A Flexible Framework to Support Collaborative Software Evolution Analysis

Marco D'Ambros Michele Lanza
REVEAL @ Faculty of Informatics – University of Lugano, Switzerland

Abstract

To understand the evolution of software researchers have developed a plethora of tools to parse, model, and analyze the history of systems. Despite their usefulness, a common downside of such tools is that their use comes with many strings attached, such as installation, data formats, usability, etc. The result is that many tools are only used by their creators, which is detrimental to cross-fertilization of research ideas and collaborative analysis.

In this paper we present the Churrasco *framework, which supports software evolution modeling, visualization and analysis through a web interface. The user provides only the URL of the SubVersion repository to be analyzed and, if available, of the corresponding bug tracking system. Churrasco processes the given data and automatically creates and stores an evolutionary model in a centralized database. This database, called* Meta-base *is connected to Churrasco through object-relational persistence. The persistency mechanism is meta-described in terms of the* EMOF *meta-meta-model and automatically generated based on any given evolutionary meta-model. In case the meta-model changes, the persistency mechanism is automatically updated.*

After providing a detailed description of Churrasco, we provide evidence, by means of an example scenario, that it allows for collaborative software evolution analysis, based on visualizations available on our analysis web portal.

1 Introduction

Software evolution analysis is concerned with the causes and the effects of software change. There are a large number of approaches, which all use different types of information about the history and the (evolving) structure of a system. The overall goal is on the one hand to perform retrospective analysis, useful for a number of maintenance activities, and on the other hand to predict the future evolution of a system.

Such analyses are intrinsically complex, because modeling the evolution of complex systems implies (1) the retrieval of data from software repositories, such as CVS or SVN, (2) the parsing of the obtained raw data in order to extract the relevant facts and to minimize the noise that such large data sets naturally exhibit, and (3) the population of models, which can come as mere text files, be handled in memory, or stored in a database. Researchers spend a large amount of time doing these tasks, while ultimately the goal is to analyze the populated models. Based on our own and other researchers' experiences we observed a number of concerns which arise when designing, developing and using software evolution analysis tools:

Modeling. Several, and largely similar, approaches have been proposed to create and populate a model of an evolving software system, considering a variety of information sources, such as the histories of software artifacts (as recorded by a versioning system), the problem reports stored in respositories such as Bugzilla [1, 11], mail archives [13], user documentation [3], *etc.* Even if these are comprehensive for modeling the evolution, they are "hard-coded" in the sense that their creators took deliberate design choices which were in accordance with their research goals. However, this has an impact on *flexibility* (*e.g.,* if the meta-model needs to be changed, what happens to all the previous work, the tools, the case studies, *etc.*?) and *extensibility* (*e.g.,* if the analysis is extended to new types of information, how does an existing meta-model need to be changed?).

Accessibility. Researchers have developed a plethora of evolution analysis tools and environments. One sad but true commonality among most of these prototypes is their limited usability, *i.e.,* often only the developers themselves know how to use them, which hinders the development and/or cross-fertilization of analysis techniques. There are some notable exceptions, such as Kenyon [1] or Moose [8], which have been used by a larger numbers of researchers over the years, but they come with many strings attached (*e.g.,* cumbersome installation procedures, un-documented data exchange formats, *etc.*). Because of this variety of models and tools, researchers investigated also ways for exchanging information about software systems, such as exchange languages and models [14, 22] for evolution analysis, approaches which however are seldom followed up because of lack of time or manpower.

Collaboration. Results of analyses and findings on software systems produced by tools are written into files and/or

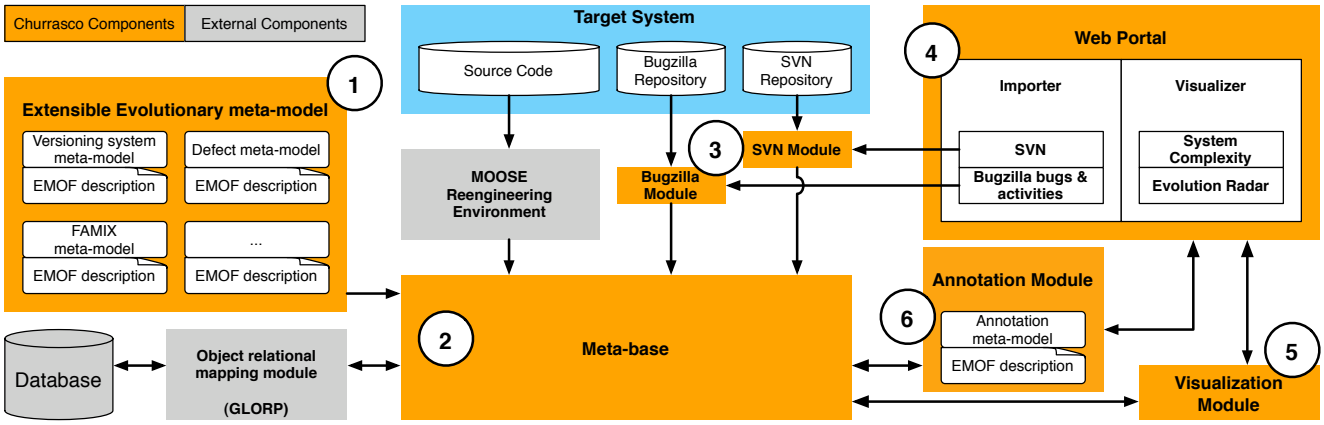


Figure 1. The architecture of the Churrasco framework.

manually crafted reports. To the knowledge of the authors, there is no approach in which such results are incrementally and consistently stored back into the analyzed models. Such incrementally enriched software models would have several advantages, as they would allow (1) other kinds of analyses to benefit from the result of a previous analysis (cross-fertilization of ideas/results), (2) different users to collaboratively analyze a system, (3) the creation of a benchmark for analyses targeting the same problem, and (4) to combine techniques targeting different problems.

