A Version Control Tool for Framework-based Applications

Maria Istela Cagnin[1] *, Rosana T. V. Braga[1], Rosângela Penteado[2],
Fernão Germano[1], José Carlos Maldonado[1]

[1]University of São Paulo
Instituto de Ciências Matemáticas e de Computacão
São Carlos, São Paulo, Brasil
Caixa Postal 668, CEP 13560-970
{istela, rtvb, fernao, jcmaldon}@icmc.usp.br

[2]Federal University of São Carlos
Departamento de Computacão
São Carlos, São Paulo, Brasil
Caixa Postal 676, CEP 13.565-905
rosangel@dc.ufscar.br

Abstract

Framework based application development is increasingly being adopted by software organizations. Frameworks provide reuse of both software design and code, and supply more trustable applications, as the components used to implement them have been previously tested. However, version control is more problematic than in conventional software development, as it is necessary to control both the framework versions and the versions of the applications created with it. Furthermore, aiming to minimize the impact of system requirement changes, framework based software development and reengineering processes adopt the incremental approach, which is a "must" in agile methodologies. This approach makes easier to fulfill requests for system requirements change at any time during the process application. In that context, there is a lack of tools that support version control of applications created with frameworks. This paper presents a tool that aims to aid in the fulfillment of that need, contributing to quality assurance of the products that result from software development or reengineering.

1 Introduction

Software evolves constantly to attend changes in management, functions, business, government rules, among others, so that the fulfillment of user needs is always kept. To control such changes, it is necessary to establish an efficient and systematic control of the versions produced and delivered to users. This can be accomplished through *Software Configuration Management - SCM*, which is a set of activities that aims to manage changes occurred during all software life cycle, and also helping to guarantee its quality (Pressman, 2001). Software organizations are increasing their investments on development of

*Financial Support from FAPESP #00/10881–4.

framework based applications (Taligent, 1997; Fayad and Schmidt, 1997). A framework is a set of classes that embeds an abstract and reusable design of solutions for a family of related problems in a particular domain (Johnson and Foote, 1988). They allow the reuse of both design and code, and the resulting applications are more reliable, because the components used to build the applications were carefully tested and were probably used before by other organizations. Moreover, Fayad and Schmidt (1997) state that, using frameworks, applications can be developed faster and with less effort.

However, version control is more complex in the context of framework based development than it is in the context of traditional software development, due to the need of controlling not only the framework versions but also of the applications generated from it. Frameworks version control has to be done with special care, because framework evolutions can change its design and, consequently, the design of dependent applications. As a result, these applications might not fit the new design and behave improperly.

To minimize the impact of changes in system requirements, some development and reengineering processes, based on frameworks, adopt the incremental approach, which is indispensable to agile methodologies (Abrahamsson et al., 2002; Beck, 2000; Turk et al., 2002). This approach eases the fulfillment of requests to change system requirements, done anytime during the process application. In this context, there is a lack of tools to support version control of both frameworks and applications derived from them. Framework instantiation allows the inclusion or removal of components in an application that was created before with the same framework. However, in case the application had been modified manually to include new functions that were not provided by the framework, if the framework is instantiated again to add new functions, all the manually included source code will be lost.

This paper presents a tool, which is specific to control the versions of applications generated from a particular framework. Its goal is to provide means of soften this lack, as well as to support the execution of the task *version control* during the SCM acitivity, and helping to guarantee the resulting product quality, both during forward engineering and reengineering. Related work is discussed in Section 2. The mentioned tool, named *GREN-WizardVersionControl*, is presented in Section 3, together with details of its conceptual modeling, architecture, and implementation. In Section 4, the experience of using this tool in a reengineering case study is presented. In Section 5, conclusions and future work is discussed.

2 Related Work

SCM is considered as one of the most important activities to obtain quality certification in several standards, such as ISO 9000 (ISO 9000, 1993), *Capability Maturity Model* (CMM) (Paulk, 1993), *Capability Maturity Model Integration* (CMMI) (Carnegie Mellon University, 2002) and ISO/IEC 15504 (*Software Process Improvement and Capability dEtermination*) or SPICE (ISO/IEC 15504, 1998). According to Pressman (2001), SCM is composed of five tasks: (1) changes identification, (2) version control, (3) changes control, (4) configuration auditing, and (5) reports (to inform what happened, who did the changes, when the change was done, what will be affected with the change, etc). The *version control* task is the most known and, at the same time, the one that has more responsibilities, because it deals with the storage and retrieval of different versions of the artifacts generated during the software process.

