Enabling the Definition and Enforcement of Governance Rules in Open Source Systems

Javier Luis Cánovas Izquierdo
AtlanMod, École des Mines de nantes – INRIA – LINA
Email: javier.canovas@inria.fr

Jordi Cabot*†
*ICREA
†Internet Interdisciplinary Institute, UOC
Email: jcabot@uoc.edu

***Abstract*—Governance rules in software development projects help to prioritize and manage their development tasks, and contribute to the long-term sustainability of the project by clarifying how core and external contributors should collaborate in order to advance the project during its whole lifespan. Despite their importance, specially in Open Source Software (OSS) projects, these rules are usually implicit or scattered in the project documentation/tools (e.g., tracking-systems or forums), hampering the correct understanding of the development process. We propose to enable the explicit definition and enforcement of governance rules for OSS projects. We believe this brings several important benefits, including improvements in the transparency of the process, its traceability and the semi-automation of the governance itself. Our approach has been implemented on top of Mylyn, a project-management Eclipse plug-in supporting most popular tracking-systems.**

I. Introduction

The development of large-scale software is a long-life process which has to cope with a huge number of development tasks consisting of either implementing new issues or fixing bugs [1]. Effective and precise prioritization of these tasks is key for the success of the project.

With this purpose, each project defines and applies its own set of governance rules, that is, a set of instructions which describe how to contribute to the project and how decisions regarding the acceptance/rejection of such contributions are going to be made. For instance, rules can be as simple as *team leaders decide the tasks to do* or more complex as *the task to be done will be the one most voted by the developers participating in the project*. Governance rules enable the coordination of developers in order to advance the project during its whole lifespan and, more importantly, their evolution allows the project to adopt societal changes on the way people want to collaborate, thus promoting the sustainability of the development process.

Despite their importance, in practice, governance rules are hardly ever explicitly defined, specially in the context of Open Source Systems (OSS), where it is hard to find an explicit system-level design, a project plan or list of deliverables [2], [3], [4] (e.g., browsing issue and bug trackers for any project quickly uncovers "forgotten" requests from several years ago where new users periodically comment on trying to guess if that feature/bug has been implemented/fixed or not, if it will be, and how they can help to push it up on the priority list). Moreover, the support for enforcing such governance rules is even more limited. This hampers and can even scare away the participation of new users/developers who must invest some time understanding the "culture" of the project.

To alleviate this situation, mechanisms to facilitate the communication and assignment of work are considered crucial [5], [6]. Tracking systems (i.e., bug-tracking such as Bugzilla[1] and issue-tracking systems such as Mantis[2]) are broadly used to manage the tasks to be performed. Other collaborative tools such as mailing-lists or forums are also used to coordinate the developers involved in the project. While these tools provide a convenient compartmentalization of work and effective means of communication, they fall short in providing adequate support for specifying and enforcing governance rules (e.g., automatically prioritizing tasks based on votes, easy tracking of decisions made in the project, etc.).

Therefore, we believe the explicit definition of governance rules along with the corresponding infrastructure to help developers follow them would have several benefits, including improvements in the transparency of the decision-making process (promoting the understanding of the project evolution), traceability (being able to track why a decision was made and who decided it) and the automation of the governance process (e.g., liberating developers from having to be aware and follow the rules manually, minimizing the risk of inconsistent behaviour in the evolution of the project).

This paper makes the following original contributions:

- We report on the results of a survey conducted to assess the importance of making explicit the governance rules. The survey revealed that, while contributing is usually easy, knowing how such contributions are going to the treated is hard. We confirmed that a proper definition of the project governance would help to understand how it evolves and therefore encourage people to contribute.
- We propose a new Domain-Specific Language (DSL) to let project managers easily define the governance rules of their projects, thus promoting sustainability. The DSL is defined based on our analysis of the (implicit) rules used in the governance of a set of OSS development projects.
- We provide a collaborative infrastructure to enforce those rules as part of the project evolution, thus promoting traceabiliy and automation of the process. Our approach

[1] http://www.bugzilla.org/
[2] http://www.mantisbt.org/

has been implemented on top of Mylyn, a project management plugin for Eclipse with connectors for most popular tracking systems. Thanks to these connectors our approach can be used together with the existing tracking systems already in place in OSS projects.

The rest of the paper is structured as follows. Section II reports on the results of a survey confirming the importance of making explicit the governance rules of OSS projects. Section III analyzes the governance rules used in several OSS projects. Our DSL to define governance rules is described in Section IV while the infrastructure needed to apply them is presented in Section V. Section VI presents an illustrative case study. Finally, Section VII shows the related work and Section VIII finalizes the paper.

II. Making the Governance Rules Explicit

To assess the importance of having explicit governance rules in OSS projects we conducted a survey. Our aim is to answer the following research questions: (RQ1) How hard is it to contribute to OSS projects?, and (RQ2) is it feasible to make explicit the governance rules? if so, is it useful? To promote the participation, the survey was announced and distributed online, being freely accesible and targeting any developer willing to participate. To maximize the number of responses, we kept the number of questions as reduced as possible. The survey comprised questions on three different aspects:

- QA **User profile**, including questions to characterize the survey participant according to his/her experience contributing to OSS projects, the role (i.e., project leader, contributor, end-user, no expertise or bad experience), and the number of projects contributed (if any).
- QB **Understanding of the contribution process**, which analyzes how difficult it is to understand how to contribute (e.g., providing source code, notifying bugs, proposing new features, etc.) to the project. This category comprised questions to assess the difficulty level (ranking from 1 - easy, to 5 - hard) of the main contribution activities (i.e., how difficult was to propose a change request, a bug or a patch) and understanding the governance rules applied in the process (i.e., how difficult was to know when a proposal was accepted or whether/when it will be included in the next release).
- QC **Governance rule definition**, with questions to analyze the opinion regarding the feasibility and usefulness of making governance rules explicit.

The survey was promoted through social media (i.e., twitter, facebook, etc.) and in one of the author's blog[3]. It was available for one month and answered by 51 people that had the option to remain anonymous if they wished[4]. All participants answered the full set of questions. A summary of the results is shown in Figure 1.