In this paper we present a framework called *Churrasco*, which hides all the data retrieval and processing tasks from the users, to let them focus on the actual analysis. It provides an easily accessible interface over a web browser to model the data sources to be analyzed. It copes with the modeling and populating problems by providing a flexible and extensible object-relational persistency mechanism. Any data meta-model can be dynamically changed and extended, and all the data is stored in a centralized database, thus tackling the data exchange problem. In terms of analyses we provide an extensible set of collaborative visual analyses which are accessible over an interactive web interface, thus tackling the problem of tool installation and configuration. We support collaborative analysis by providing tools which permit the annotation of the analyzed data. The user can thus store the findings into a central DB to create an incrementally enriched body of knowledge about a subject system, that can be exploited by the subsequent Churrasco users.

Structure of the paper. In Section 2 we provide a description of the Churrasco framework, its architecture, and its main components. We then showcase, by means of an analysis scenario, collaborative and distributed software evolution analysis using Churrasco (Section 3). We discuss our approach in Section 4, survey related work in Section 5, and conclude in Section 6 with a summary of our contributions.

2 The Churrasco Framework

Figure 1 shows Churrasco's architecture in terms of its internal components, its relationship to a target system to be analyzed, and its connection to external components. In the remainder of this section we discuss each internal component in detail, namely:

1. *The Extensible Evolutionary meta-model* describes the internal representation of software systems evolution, which can be extended using the facilities provided by the Meta-base module.

2. *The Meta-base*, the core component of Churrasco, supports flexible and dynamic object-relational persistency. The Meta-base uses the external component GLORP, an object relational mapping module providing object-relational persistency, to read/write from/to the database. The meta-base also uses the Moose reengineering environment [8] to create a representation of the source code based on the FAMIX meta-model.

3. *The Bugzilla and SVN modules* retrieve and process the data from SVN and Bugzilla repositories.

4. *The Web portal* represents the front-end of the framework accessible through a web browser.

5. *The Visualization module* supports software evolution analysis by creating and exporting interactive Scalable Vector Graphics[1] (SVG) visualizations.

6. *The Annotation module* supports collaborative analysis by enriching any entity in the system with annotations. It communicates with the web visualizations to depict the annotations within the visualizations.

[1] http://www.w3.org/Graphics/SVG/

2.1 The Extensible Evolutionary Meta-Model

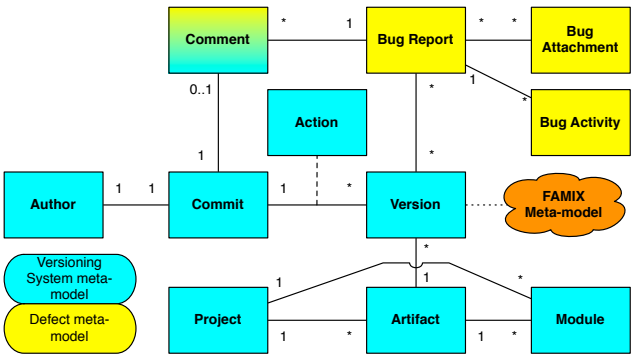


Figure 2. A subset of our extensible evolutionary meta-model.

Using a union of different meta-models (depicted in Figure 2), we currently represent system evolution in 3 ways:

1. The Versioning System meta-model models the history of software artifacts, as recorded by a versioning system: A software project is composed of a collection of software artifacts and, when the information is available, of a collection of software modules. Each module in turn is composed of software artifacts. A software artifact has a collection of versions, which compose its history. Each version has a commit performed by an author (who can enter a comment) and can be affected by defects. Each commit can be tied to a list of versions by actions such as "add", "remove", etc.

2. The Defect meta-model models bug reports as recorded by bug tracking systems such as Bugzilla. The core entity is the bug report, which internally models the problem (description, location in the system), its criticality (priority to fix and severity), the people involved (developer in charge to fix, quality assurance), the condition in which the bug was detected (operating system, platform) and the state of the bug (status, *e.g.,* new, resolved, closed *etc.* and resolution, *e.g.,* fixed, invalid, duplicate *etc.*). Each bug has a collection of attachments (*e.g.,* patches), a collection of comments about the problem and possible solutions, and a collection of software artifacts affected by it. The defect meta-model takes time into account. Every field of a bug can be modified over time thus generating a bug activity. The activity records which field is changed, when, by whom and the pair of old and new values. Activities are important because they allow us to keep track of a bug's life cycle, *i.e.,* the sequence of statuses the bug went through.

3. The FAMIX meta-model [7] models one or more versions (usually the last version, being the most relevant one) of a system in a fine-grained way.

2.2 The Meta-base

The *Meta-base* [6] is the core module of Churrasco, which provides flexibility and persistency to *any* meta-model in general, and to our evolution meta-model in particular. It takes as input a meta-model described in EMOF and outputs a descriptor, which defines the mapping between the object instances of the meta-model, *i.e.,* the model, and tables in the database. EMOF (Essential Meta Object Facilities) is a subset of MOF[2], a meta-meta-model used to describe meta-models. The Meta-base uses an implementation of EMOF called *Meta*[3]. The Meta-base ensures persistency with the object-relational module GLORP [15] (Generic Lightweight Object-Relational Persistence). The Meta-base provides flexibility by dynamically and automatically adapting to any provided meta-model, by generating descriptors of the mapping between the database and the meta-model. This allows the Churrasco users to dynamically both modify and extend the meta-model of the evolution of the system.

Example. We provide an example meta-model, to show how it has to be described to be used in the Meta-base.

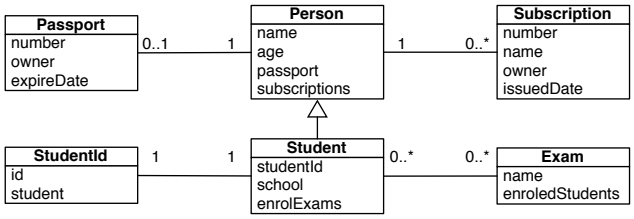


Figure 3. Our example meta-model.