Some desirable features of an SCM system are highlighted by Midha (1997): a) usability, so that the user can use the tool functionality as part of his/her job execution; b) easiness of managing the tool; c) support for distributed development, so that the system can be also used by remotely distributed development teams; e d) SCM functionality integragrion with other tools. To minimize repository storage space for the artifact versions, SCM systems often use **delta** *scripts*, or simply **deltas**, in which only one artifact version is stored integrally, while the other versions store only the difference (Reichenberger, 1991). There are two approaches that deal with deltas: **negative delta** and **positive delta**. The former, also known as **reverse delta**, stores integrally the most recent version and the differences until then, so that the last version is available in a faster way. The **positive delta** stores integrally the oldest version

and the differences since then. The tool presented in this paper stores only the modifications done in each application version, because there is a framework instantiation historic that is considered by the tool, in which it is possible to obtain the functionality of the generated applications. Thus, it is possible to automatically obtain information of the applications oldest version.

There are several SCM tools, with varying functionality, complexity, and price. This requires the software engineer to evaluate each one to choose the most appropriate to his/her needs. *ClearCase* (IBM, 2004), *Continuus/CM* (Telelogic, 2004), *Visual Source Safe* (Microsoft, 2004), *QVCS* (Quma Software Inc., 2004), *FreeVCS* (FreeVCS, 2004), *CVS* (CVS, 2004), *VersionWeb* (Soares et al., 2000) and *PVCS* (Synergex, 2004) are examples of existing SCM tools. None of them fulfills the need of controlling the several versions of applications generated from frameworks. However, a tool was found, proposed by Tourwé (Tourwé, 2002), to evaluate the impact that changes can have, both in the framework and in the applications class hierarchy. To achieve that: a) it first provides the definition of changes propagation, allowing the developer to evaluate the impact of transformations in the framework and depending applications, to detect possible "*merge*" conflicts; b) it suggests how these problems can be solved. Tourwé's tool has different goals than the tool presented in this paper, which aims at controlling the changes caused in frameworks and corresponding derived applications.

3 *GREN-WizardVersionControl* tool

The motivation to create the *GREN-WizardVersionControl* tool became evident after two reengineering case studies: the first for a library legacy system (Cagnin et al., 2003a; Chan et al., 2003), and the second for a electronic appliances repair shop legacy system (Cagnin et al., 2003b), both developed in Clipper. The reengineering of these systems was done with the support of the PARFAIT [1] agile reengineering process(Cagnin et al., 2003b).

PARFAIT is incremental and iterative, as the software engineer has the flexibility to return to previously executed steps to refine the produced artifacts. It is considered an agile process, because it provides several *practices* of agile methodologies (Beck, 2000), for example: a) a new version of the system is delivered as soon as possible – attends the *"small releases" XP (eXtreme Programming) practice*; b) users participate actively of most reegineering project activities and approve the product as the design evolves. In each interaction with users, product improvements are done – attends the *"customer is present" XP practice*; c) tests in the new system are done frequently and compared to the legacy system tests – attends the *"frequent tests" XP practice*; d) reengineering planning is done again in each process iteration – attends the *"planning game" XP practice*; and e) stimulates the *"pair programming" XP practice*.

PARFAIT uses the GREN framework (from Portuguese "Gestão de REcursos de Negócios", which means Business Resource Management) (Braga, 2003) as a computational support: a) to the reverse engineering activities: during new requirements elicitation and during the identification of legacy system business rules; and b) to the forward engineering activities to create the new system prototype. GREN construction was based on the analysis pattern language for the same domain, named GRN (Braga et al., 1999), so it is specific to ease the reengineering of legacy systems belonging to the business resource management domain, which includes applications for trading, renting or maintaining business resources. The framework was built using the Smalltalk programming language and the persistence of objects is done using the MySQL database management system (MySQL, 2003). There is a tool to ease GREN instantiation, named GREN-Wizard (Braga and Masiero, 2003), which requests the GRN patterns used to model the system and generates the classes of the application (in Smalltalk), together with the MySQL tables needed to persist the objects.

[1] PARFAIT is the acronym for Processo Ágil de Reengenharia baseado em FrAmework no domínio de sistemas de Informação com VV&T (in Portuguese), which means "Framework-based Agile Reengineering Process in the Information System Domain with VV&T".

After creating the library system, during the reengineering supported by GREN-Wizard, some changes were done in its source code to make it functionally compatible with the legacy system. Afterwards, a new functionality was requested by the customer, which was not present in the legacy system, which was: *"Charge a Fine"* during the book devolution if the user returns the book after the due date. This functionality is supported by one of GRN patterns and, so, is provided by the GREN framework, without the need to implement it manually. This could be as simple as creating the application again using GREN to add the required functionally. However, doing that would cause the loss of the source code lines included manually in the application in the previous maintenance. This problem has motivated the need to create a tool to store the changes done in the applications source code, so that all changes manually done in applications generated by GREN could be retrieved and incorporated in later instantiations. Besides making evident the need to create the version control tool for applications created from the GREN framework, it was also observed the need to evolve the framework instantiation tool (GREN-Wizard), so that it could incorporate, during the creation of the new application version, the changes done in the source code in previous maintenances. This allows GREN to effectively support incremental development and reengineering, making possible to add new functions at any time during the application of these processes.