[3]http://modeling-languages.com/dont-contribute-open-source-projects

[4]The results are available at http://goo.gl/tvzYz9

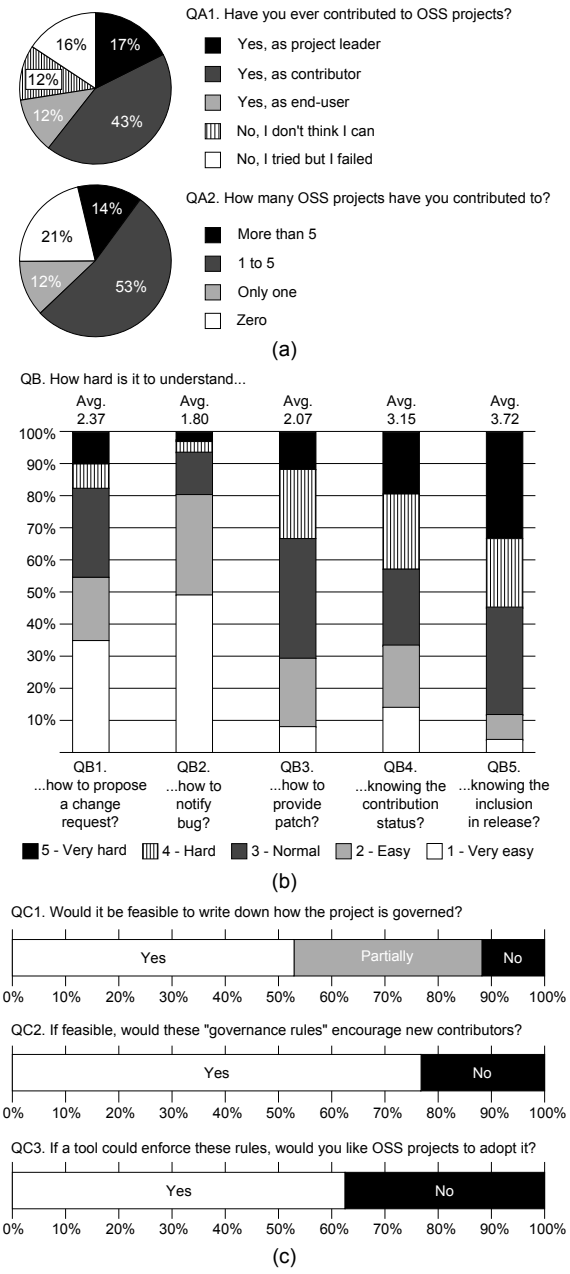

Fig. 1. Results of the survey.

Figure 1a depicts the main results for the questions profiling the surveyed developers (i.e., QA questions). Note that respondents who tried but failed to contribute (i.e., QA1) sometimes considered their attempt as contribution (i.e., QA2), which will influence the remainder of the survey. As can be seen, the great majority of the respondents are contributors (see QA1 results) who have participated in 1 to 5 projects (see QA2 results).

Figure 1b shows the results regarding the difficulty of understanding the contribution process and addresses RQ1. The results reveal that in average contribution activities are easy to understand: proposing a change request receives an average value of 2.37 (see QB1 results), notifying a bug receives an average value of 1.80 (see QB2 results) and sending a patch receives an average value of 2.07 (see QB3

results). However, the results regarding the understanding of the application of governance rules get worse evaluations in average. Thus, knowing the status of a proposal (i.e., QB4) or its inclusion in the next release (i.e., QB5) receive an average value of 3.15 and 3.72, respectively.

Figure 1c shows the results with regard to the governance rule definition, thus answering RQ2. Respondants believe that most governance rules can be written down fully or partially (see QC1 results) and, in those cases where it is possible, they also acknowledged its positive impact to encourage new contributors to participate (see QC2 results) in the project. Furthermore, the results also show that a tool enforcing governance rules would be welcome in the context of OSS projects (see QC3 results), though with less support, which may indicate reluctance to (semi)automate the development process.

Our survey also included an open question to gather opinions about how to make the understanding of the governance easier. Among the answers, it is remarkable that a couple of developers noted that it is a common problem to make the effort of sending patches and then realize that they are delayed or ignored with no clear reason (e.g., one of the answers was *I really wanted to work on a project and fixed few bugs. They answered immediately that they will use it, but it then took more than a year until they really did*). There were also other answers commenting about the need of better mechanisms to facilitate the understanding of how the project is governed, such as better user interfaces (e.g., *one for the end-user (simple, elegant, clear) and another for developers/maintainers with full information*), delegating a person to specifically care about first-time contributions or ranking the projects according to their *friendliness*. These answers reveal that developers are willing to contribute but the effort to make is still too high for some of them.

Although the results of our survey may not represent the universe of all developers in OSS projects, we believe that they are illustrative enough to motivate the need of explicitly defining the governance rules behind OSS projects.

III. How OSS Projects are Governed

While the survey presented in the previous section shown the need of explicitly defining the governance rules in OSS projects, in this section we analyze eight well-known OSS projects in order to empirically study how existing projecs are governed. This information will help us to further confirm the survey results and to know better this domain in order to develop our solution as described in the next sections.

Most large OSS projects are organized in terms of tasks (either feature requests or bugs) that are solved by means of patches (implementing new features in response to the requests or correcting the bugs) which at some point can be selected to be part of the next software release. Figure 2 summarizes this workflow. The workflow includes three main decision points, namely: (1) task selection, (2) patch review and (3) release selection. Decisions at each point are taken based on the governance rules defined for the project.

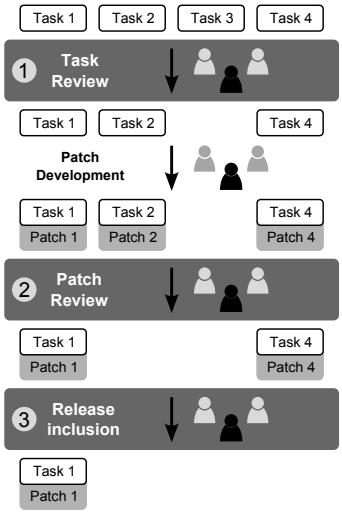

Fig. 2. Workflow followed by a task and main decision points.