Figure 3 shows the UML diagram of the example meta-model, while the code snippet below shows its EMOF description (due to lack of space we only show the part for Person and Passport).

```
Person>>metamodelAge
  ^(EMOF.Property name: #age type: Number)
Person>>metamodelName
  ^(EMOF.Property name: #name type: String)
Person>>metamodelPassport
  ^(EMOF.Property name: #passport
    opposite: #owner type: Passport)
Person>>metamodelSubscription
  ^(EMOF.Property name: #subscription opposite: #owner
     type: Subscription multiplicity: #many)
Passport>>metamodelExpireDate
  ^(EMOF.Property name: #expireDate type: Date)
Passport>>metamodelNumber
  ^(EMOF.Property name: #number type: Number)
Passport>>metamodelOwner
  ^(EMOF.Property name: #owner
     opposite: #passport type: Person)
```

[2]MOF and EMOF are standards defined by the OMG (Object Management Group) for Model Driven Engineering. For more details about MOF and EMOF consult the specifications at: http://www.omg.org/docs/html/06-01-01/Output/06-01-01.htm

[3]http://smallwiki.unibe.ch/moose/tools/meta/

Once the meta-description is defined as shown in the code snippet, we can automatically generate the object-relational descriptor, and then read and write objects instances of the meta-model from and to the database in a transparent way.

Person

dbid	school	studentid	passport	age	name	classtype
1	UZH	2	4	22	Anna Cazzulani	Student
2	USI	3	5	20	Giorgio Diegoli	Student
3	NULL	NULL	1	27	Peppe Castiglia	Person
4	NULL	NULL	6	28	Michele Vanzo	Person
5	NULL	NULL	2	31	Jhonny Bravo	Person
6	Politecnico	1	3	18	Carmelo Varicella	Student

PersonExamLink

dbid	personid	examid
1	2	2
2	1	3
3	6	4
4	2	4
5	6	1
6	1	1
7	6	2

StudentId

dbid	id	student
1	1	6
2	3	1
3	2	2

Exam

dbid	name
1	Calcolus
2	Algebra
3	Greek
4	Physics

Figure 4. The generated database.

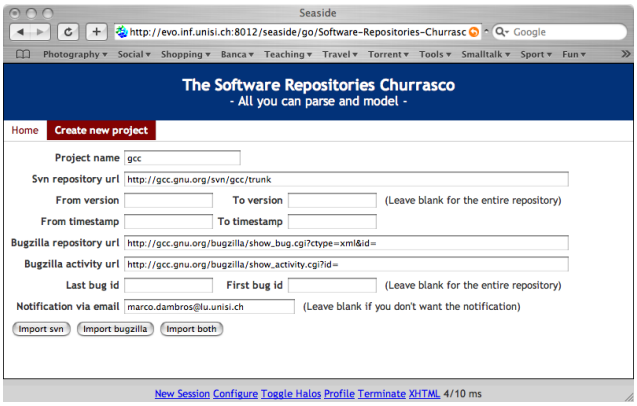
Figure 4 shows the database tables automatically generated and populated. As shown in this example, the Meta-base supports one-to-one, one-to-many and many-to-many relationships among meta-model entities. It also supports inheritance between meta-model classes by means of filtered inheritance. All of the classes are represented in a single table, with a discriminator field for which subclass they are. The table has the union of all possible fields for all classes.

2.3 The SVN and Bugzilla Modules

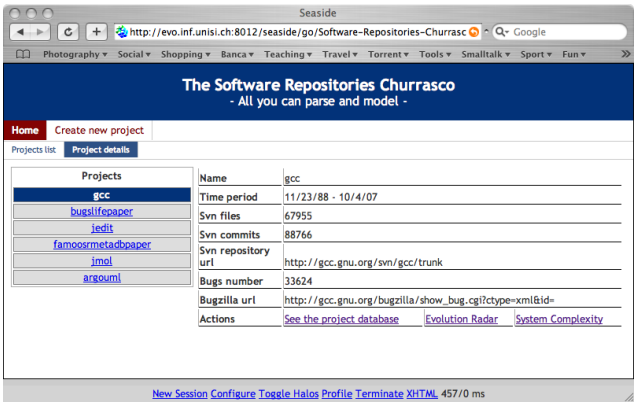
The SVN and Bugzilla modules retrieve and process data from, respectively, Subversion and Bugzilla repositories. They take as input the URL of the repositories and then populate the models using the Meta-base. The modules are initially launched from the web importer (discussed later) to create the models, and then they automatically update all the models in the database every night, with the new information (new commits or bug reports).

The SVN module populates the versioning system model, by checking out (or updating) the project with the given repository, creating and parsing SVN log files. The checked out system is then used to create the FAMIX model of the system with the external component Moose.

The Bugzilla module retrieves and parses all the bug reports (in XML format) from the given repository. Then it populates the corresponding part of the defect model. It then retrieves all the bug activities from the given repository. Since Bugzilla does not provide this information in XML format, HTML-pages have to be parsed and the corresponding part of the model is populated. Finally, it links software artifacts with bug reports. To do this it combines the technique proposed in [11] (matching bug report ids and keywords in the commit comments) with a timestamp based approach. The algorithm starts by looking into bug activities and extracting bug fixing activities, *i.e.,* activities in which the status changed to resolved, verified or closed. For this activities the algorithm takes the timestamp and the author and then it looks for a commit performed by the same author in a temporally close timestamp. The underlying idea is that the author fixes the bug, commits the changes and changes the bug report to a fixed status. For the comparison between SVN and Bugzilla authors (two different accounts denoting the same person), we use a set of heuristics.



(a) The importer page.



(b) The projects page.

Figure 5. The Churrasco Web Portal.

2.4 The Web Portal

The web portal is the front end interface provided by Churrasco. It allows users both to create the models, and to analyze them by means of two web-based visualizations.