Thus, the tool was created so that agility is not injured during the PARFAIT reengineering process, as it uses the incremental approach during both the reverse engineering and the forward engineering activities. The tool is useful not only in PARFAIT but also in the development of new applications in an incremental way.

3.1 Conceptual Modeling

This section presents the *GREN-WizardVersionControl* tool functionality and metamodel, through a use case diagram and a class diagram respectively. Both are UML (*Unified Modelling Language*) diagrams (Fowler and Scott, 1997; OMG, 2003).

3.1.1 Tool Functionality

The *GREN-WizardVersionControl* tool use case diagram is shown in Figure 1. All use cases, except for that filled with grey color ("Decide about conflicts between methods"), are performed during the modifications of the source code in applications generated from the framework. The "Decide about conflicts between methods" use case is performed during the framework instantiation with the support of the updated GREN-Wizard tool to make part of the *GREN-WizardVersionControl* tool, specifically when some method inherited from a framework class had been manually modified by the application engineer. The "Change Attribute" use case is performed when the application developer needs to add basic attributes (`integer, string, float, date`) that are not inherited from the framework. Besides adding basic attributes, it is possible to add attributes with special treatment: enumerated attributes ("Add enumerated attribute") and multivalued attributes ("Add multivalued attribute").

"Remove class", "Remove attribute", "Remove method category" and "Remove method" use cases are executed only for classes, attributes, method categories, and methods not inherited from the framework, respectively. Removal restrictions were established to avoid the accidental or intentional removal of components (i.e., classes, attributes, methods, etc) inherited from the framework. So, it is only allowed for the application engineer to override or modify the inherited methods, to ensure that none of the framework *hot-spots* [2] are damaged, and that the application structure is correct and works properly.

[2] variable framework structures that contain the components that can be adapted and extended by the software engineer in the application instantiated from the framework, so as to adapt them to the needs of a particular system (for example, business rules, functional requirements inherent to the internal organization policies, etc).