We have studied eight OSS projects where external contributions are welcome (i.e., Android[5], GNOME[6], Apache web server[7], Mozilla[8], Python[9], Moodle[10], EMF[11], MoDisco[12]). Each project was analyzed according to nine dimensions targeting three main viewpoints, specifically: organizational viewpoint, which studies how developers are organized (mainly hierarchically or not) (*organization* dimension), the main communication means (*coordination* and *tracking system* dimensions) and who can participate (*participation* dimension); development process viewpoint, which assesses how the review process is done in each one of the three main decision points (*task review*, *patch review* and *release decision* dimensions); and governance rule definition viewpoint, which studies where the governance rules are usually defined and who is involved (*Rules Def. / App.* and *Roles* dimensions). We believe these dimensions cover all elements that are part of the governance of an OSS project. Table I shows a summary of the data we gathered. Next we describe the results in more detail.

- **Organization**. The organization followed by a project summarizes its main development philosophy. The vast majority of analyzed projects follow a hierarchical scheme, meaning that there exist several hierarchical levels of user groups collaborating in the development (e.g., leaders, contributors, users, etc.), where each group/role has different permissions and tasks. A good example is the Android project, where each role has assigned a particular function in the process (e.g., *approvers* approve tasks, *verifiers* review patches, etc.) supporting the project leader. On the other hand, Apache differs from the others

[5]http://www.android.com
[6]https://www.gnome.org
[7]https:/www.apache.org
[8]https://www.mozilla.org
[9]https://www.python.org
[10]https://moodle.org
[11]https://www.eclipse.org/emf
[12]https://www.eclipse.org/modisco

TABLE I
COMPARISON OF HOW OSS SYSTEMS ARE GOVERNED.

Project	Organization	Coordination	Tracking system	Participation	Task Review	Patch Review	Release Decision	Rules Def. / App.	Roles
Android	Hierarchy	Forum	Git Gerrit	Anyone	Yes (Approver)	Yes (Verifier)	Yes (Project Lead)	Documentation / Track system	Contributor Developer Verifier Approver Project Lead
GNOME	Hierarchy	IRC mailing-list	Bugzilla	Anyone	Yes	Yes (Bug Squad)	Yes	Documentation / Track System	Contributor Mentor
Apache Web Server	Meritocracy	Mailing-list	Bugzilla	Anyone	Yes (Voting)	Yes (Voting)	Yes (Voting)	Documentation / Mailing-list	User Developer Committer PMC Member PMC Chair
Mozilla	Hierarchy	Forum Mailing-list IRC	Bugzilla	Anyone	Yes (Module Owner)	Yes (Module Owner & Super-reviewers)	Yes (Designated Group)	Documentation / Track System & Mailing-list	User Committer Module Owner Super-reviewer
Python	Hierarchy	Mailing-list IRC Blogs	Mercurial Roundup	Anyone	No	Yes (Reviewer)	Yes (Core Developer)	Documentation / Track System	Contributor Reviewer Core Developer
Moodle	Hierarchy	Forum	Moodle tracker	Anyone	Yes (Component leads)	Yes (Developers)	Yes (Component Leads)	Documentation / Track System	Users Developers Component Leads Integrators Testers Manteiners
EMF	Hierarchy	Forum	Bugzilla	Anyone	Yes (Committer)	Yes (Committer)	Yes (Project Leader)	Documentation / Track System	User Contributor Committer Project Leader
MoDisco	Hierarchy	Forum	Bugzilla	Anyone	Yes (Committer)	Yes (Committer)	Yes (Committers)	Documentation / Track System	User Contributor Committer Project Leader

by using a meritocracy organization where users can gain merits as they contribute to the project.

- **Communication mechanisms**. This feature includes the main tools used to help users to collaborate during the development process. As can be seen, mailing-lists and forums are the most popular tools. Note that even though these tools are useful to keep in touch with the rest of developers, they are normally used to enforce some governance rules as well, which goes beyond their original purpose. This is the case of Apache, which uses a mailing-list to vote which feature requests/bugs should be implemented/fixed next.
- **Tracking system**. The managament of tasks is performed by tracking systems, Bugzilla being the most popular one. Interestingly enough, tracking systems are also used as a forum to discuss possibles tasks which are not mature enough to be considered as real issues. For instance, in the GNOME project, some tasks[13] are used to openly discuss possible changes in the development strategy.
- **Participation**. In general, in OSS, anyone is able to take part in the development, i.e., it is not necessary to be an official developer to inform new feature requests or bugs.
- **Task review**. Once a task (i.e., feature request/bug) is notified, it can be reviewed to be accepted or rejected. Except for Python, where all the tasks are accepted and only reviewed if they include the corresponding patch, the rest of the projects have a decision process to accept or reject the task proposal. In general, the decision process is made by either the leader (e.g., component leaders in Moodle) or by unanimous agreement if there are several leaders (e.g., in big Eclipse-based projects such as EMF).
- **Patch review**. Tasks may come with the corresponding patch implementing the improvement (if it is a feature request) or the fix (if it is a bug). Before incorporating the patch into the product, it is possible to perform a revision and test the quality of the patch. All the studied projects include a review process which analyzes the patch and eventually decides its acceptance or rejection.
- **Release Decision**. As the tasks and the corresponding patches are accepted, new software product releases have to be published. In the analyzed projects, the decision of selecting when to perform the release and which tasks should be incorporated is normally taken by the product/component leader or by unanimous agreement when there are several leaders. The only exception is the decision process for Apache, where a ballot takes place.
- **Governance rule definition and application**. This feature shows how governance rules are defined and applied. In general, none of the projects explictly defines these rules, and to make things worse, the result of their application is scattered throughout the management systems (i.e., track systems or mailing-lists).
- **Roles**. The group of users collaborating in the development are classified according to a set of roles. All the analyzed projects include both the role of developer (a.k.a. contributor/committer), who can commit changes into the

[13]http://felipec.wordpress.com/2011/09/23/no-gnome-doesnt-want-user-feedback-how-i-argued-in-favor-of-voting-in-bugzilla-and-got-banned-as-a-result

source code repository of the software projecs, and the corresponding role for leaders (a.k.a owners/chairs).