Figure 5(a) shows the importer web page of Churrasco, ready to import the gcc[4] software system. All the infor-

[4]http://gcc.gnu.org/

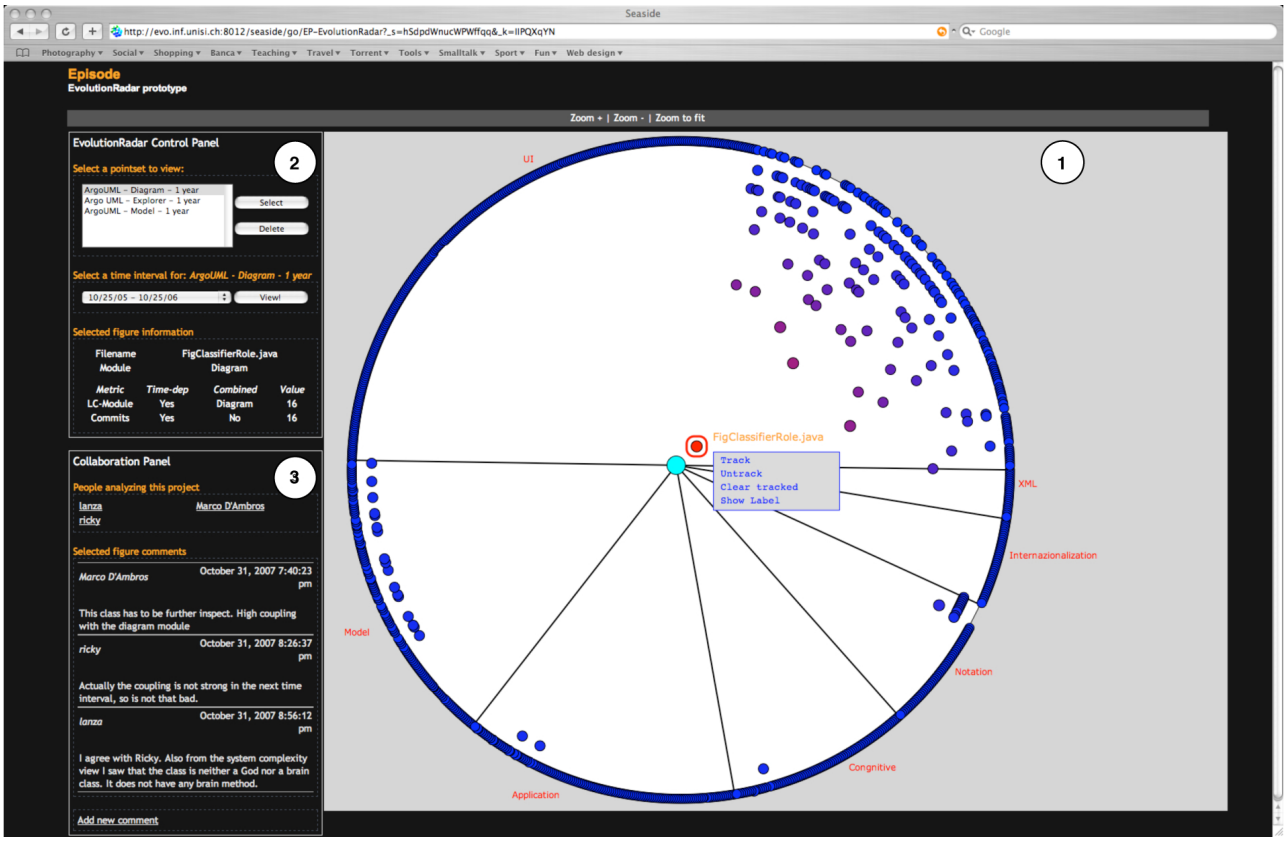


Figure 6. The web portal of Churrasco showing an interactive Evolution Radar visualization.

mation needed to create the model is the URL of the SVN repository and, if available, the URLs of the bugzilla repository (one for bug reports, one for bug activities). Since, depending on the size of the software system to be imported, this can take a long time, the user can also indicate the e-mail address to be notified when the importing is finished.

Figure 5(b) shows the projects web page of Churrasco, which contains a list of projects available in the database, and for a selected project, information such as the number of files and commits, the time period (time between the first and last commits), the number of bugs *etc.* The page provides *actions* to the user, *i.e.,* links to the visualizations and to the database contents.

Figure 6 shows one visualization web page. The page is composed of three main parts: (1) The actual view, (2) the application control panel, and (3) the collaboration panel. The view and application control panel are described in Section 2.5, the collaboration panel in Section 2.6.

2.5 The Visualization Module

The visualization module is responsible for creating interactive visualizations within the Churrasco web portal, to support the analysis of the evolution of a software system. At the moment there are two visualizations available:

The Evolution Radar [4, 5] supports software evolution analysis by depicting integrated logical coupling information, both at the file and module level. The left part of Figure 7 shows the principle of the Evolution Radar. The visualization shows the dependencies between a module, represented as a circle and placed in the center of a pie chart, and all the other modules in the system represented as sectors of the pie chart. In each sector, all the files belonging to the corresponding module are represented as colored circles and positioned according to the change coupling they have with the module in the center (the higher the coupling the closer to the center). The evolution radar visualization is created from versioning system models.

The System Complexity [16] supports the understanding of object oriented system, by enriching a simple 2D depiction of classes and inheritance relationships with software metrics (see the right part of Figure 7). The size of the nodes is proportional to the number of attributes (width) and methods (height), while the color renders the number of lines of code. This view is created using a FAMIX model.

Both visualizations are created by the visualization mod-

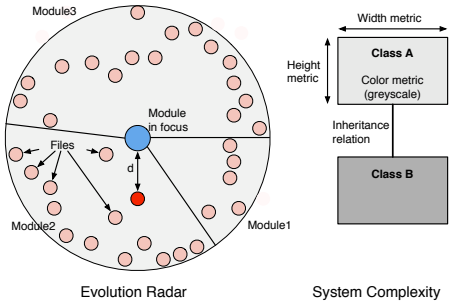


Figure 7. The principles of the Evolution Radar and System Complexity visualizations.

ule in two steps: (1) first the visualizations are generated by the Evolution Radar tool or by the Mondrian framework [18] (residing in Moose) and then (2) the web versions of them are created using the Episode framework [21] residing within Churrasco's visualization module. To make the visualizations interactive within the web portal, Episode attaches callbacks to the figures. Figure 6 shows an example of an Evolution Radar visualization rendered in the Churrasco web portal. The part marked as 1 is the view where all the figures are rendered as SVG graphics. The figures are interactive: Clicking on one of them will highlight the figure, generate a context menu (as the example in Figure 6) and show the figure details in the application control panel (marked as 2). This panel, different from visualization to visualization, provides the control (on top) to apply, modify and interact with the visualization, and shows the information (at the bottom) about the selected figure (which entity it represents and the properties of the metrics used in the view).