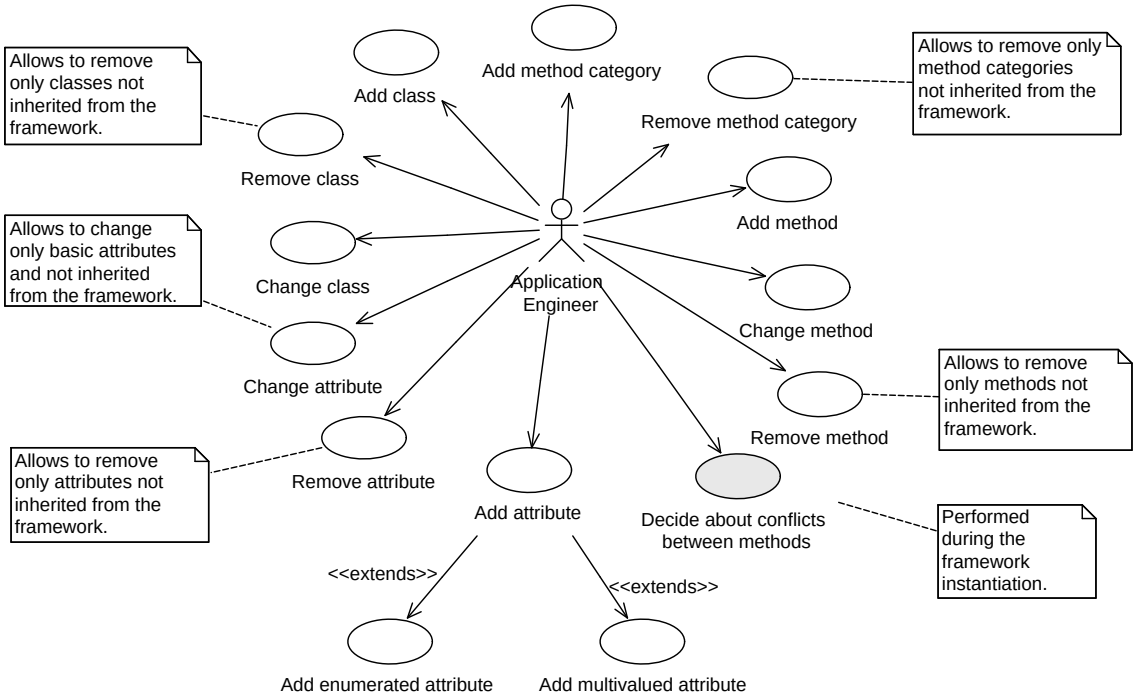

Figure 1: Use Case Diagram for the *GREN-WizardVersionControl* tool

3.1.2 Tool Metamodel

The *GREN-WizardVersionControl* tool metamodel is presented through a class diagram, as illustrated in Figure 2. Some existing classes from GREN (`PersistentObject`) and GREN-Wizard (`GrenWizardCodeGenerator`, `GrenWizardGUI`, `GrenApplication`, `AttributeMappingForm`, and `AttributeListForm`) had to be modified, so they are shown in grey. Classes: `ClassVersionControl`, `ClassProtocolVersionControl`, and `ClassMethodVersion` contain relevant information about the class being changed, about the method protocol or category [3], and about the method, respectively. As these classes contain some common information, they inherit from `MetaVersionControl`. The `GREN-WizardVersionControl` class is responsible for the version control as a whole and contains an interface similar to the *System Browser*, which is one of the VisualWorks (Cincom, 2003) tools used to develop our tool. The `GrenWizardDifferatorTool` class [4] is responsible for showing conflicting parts of the source code for methods inherited from GREN that were modified in a previous application version. These conflicts occur during the instantiation of a new application version through the updated GREN-Wizard tool and need to be taken care of by the application engineer.

[3] in Smalltalk, class methods are organized in protocols (or categories), which are groups of semantically related methods.
[4] based on VisualWorks `Differator` class.

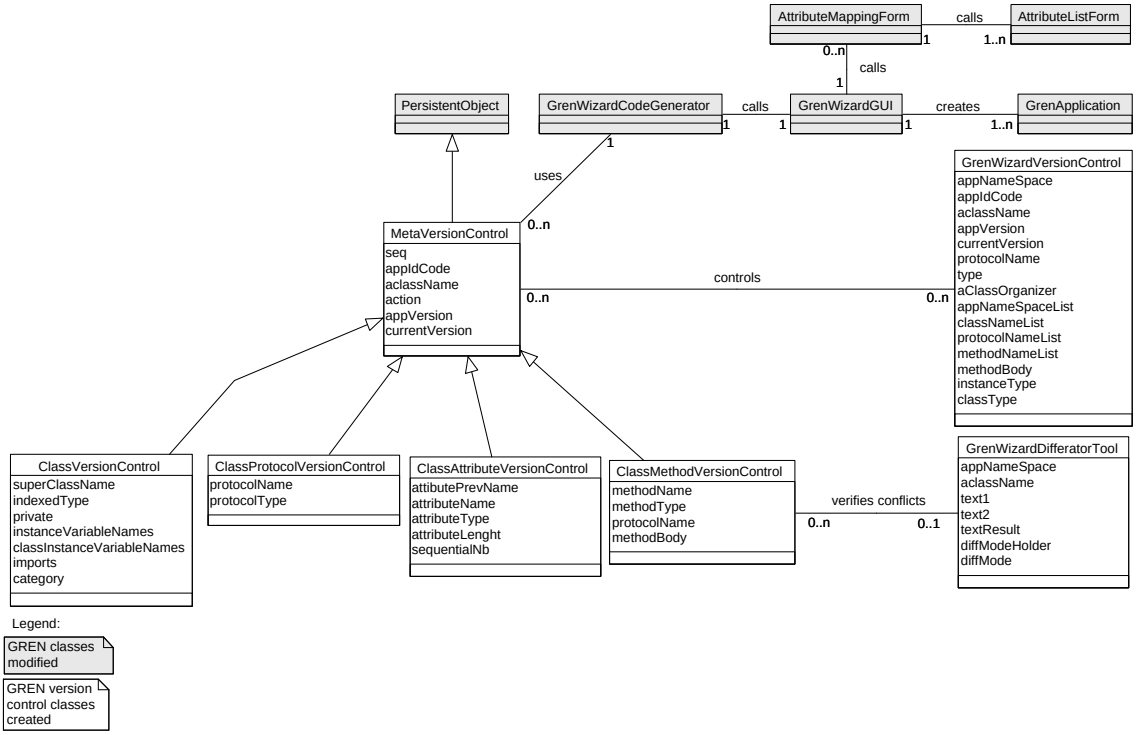

Figure 2: Class Diagram for the *GREN-WizardVersionControl* tool

3.2 Architecture

The *GREN-WizardVersionControl* tool architecture was based both on GREN and GREN-Wizard architectures. GREN architecture was designed in three layers: persistence, business and graphical user interface (GUI), as presented in Figure 3. The **persistence layer** has classes to deal with database connection, management of objects identifiers and objects persistence. The **business layer** communicates with the persistence layer to store objects. In this layer there are several classes derived directly from the analysis patterns that compose the GRN pattern language, i.e., the classes and associations contained in each pattern have the corresponding implementation in this layer. The **GUI layer** contains classes to deal with data input and output, allowing the interaction with the system final user. This layer communicates with the business layer to obtain the objects, to be shown in the GUI and to return information to be processed by the business layer methods.