In summary, most projects are hierarchical where contributors are driven by a leader (or a set of leaders) who normally decides which tasks should be completed first. Patches are normally reviewed and tested by contributors while the product release is decided by a leader group as well. Thus, the resulting governance rules are mainly controlled by the group of leaders, who can decide how the software product should evolve. When there is a group of leaders, the decision is normally made by unanimous agreement and normally using collaboration tools (e.g., email, mailing-lists or forums).

The development of Apache web server is the main exception. The project is governed by a voting-based decision rule where all contributors can decide which tasks should be accepted and which ones should be included in the final release. It is important to note that although anyone can vote, only the votes from contributors are binding, the rest are helpful to see the general opinion of the community. The project also establishes how the voting process should be performed. Thus, if the task involves changes in the source code, the voting should result in an unanimous agreement in order to make the change and at least three votes must be casted. Otherwise, if the change does not involve code, a majority of positives votes (and at least three) are required. Moreover, any negative vote must include a rationale, which can therefore help to solve the disagreement.

Interestingly enough, in all the analyzed projects it was not trivial for us to discover the governance rules being applied since the available information was scarce and normally scattered among the documentation of the project. In fact, some projects such as GNOME recommend to be patient since it can take a long time to become a contributor. Potential contributors are therefore required to observe existing mailing lists, conversations on IRC, etc., to discover the way of working in the project. This evidence confirms the results gathered in the survey presented in the previous section.

Moreover, the result of the application of these rules is directly updated in the tracking systems, where normally there is no traceability information that helps to clarify later on why that decision was taken nor the possible discussion threads (maybe taken place outside the tracking systems, e.g., by chat or email as it is the case for smaller projects such as MoDisco) among the leaders that led to that decision.

Clearly, making explicit the governance rules followed in a project could help the developer community to understand and enforce those rules as part of their daily development activities. Next sections describe our proposal to define and enforce governance rules.

IV. Definition of Governance Rules

In order to define the set of governance rules for development tasks, we propose to use a new Domain-Specific Language (DSL) providing specialized constructs that facilitate the specification of decision rules to be followed during the management of development tasks. Our goal is to provide a simple but precise language to make explicit such rules, thus promoting the understanding of how the project is developed and eventually encouraging developers to contribute. Based on our validation, the language is expressive enough to cover typical governance scenarios; nevertheless it could be tailored to cover more complex situations. Furthermore, this opens the door to automatically enforce those rules as discussed later on.

A DSL is defined by three main elements [7]: abstract syntax, concrete syntax and semantics. The abstract syntax defines the main concepts of the language and their relationships, and also includes well-formedness rules constraining how to use them. The concrete syntax defines the language notation (textual, graphical or hybrid) and a translational approach is normally used to provide semantics.

Our DSL has been defined in the context of Model-Driven Engineering, thus we use metamodelling techniques [8] to define the abstract syntax. Similar to grammars, metamodels restrict the possible structure of valid models for a DSL. More specifically, we have defined a governance metamodel to represent the concepts and relationships needed to specify governance rules. Governance models conforming to this metamodel (i.e., models that can be expressed as valid instances of this metamodel) represent specific sets of governance rules of software projects. As concrete syntax, we have opted for a textual language following a typical block-based structure. Thus, each instance of a metaclass is textually represented by its keyword and a block that contains the name and value of its attributes. Containment references are represented as nested blocks while non-containment references use an identifier to refer to the referred element. Both abstract and concrete syntaxes have been implemented in our prototype tool [9].

The abstract syntax metamodel of the language is shown in Figure 3. The concepts represented in the metamodel cover all the governance aspects identified in our study of OSS projects. In particular, a development project (`Project` metaclass) includes a group of roles of users (`roles` reference), rules (`rules` reference) and deadlines (`deadlines` reference). A role has an identifier (`id` attribute in `Role` metaclass) and represents a group of people in the development community who can vote. Decision rules are applied to a particular type of collaborations (`appliedTo` attribute) according to their nature in the tracking system (i.e., task, patch or comment) and at a specific moment (`stage` attribute) of the process (i.e., task review, patch review and release). The scope of the rule can also be defined (`queryFilter` attribute) (e.g., only those tagged as high priority).

We have predefined several types of decision rules (included in the hierarchy with root `Rule`), which cover the cases analyzed before:

- **Majority Rule**. Under this rule (`Majority` metaclass) only those collaborations which have received a support over 50% will be selected. The way of counting votes may differ depending on who voted during the lifespan of the rule. Since different terminology is used to refer the several types of majority rules (e.g., *plurality* or *relative majority* used in North America is called *simple majority*

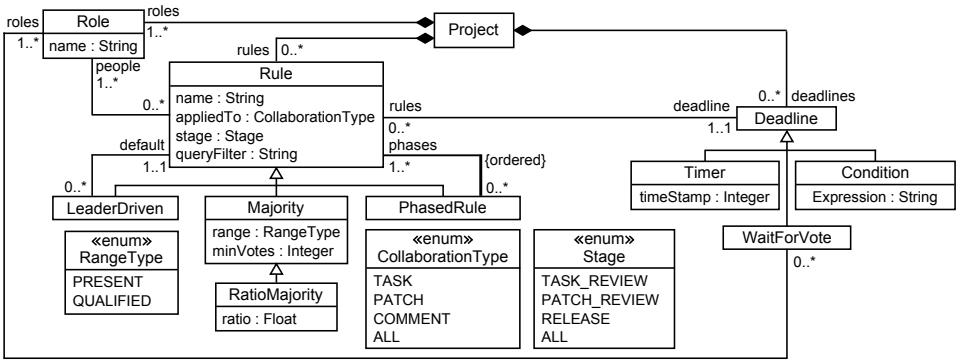

Fig. 3. Abstract syntax metamodel of our DSL to represent governance rules.

in Europe), we will use the neutral term *majority range* to determine exactly how the votes should be counted. Thus, if the range is `present`, the majority is based on the votes of those participants presented during the voting time. Otherwise, a `qualified` range is based on the votes of those participants qualified to vote (having voted or not, where abstention means disagreement). A majority rule can also require a minimum number of votes to be triggered (`minVotes` attribute).