2.6 The Annotation Module

The Annotation module of Churrasco supports collaborative analysis: The idea is that each model entity can be enriched with annotations, and these annotations can be used (1) to store findings and results incrementally into the model and (2) to let different users collaborate in the analysis of a system in parallel.

Annotations can be attached to *any* model entity, by means of the proxy pattern [12], and each entity can have several annotations. The annotation is composed of: The author who wrote it, the creation timestamp and the text. The part marked as 3 in Figure 6, called collaboration panel, shows how the annotations are used in the Churrasco web portal. The collaboration panel, which has the same structure in all the visualizations, is composed of three parts: The first one, on top, lists all the people who annotated the visualizations, *i.e.,* people collaborating in the analysis. When one of these names is clicked, all the figures annotated by the corresponding person are highlighted in the view, to see on which part of the system that person is working on. The second part of the collaboration panel, in the middle, lists all the annotations of the selected figure, showing the author, the date and time and the text. The last part of the panel, at the very bottom, is used to create a new annotation to attach to the selected figure. Since the database behind Churrasco is centralized, when a new annotation is added it is immediately visible to all the people using the web visualization. This allows different users to simultaneously work on the same system and to collaborate in the analysis.

3 Collaborative Software Evolution Analysis

In this section we illustrate how Churrasco supports collaborative software evolution analysis. For space reasons we do not describe an entire collaborative session, but just anecdotal evidence that different, geographically distributed users collaborate, using dedicated visualization, in the analysis of a system. Our goal is not to prove the usefulness of the visualizations themselves, but to demonstrate that they can be used to do collaborative software evolution analysis.

Scenario. Two Churrasco users, *i.e.,* the authors of this article, work on different machines in different locations to study the evolution of ArgoUML (http://argouml.tigris.org/), a UML modeling tool, composed of ca. 1800 Java classes, developed over the course of ca. 7 years.

Churrasco in Action. The users first create the evolutionary model by indicating the URL of the ArgoUML SVN repository in the importer page of Churrasco[5]. Once the model is created and stored in the centralized database, they start the analysis with a system complexity view of the system. Each user renders the visualization in his web browser, and attaches annotations to interesting figures in the visualization. The annotations are immediately visible to the other user on the left side of the browser window.

While Michele is analyzing the entire system, Marco focuses (creating an ad-hoc view) on the `Model` namespace, which contains several classes characterized by large number of methods and many lines of code. The entities annotated by Marco in the fine-grained view are then visible to Michele in the coarse grained system complexity. Marco has the advantage of a more focused view, while Michele can see the entire context. Figure 8 shows a screenshot of Marco's web view on the left, while Michele's view is depicted on the right (only the view, without the entire web page for space reasons). Marco selected the `FacadeMDRImpl` class (marked as 1), and is reading Michele's comments about that class (marked as 2 in Michele's view). In the meantime, Michele highlighted all the figures annotated by Marco (marked as 2, 3, 4). We have two examples of collaboration:

[5]Bug information is not needed in this example scenario.