The GREN-Wizard layer, responsible for generating the application source code from its specification based on GRN patterns, stays above GREN, as it uses all the other layers. The GREN-Wizard layer communicates with the *GREN-WizardVersionControl* layer, which is responsible for controlling the versions of applications generated from the GREN framework. It supports incremental development and reengineering and, consequently, processes based on agile methodologies. Specific applications can be built from its GUI layer, without the support of the GREN-Wizard layer, by using (through inheritance or object referencing) classes of all GREN layers below it, or can be built based on the business layer, if the GUI layer is not reused from GREN, but implemented separately.

Figure 4 shows GREN-Wizard and *GREN-WizardVersionControl* architectures. Two actors interact

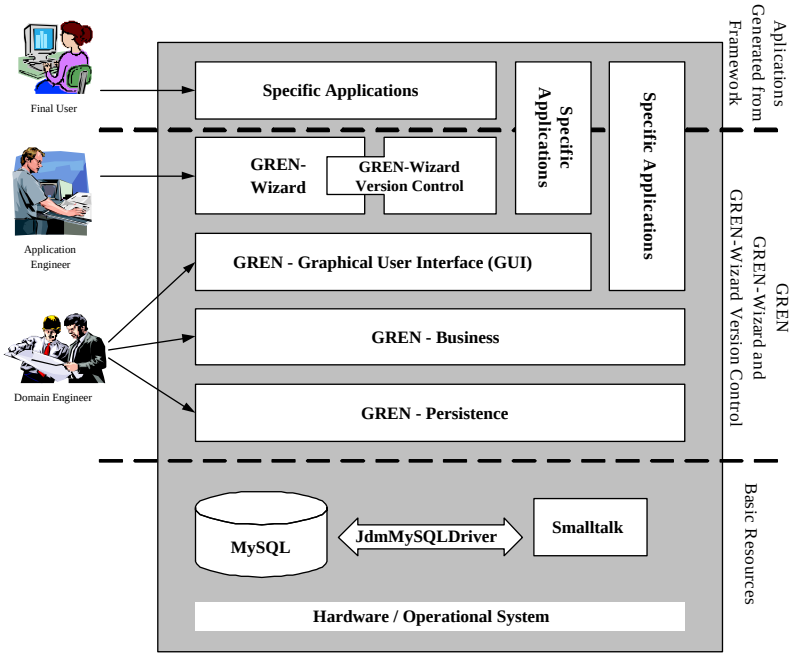

Figure 3: GREN framework architecture (adapted from (Braga, 2003))

directly with GREN-Wizard: the domain engineer, who is responsible for building the wizard (and, possibly, also has participated in the pattern language creation and in the framework development) and the application engineer, who uses GREN-Wizard to generate specific applications through its GUI. This GUI eases the specification of applications according to the GRN patterns used to model them. The final user executes the specific application code generated by GREN-Wizard.

GREN-Wizard is composed of three modules: the domain specification module, the application specification module, and the code generation module.

The **domain specification** module allows the representation of information about the GRN pattern language and its mapping to the corresponding GREN framework classes. The **application specification** module is responsible for storing information about the modeling of specific applications using the pattern language. Finally, the **code generation** module takes as basis the information available about the domain specification and about the application specification, to generate application specific code. This code, together with the framework code, is used to produce the application to be delivered to the final user.

According to Figure 4, the application engineer should first use the GREN-Wizard tool to generate the first version of the application. In this tool, as mentioned before, the application is specified in terms of the GRN patterns used to model it. If any manual change in the application source code is required, for example to include new functions not provided by the framework, it has to be done through the *GREN-WizardVersionControl* tool, to guarantee that each modification is stored in its meta-database. Afterwards, when new functions offered by the framework are requested to be included in the application, the application engineer uses the GREN-Wizard tool again. In this case, the tool generates the new application version, with the new functions inherited from the framework, together with all the modifications that were previously done in the source code. The final user interacts with all application versions released for use.