As an example, a majority rule to be applied only to tasks, voted by the committers if they are presently available, with no need of minimum votes and with a deadline of seven days from the task creation date would be specified with our textual concrete syntax as:

```
Project myProject {
  Roles: Committers
  Deadlines:
    myDeadline : 7 days
  Rules:
    myMajorityRule : Majority {
      applied to Task
      when TaskReview
      people Committers
      range Present
      minVotes 0
      deadline myDeadline
    }
}
```

A different percentage for the majority can also be set. In this case the rule will be called ratio majority (`RatioMajority` metaclass) and the ratio value must be specified. For instance, this type of rule would allow implementing well-known majorities such as three-fifths or two-thirds, which may be required for changes on fundamental laws. As example of ratio majority, the following definition modifies the rule presented before to be ratio-based with a ratio of acceptance of 75% and to be voted only by the project leaders.

```
Project myProject {
  Roles: Committers, ProjectLeader
  Deadlines:
    myDeadline : 7 days
  Rules:
    myRatioRule : Ratio {
      applied to Task
      when TaskReview
      people ProjectLeader
      range Present
      minVotes 0
      ratio 0.75
      deadline myDeadline
    }
}
```

- **Leader-Driven Rule**. When the decision of accepting a collaboration relies on a user playing the role of leader (e.g., component or project leader), a leader-driven rule (`LeaderDriven` metaclass) is followed. Thus, this decision rule relies on the leader of a collaboration to decide its acceptance. Next section describes how the leader is represented. The leader must also define a default rule (`default` reference) where to delegate the decision in case the leader is not available (i.e., the leader doesn't vote before the deadline)

  An example of leader-driven rule to be applied only to tasks, with a majority default rule to be applied when the leader does not make the decision and with a deadline of seven days from the task creation date would be (note that the `myMajorityRule` rule defined before is reused as default behavior):

```
Project myProject {
  Roles: Committers
  Deadlines:
    myDeadline : 7 days
  Rules:
    myMajorityRule : Majority { ... }
    myLeaderDrivenRule : LeaderDriven {
      applied to Task
      when TaskReview
      default myMajorityRule
      deadline myDeadline
    }
}
```

- **Phased Rule**. This is a composite rule which allows applying several rules in a chained way. The rules are defined and applied in an ordered way (`phases` reference). Thus, a set of collaborations are selected according to the first rule, the selected ones are then voted again and filtered according to the second rule, and so on.

  An example of phased rule composed of two phases where the first one applies a majority among the committers and with a deadline of seven days from the task creation date, and the second phase, which has a deadline of seven days after the first phase has been done, applies

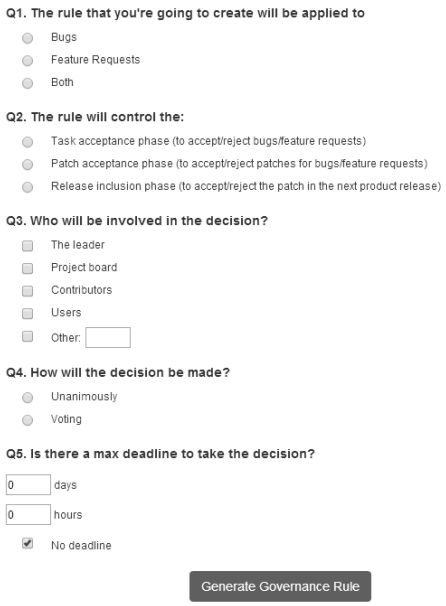

Fig. 4. Form-based generator to define governance rules.

a leader-driven rule among the project leaders would be (note that `myMajorityRule` and `myRatioRule` are reused):

```
Project myProject {
  Roles: Committers, ProjectLeader
  Deadlines:
    myDeadline : 7 days
  Rules:
    myMajorityRule : Majority { ... }
    myRatioRule : Ratio { ... }
    myPhasedRule : Phased {
      phases {
        myMajorityRule
        myRatioRule
      }
    }
}
```

Regarding the deadlines that trigger a decision rule, we have currently defined three covering deadlines based on: (1) time (`Timer` metaclass), (2) an OCL[14] condition to be fulfilled in the collaboration (`Condition` metaclass) (e.g., the change of a tag in the collaboration model) and (3) a set of users have voted (`WaitUserVote` metaclass).

A. Rule Generator

To facilitate the use of the DSL beyond the textual syntax we also provide a form-based generator to define governance rules (i.e., our DSL instances). Figure 4 shows a snapshot of the tool[15] where developers can configure the governance rule for their development process and generate both the English text and the DSL instance describing such rule.

Additionally, the tool creates a permanent link to help referring the governance rule from any project (i.e., the link automatically shows the rule in English and the DSL instance). We envision that projects may define a `governance.md` file where governance rules can be specified along with the corresponding permanent links. To start with, we are already using this approach in the development of our research projects[16].

B. DSL validation

In order to validate our DSL, we circulated the form-based generator among a group of developers who have participated in the development of OSS projects. The group was composed of 7 developers, with the following main roles: 3 leaders, 2 contributors, and 2 end-users. All of them were participating actively in the corresponding software project. After they tested the generator, we asked them several questions to assess their opinion on the expressiveness of the DSL. In particular, we were interested in learning whether they had been able to specify all the rules governing the projects they were working on with our DSL.

The participants reported a proper coverage for representing governance rules used in their projects but asked for an extended support for (1) roles and (2) deadlines. It turned out that both aspects where already supported by the DSL but were simplified in the implementation of our generator.