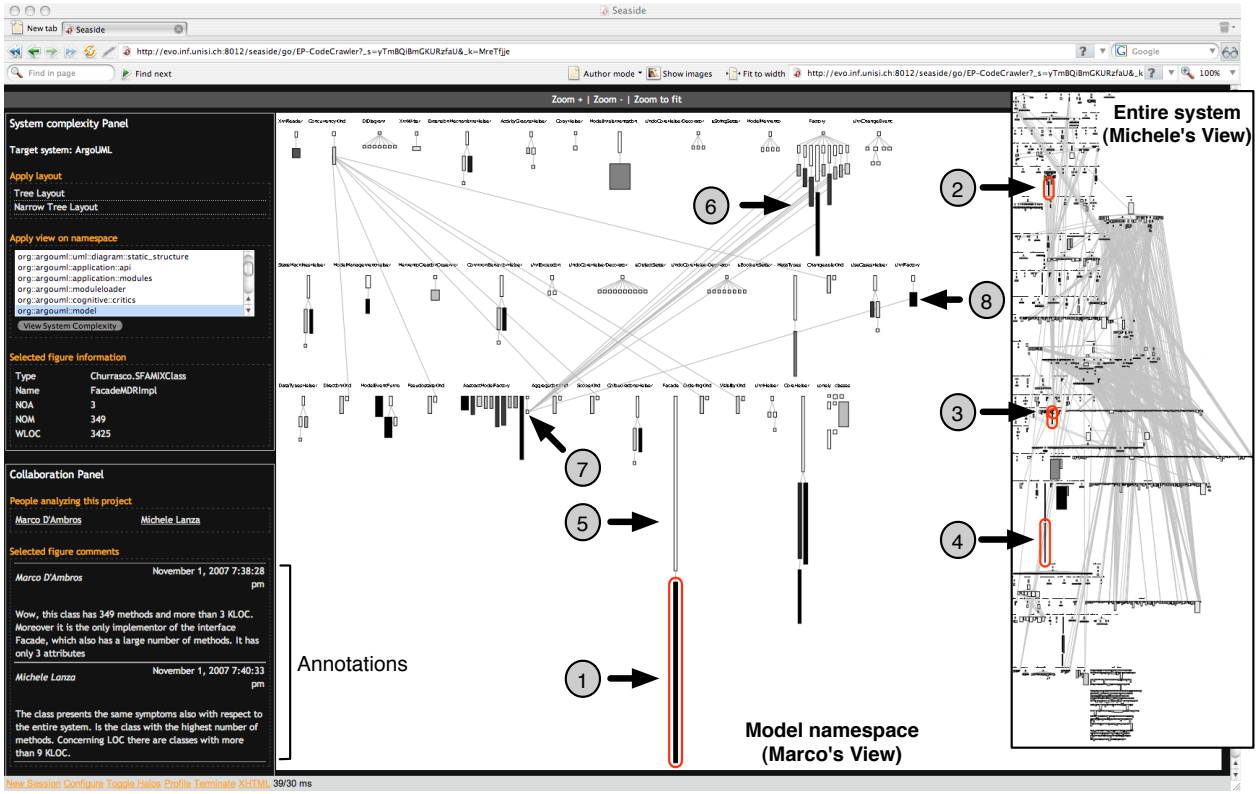


Figure 8. The web portal of Churrasco visualizing two system complexities of ArgoUML: For the entire system (right) and for the `Model` namespace (left).

1. Marco, focusing on the `Model` namespace, annotates that the class `FacadeMDRImpl` (marked as 1) shows symptoms of bad design: it has 350 methods, 3400 lines of code, only 3 attributes, and it is the only implementor of the `Facade` interface (marked as 5). Michele adds a second annotation that Marco's observation holds also with respect to the entire system, and that `FacadeMDRImpl` (marked as 4) is the class with the highest number of methods in the entire system.

2. Marco sees that several classes in the `Factory` hierarchy (mark as 6) implement the `Factory` interface and also inherit from the `AbstractUmlModelFactoryMDR` class (marked as 7) belonging to another hierarchy. This is not visible in Michele's large scale view (where `Factory` is marked as 2 and `AbstractUmlModelFactoryMDR` as 3). Michele discovers that fact by highlighting the entities annotated by Marco and then reading the annotations.

Both now want to find out whether these design problems have always been present in the system. They analyze the system history in terms of its logical coupling using the Evolution Radar. This visualization is time-dependent, *i.e.,* different radar views are used to represent different time intervals. Figure 9 shows two evolution radar visualizations: The large one on the left corresponds to the time interval October 2004 – October 2005, while the small one on the right (without the left panels for space reasons) corresponds to October 2006 – October 2007. They both represent the dependencies of the `Diagram` module (in the center) with all the other modules in ArgoUML, by rendering individual classes. Marco is looking at the time interval 2004/05 (left part of Figure 9). He selects the class `UMLFactoryImpl` (marked as 1), belonging to the `Model` module, because it is close to the center (high coupling with the `Diagram` module in the center) and because it is large (the size maps the number of commits in the corresponding time interval). Marco attaches to the class the annotation that it is potentially harmful[6], given the high coupling with a different module. In the meantime Michele is looking at the time interval 2006/07 (right part of Figure 9). He highlights the classes annotated by Marco and sees the `UMLFactoryImpl` class. In Michele's radar the

[6]See [4] for a detailed description of the Evolution Radar.

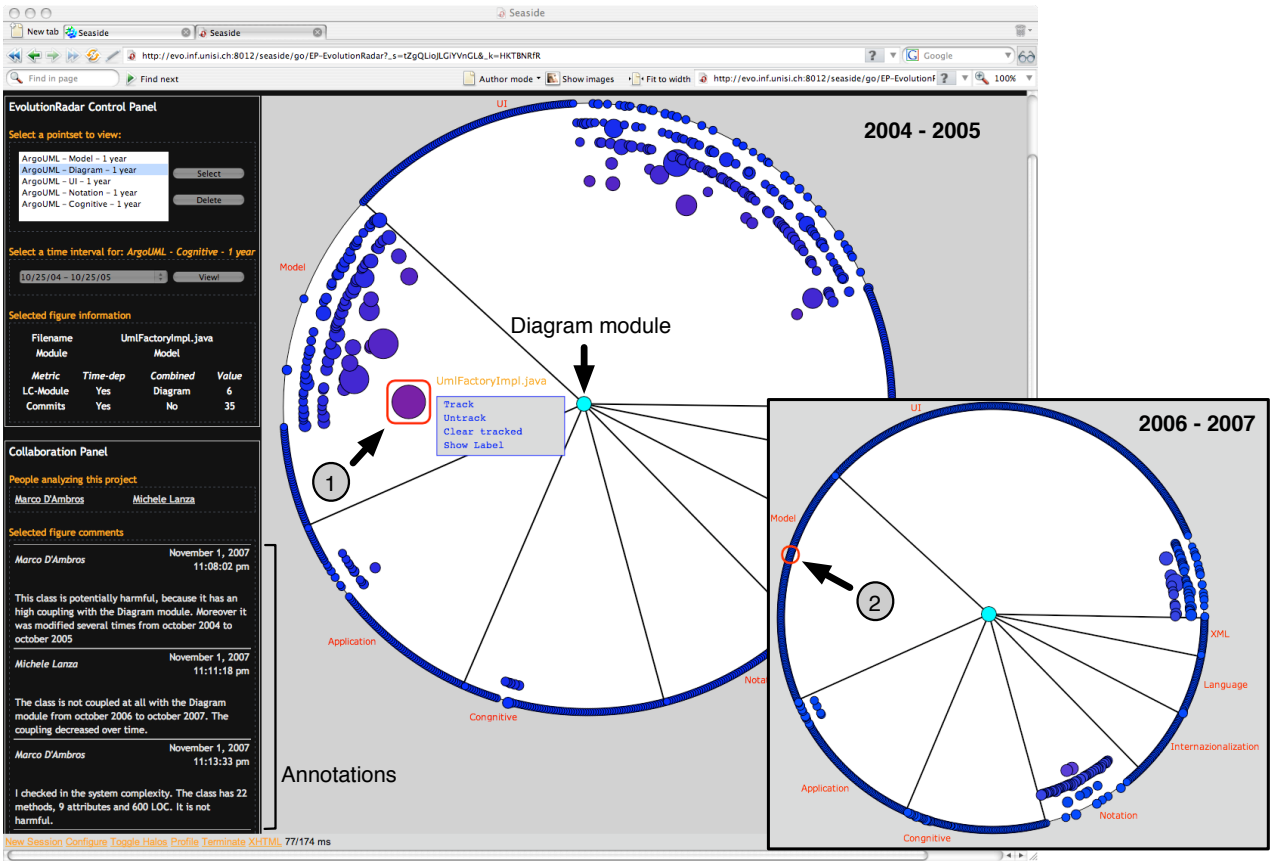


Figure 9. The web portal of Churrasco visualizing two Evolution Radars of ArgoUML: 2004-2005 (left) and 2006-2007 (right).