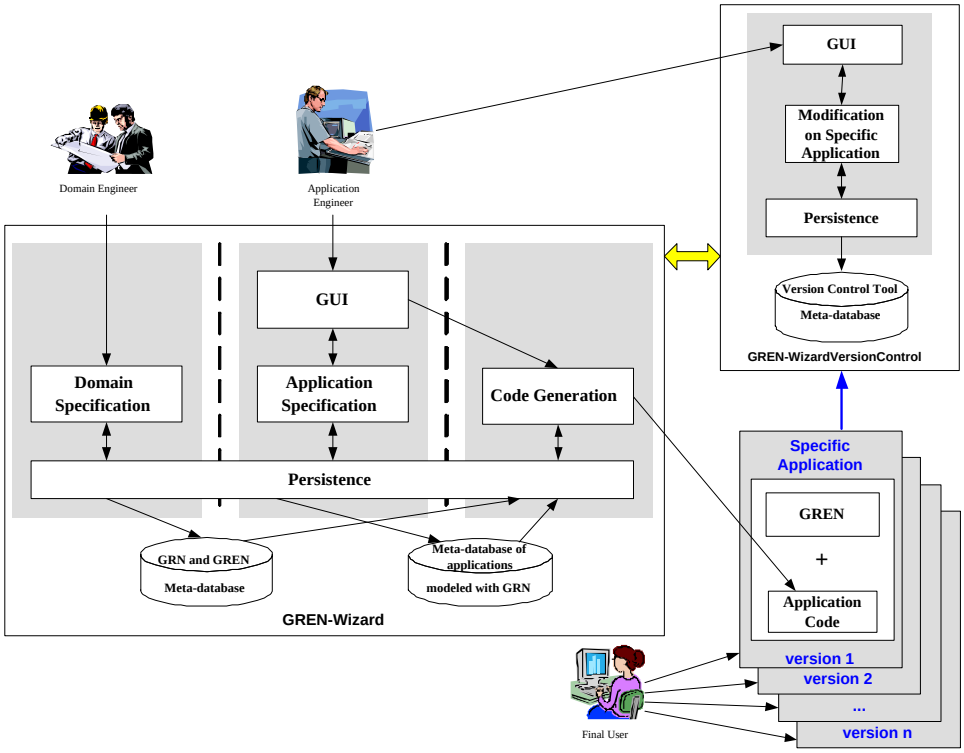

Figure 4: GREN-Wizard and *GREN-WizardVersionControl* architectures

3.3 Implementation

The *GREN-WizardVersionControl* tool was implemented using the same programming language used to implement GREN and GREN-Wizard, which is Smalltalk (specifically using the VisualWorks environment), so that these tools could be better integrated, to attend one of the desirable features of an SCM system, as mentioned by Midha (Midha, 1997). The *GREN-WizardVersionControl* meta-model was mapped to a relational database (MySQL (MySQL, 2003)).

Figure 5 presents the main screen for the *GREN-WizardVersionControl* tool, whose interface design was based on the VisualWorks *System Browser*, in order to make them as similar as possible, making it easier to learn and use. This allows the user to naturally use the tool functionality, as part of his daily work. This attends to another desirable feature of an SCM system (Midha, 1997).

Moreover, managing and configuring the *GREN-WizardVersionControl* tool is easy, what makes it fit another desirable feature of an SCM system (Midha, 1997).

As mentioned before, besides creating the *GREN-WizardVersionControl* tool, it was necessary to update the GREN-Wizard tool, so that the changes done by the software engineer in the source code of a certain application were considered in the future instantiations of the framework for that application. So, when a new version of the application is generated by GREN-Wizard, it is able to detect classes, methods, and attributes that were removed, included, or updated in the previous version of the application with the support of the *GREN-WizardVersionControl*, because they are stored in its meta-database.

Both source code inclusion and removal is automatically resolved by the GREN-Wizard tool. However, modifications done to methods that are inherited from the framework are identified by the tool as a method conflict between application and framework source code. This requires the intervention of the

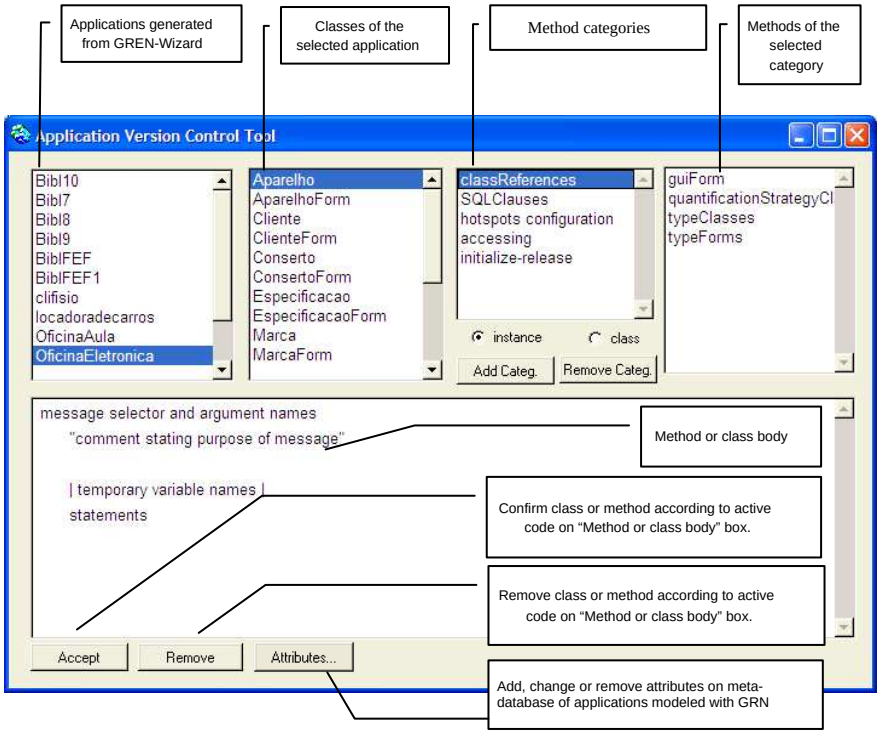

Figure 5: Main screen for the *GREN-WizardVersionControl* tool

software engineer to solve the problem at runtime, during the GREN-Wizard code generation. So, when the GREN-Wizard tool detects conflicts, a screen is shown to the application engineer, as illustrated in Figure 6 (left side). The application engineer has to inform the tool about the method content to be considered by the instantiation tool during the application generation. This is done by copying and pasting the correct content in the text box *Resulting Method* (Figure 6 (right side)). After that, the *Accept* button should be pressed to confirm the compilation of the source code lines and to proceed with the framework instantiation.