Regarding the support for roles, it was required to support the definition of custom roles. This issue appeared since the form-based generator included a fixed number of roles (the language abstract syntax actually allows for any type of roles, cf. `Role` concept). This has been now incorporated into the current version of the generator (the one shown in this paper).

With regard to the support for deadlines, the participants asked for the ability to define deadlines based on metadata (e.g., if the priority metadata is high, the deadline will be shorter). This issue was supported by the language (`Condition` element allows developers to define deadlines according to a OCL-based condition) but, for the sake of the simplicity, we decided not to over-complicate the form-based generator. Thus in those cases where it is necessary to define a more specific deadline, developers should edit the resulting DSL definition to include the OCL expression

V. Enforcement of Governance Rules

While the DSL enables developers to represent and understand how the project is governed, some projects may also want to use the specification of the rules to automatically manage the project itself. In order to do that we would need to: (1) provide support for recording all information regarding the community interactions (e.g., proposals, votes, etc.), and (2) implement a decision engine that can use this information and the governance rules to make the corresponding decisions and update the project status accordingly.

Thus, additionally to our DSL, we provide the needed infrastructure to "execute" the DSL specifications so that, beyond improving the understanding of the project internal organization, they can also be enforced to support and guide the collaboration among the project participants.

Figure 5 illustrates how the automatic enforcement of the DSL rules would change the group dynamics at a particular

[14]http://www.omg.org/spec/OCL

[15]Available at http://goo.gl/Fq79Wv

[16]For instance, in http://goo.gl/H8MMSX

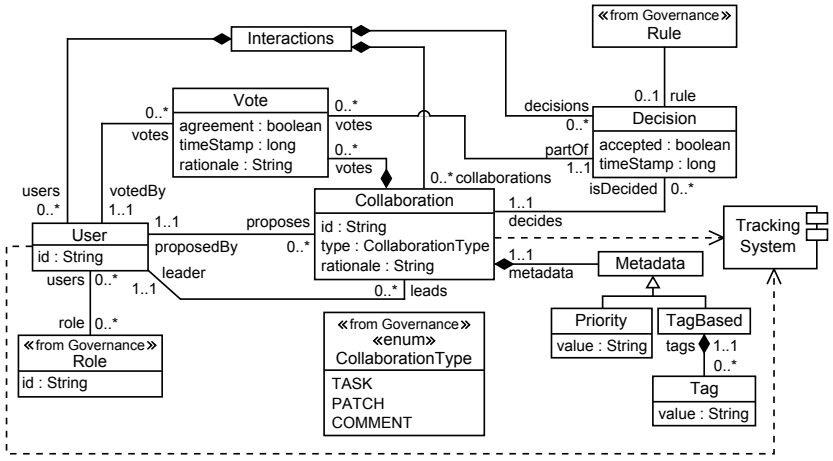

Fig. 6. Metamodel to represent collaborations.

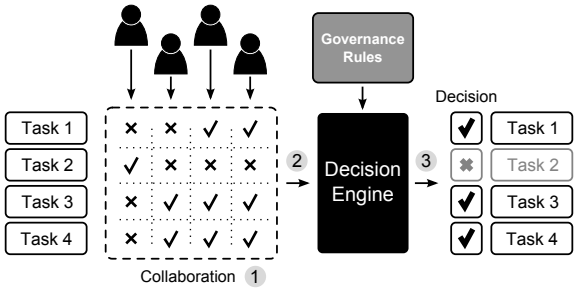

Fig. 5. Decision process proposed.

decision point. Given a set of tasks to be discussed (i.e., to accept them, accept a patch for them or include them in a release), users can vote for/against them (step 1), then a decision engine analyzes the votes according to the governances rules defined (e.g., total agreement, simple majority, etc.) (step 2) and, as a result, the status of the tasks is changed based on the decisions taken by the engine (step 3).

To record the group interactions we use the metamodel shown in Figure 6. A particular collaboration (`Collaboration` metaclass) is represented by an identifier (`id` attribute) and a rationale explaining the collaboration (`rationale` attribute). There are three types of collaboration elements (`type` attribute in `Collaboration` metaclass): *tasks*, which represent a request for a feature request or a bug; *patches*, which represent the solution for a task; and *comments*, which represent comments and annotations that users can make as a follow up of other collaboration elements. A collaboration element can also have some metadata information (`metadata` reference) and has a proposing user (`isProposed` reference) and a leader (`leader` reference). During the collaboration, a user (`User` metaclass) is identified by an identifier (`id` attribute) and belongs to a one or more roles (`roles` reference). Note also that, both, collaboration elements and users are linked to the tracking system (see `Tracking System` component) to be able to link these interactions with the status of the corresponding elements in the official project tracking system. The metamodel also allows representing users' votes (`Vote` metaclass) to express their agreement/disagreement with a collaboration (`agreement` attribute). A vote has a timestamp (`timeStamp` attribute) and a rationale (`rationale` attribute) explaining the reason for the positive/negative vote.

When the deadline for a governance rule passes, the decision engine receives the collaboration model as input and follows the rule instructions to process the data and take a decision. This process generates a new decision (`Decision` metaclass) and trace it back to the affected collaboration elements (`isDecided` reference). A decision can accept/reject a collaboration element (`accepted` attribute), includes a timestamp of the moment when the decision was made (`timeStamp` attribute) and refers to the rule applied (`rule` reference). A collaboration element has usually only a decision element assigned except when a phased rule is used, in this case the decisions for each phase and the final decision are also stored in the model.

The decision engine is provided as part of our prototype tool [9], which has been implemented as an Eclipse plug-in, and takes care of also updating the tracking system based on the changes performed on the collaboration model. The tool has been implemented on top of Mylyn[17], a project management plugin for the Eclipse platform with connectors for most popular tracking systems.

VI. Case Study: Apache

We have used our approach to model the decision process followed in the Apache project (where the choice of Apache was motivated by the fact that it has a more complex governance process than other analyzed projects). Our goal is twofold: (1) illustrating the use of our approach and (2) checking that our DSL is able to represent complex governance processes.