class is not coupled at all with the `Diagram` module, *i.e.,* it is at the boundary of the view (marked as 2). Therefore, Michele adds an annotation to the class saying that it is probably not harmful, since the coupling decreased over time. After reading this comment, Marco goes back to the system complexity view, to see the structural property of the class in the system. The `UMLFactoryImpl` class (marked as 8 in Figure 8) has 22 methods, 9 attributes and 600 LOC. It implements the interfaces `AbstractUmlModelFactoryMDR` and `UMLFactory`. After seeing the class in the system complexity, Marco adds another annotation in the radar saying that the class is not harmful after all. This information can then be used by other users in the future. Suppose that Romain wants to join the analysis with Marco and Michele, or to start from their results. He can first see on which entities the previous users worked, by highlighting them, and then reading the corresponding annotations to get the previously acquired knowledge about the system.

This simple scenario shows: (1) how the knowledge about a system, gained in software evolution analysis activities, can be incrementally built, (2) how different users from different locations can collaborate, and (3) how different visualization techniques can be combined to improve the analysis.

4 Discussion

The main benefits of Churrasco consist in its accessibility and flexibility. All the features of the framework can be accessed by a web browser: (1) The importers to create and populate evolutionary models of software systems, (2) the system complexity visualization, to support the understanding of the structure of the system and (3) the evolution radar view to study the evolution of the system modules in terms of logical coupling. The visualizations are interactive, and they allow the user to inspect the entities represented by the figures, to apply new visualizations on-the-fly from the context menus and to navigate back and forth among different views. The framework can be extended with respect to the meta-model and with respect to the visualizations. Using the facilities provided by the meta-base, the underlying evolu-

tionary meta-model of Churrasco can be enriched with new types of information. Besides the system complexity and the evolution radar, other visualizations, based on versioning systems, bug tracking systems or FAMIX data, can be ported to the web and included in the framework.

The importer part of Churrasco has been validated with both medium (*e.g.,* ArgoUML with 300 KLOC) and large scale (*e.g.,* gcc with 3.4 MLOC) systems, while the two visualizations have been validated only on medium size systems, *i.e.,* their scalability is not proven yet. The collaboration part also needs to be tested with larger numbers of participants.

Concerning interoperability Churrasco can export models of software systems in different formats (*e.g.,* XML, database dump). However, interoperability can be improved by (1) exporting the models in standard interchange formats such as TA-RE [14] and (2) exporting the annotations so that they can be imported and used by other software evolution tools.

5 Related Work

Two approaches similar to Churrasco are Kenyon [1] by Bevan *et al.* and the Release History Database (RHDB) [11], by Fischer *et al.* The Kenyon framework provides an extensible infrastructure to retrieve the history of a software project from a SCM repository or a set of releases, and to process the retrieved information. It also provides a common interface based on Object-Relational persistency to access the processed data stored in a database to perform software evolution analysis. Kenyon retrieves information from a number of SCM systems, such as CVS, SVN, ClearCase, *etc.*, but without considering other sources, such as bug tracking systems or mail archives. The RHDB was the first approach to link software artifacts with bug reports. The data is retrieved from versioning systems and bug tracking systems, processed and stored in a database for later analysis. A number of techniques were proposed on top of the RHDB: In [9] Fischer *et al.* used multidimensional scaling to visualize the evolution of features, with the aim of uncover hidden dependencies between software features. Pinzger *et al.* in [20] proposed a visualization technique, based on Kiviat diagrams, to provide integrated views on source code metrics across different releases together with logical coupling information computed from CVS log files. The EvoGraph visualization approach [10] combines release history data and source code changes to assess structural stability and recurring modifications.

Other two approaches which combine and link information from versioning systems and bug tracking systems, but which also use other sources of information are Hipikat [23] by Čubranić *et al.* and softChange [13] by German. Both techniques use information from mail archives and, in addition, Hipikat also considers data from documentation. The information stored by Hipikat forms an "implicit group memory" (group of developers) which is then used to facilitate the insertion of newcomers in the group, by recommending relevant artifacts for specific tasks. The data retrieved and processed by softChange is used for two types of software evolution analysis: (1) Statistics of the overall evolution of the project, and (2) analysis of the relationships among files and authors. These approaches rely on dedicated tools for both the data retrieval and processing tasks and for the subsequent visualization and analysis. In Churrasco all these activities can be done with just a web browser. Moreover, these approaches are based on "hard coded" meta-models, while Churrasco provides flexibility and extensibility to the evolutionary meta-model, by means of the meta-base.

A number of approaches support web-based software evolution analysis and visualizations. Beyer and Hassan proposed Evolution Storyboards [2], a visualization technique that offers dynamic views. The storyboards, rendered as SVG files (visible in a web browser), emphasizes the history of a project using a sequence of panels, each representing a particular time period in the life of a software project. These visualizations are not, or only partially, interactive, *i.e.,* they only show the names of the entities represented by the SVG or VRML figures. In contrast the views offered in the Churrasco web portal are fully interactive, providing context menus for the figures and navigation capabilities. In [17] the authors presented REportal, a web-based portal site for the reverse engineering of software systems. REportal allows users to upload their code (Java or C++) and then to browse, analyze and query it. These services are implemented by reverse engineering tools developed by the authors over the years. Reportal supports software analysis through browsing and querying, whereas Churrasco supports the analysis by means of interactive visualizations. In [19] Nentwich *et al.* introduced BOX, a portable, distributed and interoperable approach to browse UML models. BOX translates a UML model that is represented in XMI into VML (Vector Markup Language), which can be directly displayed in a web browser. BOX enables software engineers to access and review UML models without the need to purchase licenses of tools that produced the models. While BOX is focused on design documents, such as UML diagrams, in Churrasco we focus on the history of software systems.

