source definition. http://www.opensource.org/docs/definition.php, October 2006. 9, 11, 13

[139] Open source initiative (OSI). http://www.opensource.org/, October 2006. 10, 12, 13, 20

[140] Open voting consortium. http://www.openvotingconsortium.org/, Novemeber 2006. 10

[141] OpenBRR. Business readiness rating for open source - brr 2005 rfc 1. Technical report, www.openbrr.org, 2005. 39

[142] Opencola softdrink. http://www.colawp.com/colas/400/cola467_recipe.html, November 2006. 10

[143] Openoffice.org. http://www.openoffice.org/, September 2006. 10, 14, 36

[144] OpenOffice.org. Open office automated gui testing project. http://qa.openoffice.org/qatesttool/, January 2007. 36

[145] Opensourcetesting.org. http://www.opensourcetesting.org/, January 2007. 37

[146] OpenSuse. Opensuse build service. http://en.opensuse.org/Build_Service, June 2006. 4, 33

[147] Opentaps. http://www.sequoiaerp.org, October 2006. 15

[148] Organization for the advancement of structured information standards. http://www.oasis-open.org, December 2006. 31

[149] Open source quality (OSQ) project. http://osq.cs.berkeley.edu/, January 2007. 39

[150] Pentaho open source business intelligence. http://www.pentaho.org/, January 2007. 41

[151] Bruce Perens. The open source definition. In *Open Sources: Voices from the Open Source Revolution*. O'Reily & Associates, Inc., 1 edition, January 1999. 9

[152] Method for qualification and selection of open source software (QSOS) project. http://www.qsos.org/, June 2006. 4, 33

[153] Quality platform for open source software (QualiPSo) project. http://www.objectweb.org/phorum/download.php/16,271/QualiPSo_PM_Oct4.pdf, January 2007. 43

[154] Quality of open source software (QUALOSS) project. http://www.qualoss.eu/, January 2007. 37

[155] Eric. S. Raymond. The cathedral and the bazaar. http://www.openresources.com/documents/cathedral-bazaar/, August 1998. 1, 16

[156] Roberto Di Cosmo and WP2 Team. Deliverable d2.1.: Report on Software Management Dependencies. Technical report, EDOS Project, 2005. 21, 149

[157] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Martin Michlmayr. Evolution of volunteer participation in libre software projects: Evidence from Debian. In Marco Scotto and Giancarlo Succi, editors, *Proceedings of the First International Conference on Open Source Systems*, pages 100–107, Genova, Italy, 2005. 27

[158] Gregorio Robles, Jesus M. Gonzalez-Barahona, Martin Michlmayr, and Juan Jose Amor. Mining large software compilations over time: Another perspective of software evolution. In *Proceedings of the International Workshop on Mining Software Repositories (MSR 2006)*, Shanghai, China, 2006. 27

[159] Gregorio Robles-Martínez, Jesús M. González-Barahona, José Centeno-González, Vicente Matellán-Olivera, and Luis Rodero-Merino. Studying the evolution of libre software projects using publicly available data. In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*, May 2003. 40

[160] Maria Alessandra Rossi. Decoding the free/open source(F/OSS) software puzzle a survey of theoretical and empirical contributions, April 2004. 28

[161] Rth. http://rth-is-quality.com/, January 2007. 36

[162] Francesco Rullani. Dragging developers towards the core: How the free/libre/open source software community enhances developers' contribution. Technical report, LEM - Sant'Anna School of Advanced Studies, IVS - Copenhagen Business School, December 2006. 28

[163] Salome test management tool. https://wiki.objectweb.org/salome-tmf/, June 2006. 36, 44

[164] Walt Scacchi. Issues and experiences in modeling open source software processes. In *3rd. Workshop on Open Source Software Engineering, 25th. Intern. Conf. Software Engineering, Portland, OR*, may 2003. 29

[165] Science commons: Accelerating the scientific research cycle. http://sciencecommons.org/, November 2006. 1, 10

[166] Science, education and learning in freedom (SELF) project. http://www.selfproject.eu/, January 2007. 28

[167] Sendmail smtp server. http://www.sendmail.org/, December 2006. 21

[168] Skype. http://www.skype.com/, December 2006. 24

[169] I. Sommerville. *Software Engineering: 8th edition*. Addison Wesley, Essex, England, May 2006. 34

[170] Sourceforge. http://sourceforge.net/, June 2006. 4, 12, 24, 44

[171] Sourcetap. http://sourcetapcrm.sourceforge.net, October 2006. 15

[172] Software quality observatory for open source software (SQO-OSS) project. http://en.opensuse.org/Build_Service, January 2007. 37