We first represent the governance rules used to select tasks to complete. As described before, there are two types of voting procedures in Apache depending on whether the task to work on involves source code changes or not. The DSL excerpt defining these Apache governance rules would be:

[17]http://www.eclipse.org/mylyn

```
Project Apache {
  Roles : Developer, Committer,
          PMCMember, PMCChair
  Deadlines :
    twoDays : 2 days
  Rules :
    codeRule : Ratio {
      applied to Task (tag = code)
      when TaskReview
      people Developer, Committer,
             PMCMember, PMCChair
      range Present
      minVotes 3
      ratio 1
      deadline twoDays
    },
    noCodeRule : Majority {
      applied to Task (tag != code)
      when TaskReview
      people Developer, Committer,
             PMCMember, PMCChair
      range Present
      minVotes 3
      deadline twoDays
    }
}
```

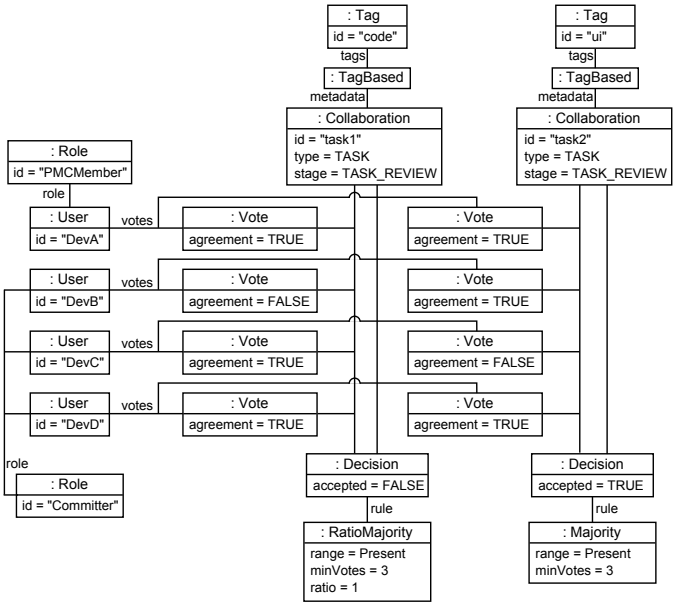

Fig. 7. Example of collaboration in Apache.

The example defines four roles and two rules called `codeRule` and `noCodeRule`. The former is a ratio majority rule which is applied to those tasks including the tag `code` (i.e., tasks involving changes in the code). All people belonging to the roles linked to the rule (see tag `people`) and presently available (see tag `range`) can vote. It is required a ratio of agreement of 100% (i.e., meaning that everybody has to agree) and at least three votes. Finally, the rule defines a deadline of two days, although the precise length is not specified in the project documentation. On the other hand, `noCodeRule` is a majority rule which is applied to those tasks not including the tag `code`. Like the previous rule, it is required at least three votes to make a positive decision and the deadline is two days after the creation of the task.

To illustrate the application of the previous rules, Figure 7 shows a possible collaboration model for a scenario where developers should take a decision about two task proposals with identifier `task1` and `task2`. The former includes the tag `code` while the latter does not. For the sake of the clarity and conciseness, the `rationale` and `timeStamp` attributes have been removed in the Figure. As can be seen, after voting, although the first task has received more than three votes, it it is rejected because there is one negative vote. On the other hand, the second task is accepted because, although there is also a negative vote, the majority agrees with the change.

Note that instead of the current process (based on mail communications and manual counting of the votes) our approach (and tool support) could automatically take the decisions based on the rules and the interactions taking place between the developers. Moreover, both the collaboration and decisions are stored along with the element decided (i.e., the task status change), thus avoiding scattering information along the project management tools (i.e., the mailing-list in this case) and improving the sustainability and traceability of the process.

VII. Related Work

The study of how people coordinate to develop a software system has been a research topic for a long time [10], [11], [12], [13], [14], including the broader topic of IT governance [15], [16], [17], [18]. However, to the best of our knowledge, little attention has been paid to the precise definition and support for the governance rules in (OSS) projects.

The work presented in [19] analyzes how broadcast based peer review influences OSS development processes. Developer characteristics such as their expertise or behaviour are studied to analyze how they may impact in the development task selection and decision making. They report that such behavior can lead to frustration in newcomer contributions who do not understand the development process. Unlike this work, ours tries to explicitly define how decisions are made to help anyone to understand how the project is governed.

Some works report on concrete case studies. For instance, [1] reports on the coordination activities performed to solve bugs (e.g., how bugs are detected and closed, means used to coordinate the work, etc.). In [20] authors present how developers use mailing lists during the development of FLOSS systems (e.g., which are the main topics discussed or the level or participation of the different user roles).

Other approaches focus on leveraging collaboration information obtained by mining software repositories to infer coordination relationships and structures. The work presented in [21] describes an approach to discover potential social structures from the threads created in the mailing list of the project. In [22] visualization techniques are applied to easily discover communication patterns from Github repository metadata (e.g., the effect of geographic distance among developers, influence among cities, etc.). The tool called CrowdWeaver is presented in [23], which allows coping with the complexity of managing crowdsourced projects. These tools could be adapted

to facilitate the discovery of existing governance rules in a project to later represent and manage them using our approach.

Our DSL can be related to process description and workflow languages ([24], [25] among others), where process and decision paths can be defined. However, our approach focuses on the decision rules used to make collaborative decisions, an aspect usually not included in these languages.

Concrete methodologies for collaboration strategies have also been proposed. For instance, [26] and [27] present approaches to support collaborative processes in groupware systems and online creative projects, respectively. However, they do not provide mechanisms to define and enforce the governance rules to apply in each project.

Collaboration has also been the focus on two other DSLs. The approach presented in [28] describes a DSL to represent synchronous collaboration workflows in modeling tools (e.g., the steps needed to create a class diagram when several users are collaborating). In [29], authors presented a DSL-based tool to record the collaboration interactions taken place during the development of a DSLs. Again, these approaches do not provide any support for the definition nor enforcement of the rules controlling such collaborations.