A major difference between all the mentioned approaches and Churrasco is that these techniques support single user software evolution analysis, while Churrasco supports collaborative analysis.

6 Conclusions

In this paper we presented Churrasco, a novel framework to support collaborative software evolution analysis and visualization. The main features of the framework, and their relevance for software evolution analysis, are:

Flexible and extensible meta-model support. The meta-model used in Churrasco to describe the evolution of a software system can be dynamically changed and/or extended, by means of the meta-base component.

Accessibility. The framework is fully web-based, *i.e.,* the entire analysis of a software system, from the initial model creation to the final study, can be performed from a web browser, without having to install or configure any tool.

Collaboration. Churrasco relies on a centralized database and supports annotations. Thus, the knowledge of the system, gained during the analysis, can be incrementally stored on the model of the system itself. We have shown, through a simple, but real, scenario, how Churrasco supports collaborative software evolution analysis.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "DiCoSA - Distributed Collaborative Software Analysis" (SNF Project No. 118063) and the European Smalltalk User Group (`http://www.esug.org`). We thank Damien Pollet for his feedback on drafts of this paper.

References

[1] J. Bevan, J. E. James Whitehead, S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. In *Proceedings of ESEC/FSE 2005*, pages 177–186, New York, NY, USA, 2005. ACM.

[2] D. Beyer and A. E. Hassan. Animated visualization of software history using evolution storyboards. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 199–210, Washington, DC, USA, 2006. IEEE Computer Society.

[3] D. Cubranic and G. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings 25th International Conference on Software Engineering (ICSE 2003)*, pages 408–418, New York NY, 2003. ACM Press.

[4] M. D'Ambros and M. Lanza. Reverse engineering with logical coupling. In *Proceedings of WCRE 2006 (13th Working Conference on Reverse Engineering)*, pages 189–198. IEEE CS Press, 2006.

[5] M. D'Ambros, M. Lanza, and M. Lungu. The evolution radar: Visualizing integrated logical coupling information. In *Proceedings of MSR 2006 (3rd International Workshop on Mining Software Repositories)*, pages 26–32, 2006.

[6] M. D'Ambros, M. Lanza, and M. Pinzger. The metabase: Generating object persistency using meta descriptions. In *Proceedings of FAMOOSR 2007 (1st Workshop on FAMIX and Moose in Reengineering)*, 2007.

[7] S. Demeyer, S. Tichelaar, and S. Ducasse. FAMIX 2.1 — The FAMOOS Information Exchange Model. Technical report, University of Bern, 2001.

[8] S. Ducasse, T. Gîrba, and O. Nierstrasz. Moose: an agile reengineering environment. In *Proceedings of ESEC/FSE 2005*, pages 99–102, Sept. 2005. Tool demo.

[9] M. Fischer and H. Gall. Visualizing feature evolution of large-scale software based on problem and modification report data. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):385–403, 2004.

[10] M. Fischer and H. C. Gall. Evograph: A lightweight approach to evolutionary and structural analysis of large software systems. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE)*, pages 179–188, Benevento, Italy, October 2006. IEEE Computer Society.

[11] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings International Conference on Software Maintenance (ICSM 2003)*, pages 23–32, Los Alamitos CA, Sept. 2003. IEEE Computer Society Press.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, Mass., 1995.

[13] D. M. German, A. Hindle, and N. Jordan. Visualizing the evolution of software using softchange. In *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE 2004)*, pages 336–341, New York NY, 2004. ACM Press.

[14] S. Kim, T. Zimmermann, M. Kim, A. Hassan, A. Mockus, T. Gîrba, M. Pinzger, J. Whitehead, and A. Zeller. TA-RE: An exchange language for mining software repositories. In *Proceedings Workshop on Mining Software Repositories (MSR 2006)*, pages 22–25, 2006.

[15] A. Knight. Glorp: generic lightweight object-relational persistence. In *OOPSLA '00: Addendum to the 2000 proceedings of the conference on Object-oriented programming, systems, languages, and applications (Addendum)*, pages 173–174, New York, NY, USA, 2000. ACM Press.

[16] M. Lanza and S. Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *Transactions on Software Engineering (TSE)*, 29(9):782–795, Sept. 2003.

[17] S. Mancoridis, T. S. Souder, Y.-F. Chen, E. R. Gansner, and J. L. Korn. Reportal: A web-based portal site for reverse engineering. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE 2001)*, page 221, Washington, DC, USA, 2001. IEEE Computer Society.

[18] M. Meyer, T. Gîrba, and M. Lungu. Mondrian: An agile visualization framework. In *ACM Symposium on Software Visualization (SoftVis 2006)*, pages 135–144, New York, NY, USA, 2006. ACM Press.

[19] C. Nentwich, W. Emmerich, A. Finkelstein, and A. Zisman. BOX: Browsing objects in XML. *Software Practice and Experience*, 30(15):1661–1676, 2000.

[20] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing multiple evolution metrics. In *Proceedings of SoftVis 2005 (2nd ACM Symposium on Software Visualization)*, pages 67–75, 2005.

[21] M. Primi. The episode framework - exporting visualization tools to the web. Bachelor's thesis, University of Lugano, June 2007.

[22] S. Tichelaar, S. Ducasse, and S. Demeyer. FAMIX: Exchange experiences with CDIF and XMI. In *Proceedings of the ICSE 2000 Workshop on Standard Exchange Format (WoSEF 2000)*, June 2000.

[23] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Transactions on Software Engineering*, 31(6):446–465, 2005.