[173] Software testing automation framework (STAF). http://staf.sourceforge.net/, January 2007. 36

[174] Richard M. Stallman. *Why "Free Software" is better than "Open Source"*, chapter 6, pages 55–60. GNU Press, Free Software Foundation, October 2002. 9

[175] Andrej Sali Stephen M. Maurer, Arti Rai. Finding cures for tropical diseases: Is open source an answer? *PLoS Med*, 1(3), December 2004. 1, 10

[176] Sun Microsystems Inc. Java 2 micro edition platform. http://java.sun.com/javame/index.jsp, December 2006. 117

[177] Sun Microsystems Inc. Openoffice.org testtool: Introduction to automated gui testing. Technical report, OpenOffice.org, September 2006. 36

[178] Sun Microsystems Inc. Remote method invocation (rmi). http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp, January 2007. 82

[179] Sun Microsystems Inc. Sun community source licensing (SCSL). http://www.sun.com/software/communitysource/, January 2007. 14

[180] Test case web (TCW). http://sourceforge.net/projects/tcw, January 2007. 36

[181] Tightening knowledge sharing in distributed software communities by applying semantic technologies (TEAM) project. http://www.team-project.eu/, January 2007. 28

[182] Testlink. http://testlink.sourceforge.net/, January 2007. 36

[183] Testopia test case management. www.mozilla.org/projects/testopia/, January 2007. 35

[184] Satish Thatte. Web services for business process design. Technical report, Microsoft Corporation, 2001. 31

[185] Linus Torvalds. The story of the linux kernel. In Sam Ockman Chris DiBona, Mark Stone, editor, *OpenSources: Voices from the Open Source Revolution*. O'Reilly an Associates, February 1999. 11

[186] Tropical disease initiative. http://www.tropicaldisease.org/, November 2006. 1, 10

[187] David Tuma. Open source software: Opportunities and challenges. *STSC Crosstalk The Journal of Defense Software Engineering*, 18(1):6–10, January 2005. 29

[188] Ubuntu linux. http://www.ubuntu.com/, August 2007. 6

[189] G. Caldiera V. Basili and H.D. Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*, pages 528–532. John Wiley & Sons, Inc., 1994. 38

[190] Andreea Vasiliu. Dashboards and scorecards: Essential tools for managing business performance. *DM Direct Special Report*, April 2006. 41

[191] Vores al : An open source beer. http://www.voresoel.dk/main.php?id=70, November 2006. 10

[192] W3C. Web service choreography interface (WSCI) 1.0. Technical report, W3C, August 2002. 31

[193] James Over Watts Humphrey, Michael Konrad and William Peterson. Future directions in process improvement. *STSC Crosstalk The Journal of Defense Software Engineering*, 20(2):17–22, February 2007. 30

[194] Dawid Weiss. Quantitative analysis of open source projects on sourceforge. In *Proceedings of the First International Conference on Open SourceSystems, Genova, Italy*, July 2005. 40

[195] WfMC. Xml process definition language (XPDL). Technical report, Workflow Management Coalition (WfMC), October 2005. 32

[196] Davis A. Wheeler. How to evaluate open source software / free software (OSS/FS) programs. Technical report, January 2007. 34

[197] Wikimedia foundation. http://wikimediafoundation.org/wiki/Home, November 2006. 10

[198] Wikipedia, the free encyclopedia. http://www.wikipedia.org/, November 2006. 1, 10, 28

[199] Wikipedia. Shared source on wikipedia. http://en.wikipedia.org/wiki/Shared_source, January 2007. 14

[200] Wiktionary, the free dictionary. http://www.wikionary.org/, November 2006. 1, 10

[201] Sean Witty. Best Practices for Deploying and Managing Linux with Red Hat Network, December 2004. 4, 33

[202] Workflow management coalition. http://www.wfmc.org/, January 2007. 32

[203] Web services description language (WSDL) 1.1. http://www.w3.org/TR/wsdl, January 2007. 31

[204] Xplanet. http://xplanet.sourceforge.net/, October 2006. 24

[205] Yellowikis open business listings. http://www.yellowikis.org/, November 2006. 10

[206] Andy Zeneski. The open for business project: Workflow engine guide. Technical report, Apache Open For Business Project, 2004. 32

This research work was supported by the European Commission with the project EDOS
(FP6-IST-004312) under 6th Framework Programme