VIII. Conclusion

In this paper we have presented an approach that enables the explicit and precise definition of the governance rules for a development project and, if desired, also its automatic enforcement. Our approach comprises a new DSL for specifying the rules, a collaboration infrastructure to record the interactions (and trace them back when necessary) and a decision engine to automatically update the task status according to the rules. An implementation of the approach has been made available in [9]. Although our approach mainly targets the OSS community, in particular, those projects where any developer can freely contribute, we believe it can also be adapted to in-house and commercial development processes.

As further work, we would like to study how to address the limitations previously identified, including as well usability aspects, e.g., regarding the enforcement of the rules and the their presentation (i.e. concrete syntax) to end-users. Also, we are interested in extending our language to be able to specify other kinds of complementary rules like the ones governing the team organization (e.g., how the project participants can be promoted between the different roles) or providing fallback behavior to deal with potential undecidable cases. Adding privacy concerns is also under evaluation (some projects may require anonymity in the voting phase, or keep private some discussions to all people with less privileges). Additionally, we would like to mine existing software repositories to infer and study their governance rules, e.g., to see if the extracted rules correspond to the perception of project participants. Preliminary results suggest that a clear rule definition could correlate with more successful (in terms of number of contributions) projects. We believe this is also worth investigating further. Finally, a more ambitious empirical validation of the mid/long term benefits for OSS adopting our approach is also needed.

References

[1] J. Aranda and G. Venolia, "The Secret Life of Bugs: Going Past the Errors and Omissions in Software Repositories," in *ICSE conf.*, 2009, pp. 298–308.

[2] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Trans. Soft. Eng. Method.*, vol. 11, no. 3, pp. 309–346, 2002.

[3] B. Shibuya and T. Tamai, "Understanding the Process of Participating in Open Source Communities," in *FLOSS conf.*, 2009, pp. 1–6.

[4] V. Singh, D. Nichols, and M. Twidale, "Users of Open Source Software: How do They Get Help?" in *HICSS conf.*, 2009, pp. 1–10.

[5] A. Mockus, "A Case Study of Open Source Software Development: the Apache Server," in *ICSE conf.*, 2000, pp. 263–272.

[6] D. Bertram and A. Voida, "Communication, Collaboration, and Bugs: the Social Nature of Issue Tracking in Small, Collocated Teams," in *CSCW conf.*, 2010, pp. 291–300.

[7] A. Kleppe, *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison Wesley, 2008.

[8] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.

[9] Governance Tool. `http://goo.gl/H8MMSX`.

[10] J. D. Herbsleb and R. E. Grinter, "Splitting the Organization and Integrating the Code: Conway's Law Revisited," in *ICSE conf.*, 1999, pp. 85–95.

[11] R. Kraut and L. Streeter, "Coordination in Software Development," *Comm. ACM*, vol. 28, no. 3, pp. 69–81, 1995.

[12] D. L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Comm. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[13] K. Crowston, K. Wei, Q. Li, U. Y. Eseryel, and J. Howison, " Coordination of Free/Libre Open Source Software Development," in *ICIS conf.*, 2005.

[14] M. L. Markus, "The Governance of Free/Open Source Software Projects: Monolithic, Multidimensional, or Configurational?" *J. of Manag. & Gov.*, vol. 11, no. 2, pp. 151–163, 2007.

[15] S. Chulani, C. Williams, and A. Yaeli, "Software Development Governance and Its Concerns," in *SDG conf.*, 2008, pp. 3–6.

[16] N. Ramasubbu and R. K. Balan, "Towards Governance Schemes for Distributed Software Development Projects," in *SDG conf.*, 2008, pp. 11–14.

[17] W. Van Grembergen, *Strategies for Information Technology Governance*. IGI Publishing, 2003.

[18] P. Webb, C. Pollard, and G. Ridley, "Attempting to Define IT Governance: Wisdom or Folly?" in *HICSS conf.*, 2006, pp. 194.1–.

[19] P. C. Rigby and M.-A. Storey, "Understanding Broadcast Based Peer Review on Open Source Software Projects," in *ICSE conf.*, 2011, pp. 541–550.

[20] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. V. Deursen, "Communication in Open Source Software Development Mailing Lists," in *MSR conf.*, 2013, pp. 277–286.

[21] C. Bird, D. Pattison, R. D. Souza, V. Filkov, and P. Devanbu, "Latent Social Structure in Open Source Projects Categories and Subject Descriptors," in *FSE conf.*, 2008, pp. 24–35.

[22] B. Heller and E. Marschner, "Visualizing Collaboration and Influence in the Open-Source Software Community," in *MSR conf.*, 2011, pp. 223–226.

[23] A. Kittur, S. Khamkar, P. André, and R. Kraut, "CrowdWeaver: Visually Managing Complex Crowd Work," in *CSCW conf.*, 2012, pp. 1033–1036.

[24] OMG, "Business Process Model and Notation (BPMN) Version 2.0," Tech. Rep., 2011.

[25] A. Wise, A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, and S. M. Sutton, "Using Little-JIL to Coordinate Agents in Software Engineering," in *ASE conf.*, 2000, pp. 155–163.

[26] R. Duque, M. L. Rodríguez, M. V. Hurtado, C. Bravo, and C. Rodríguez-Domínguez, "Integration of Collaboration and Interaction Analysis Mechanisms in a Concern-based Architecture for Groupware Systems," *Sci. Comp. Prog.*, vol. 77, no. 1, pp. 29–45, 2012.

[27] K. Luther, C. Fiesler, and A. Bruckman, "Redistributing Leadership in Online Creative Collaboration," in *CSCW conf.*, 2013, p. 1007.

[28] J. Gallardo, C. Bravo, M. a. Redondo, and J. de Lara, "Modeling Collaboration Protocols for Collaborative Modeling Tools: Experiences and Applications," *J. Vis. Lang. Comp.*, pp. 1–14, 2012.

[29] J. L. Cánovas Izquierdo and J. Cabot, "Enabling the Collaborative Definition of DSMLs," in *CAiSE conf.*, 2013, pp. 272–287.