The GREN-Wizard tool also creates the database and MySQL *scripts* to allow the creation of tables for the newly generated application. A modification was performed in this functionality to allow the application engineer to choose between: 1) generating a new database or 2) keeping the present one, just updating its modified structure to avoid loosing data already inserted into the database.

The *GREN-WizardVersionControl* tool was tested in another reengineering case study for the library system, with the PARFAIT support, and it has shown to be effective, as described in the next section.

4 *GREN-WizardVersionControl* tool - Example of usage

Figure 7 presents the "Book Loan" [5] screen and the SQL script to create the `BookLoan` table, for the first version of the library system. Both of them were generated by GREN-Wizard during the reengineering with PARFAIT support.

During the application of the reengineering process, it was observed the need to make some changes

[5] that is, "Empréstimo de Livro" in Portuguese.

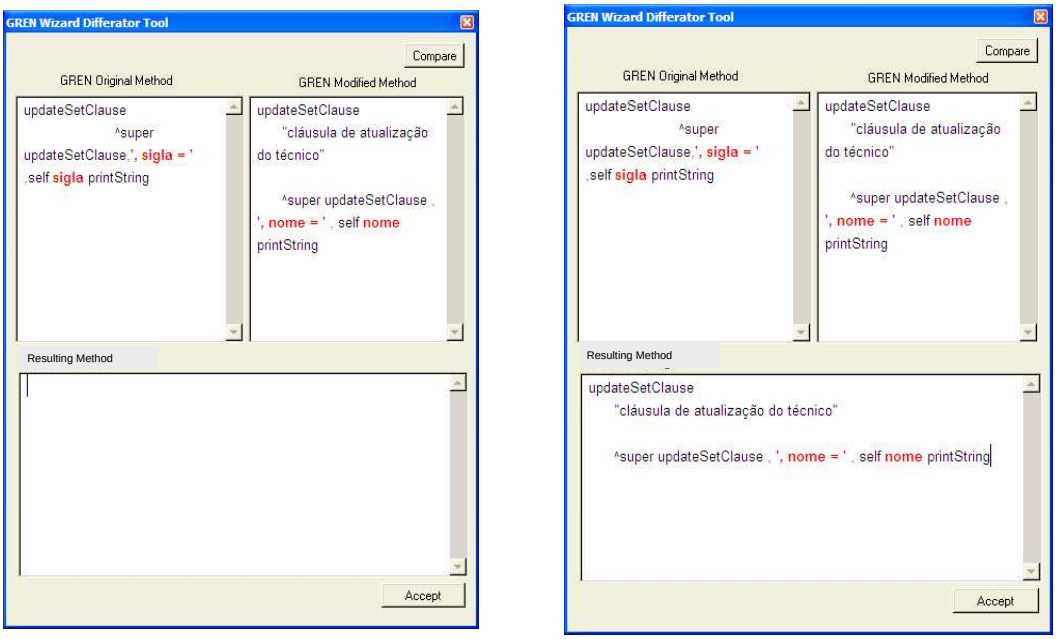

Figure 6: Screen for solving conflicts between framework and application methods

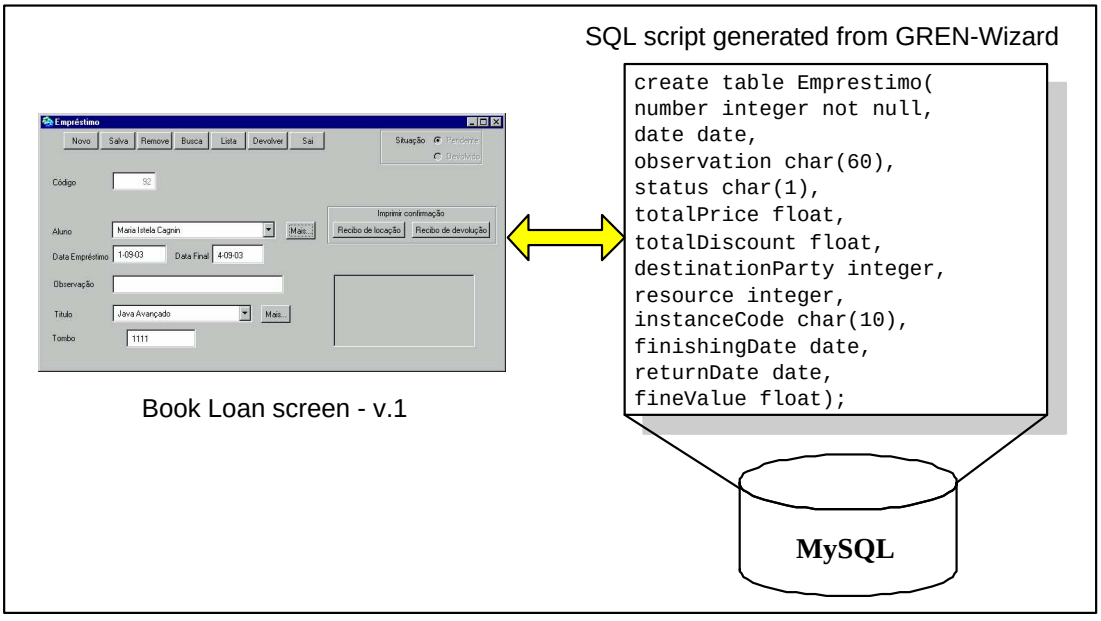

Figure 7: Part of Library system application – Version 1

in the source code of the first application version. The *GREN-WizardVersionControl* tool was used to do that. Figure 8 presents one of the changes done, to modify the `windowLabel` method, which was inherited from the GREN framework: its original content "`Loan`" was changed to "`Book Loan`".

This change was stored in the *GREN-WizardVersionControl* database (Figure 8), to allow its posterior incorporation by GREN-Wizard in future versions of the system.

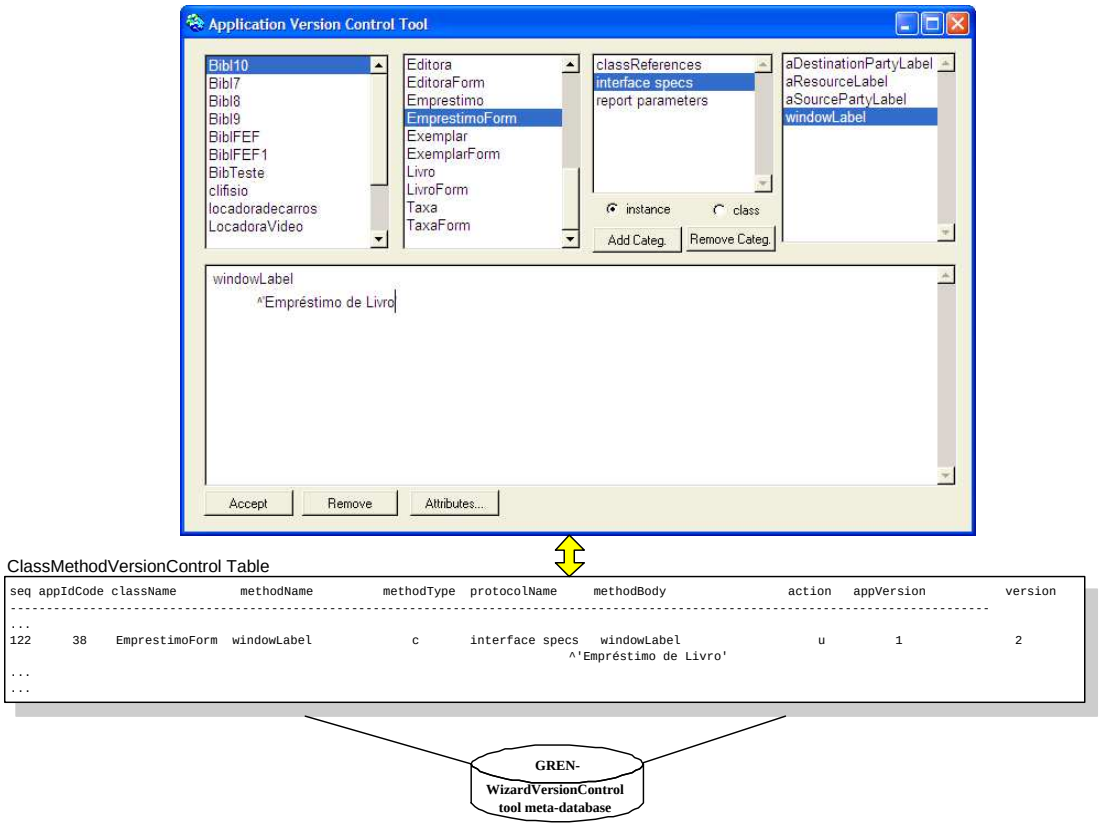

Figure 8: Source code change in version 1 of the library system

Another change requested by the customer was the addition of a new functionality (*"Charge a fine"*, as mentioned in Section 3). In this case, as GREN supplies this functionality, the GREN-Wizard tool was used to instantiate GREN again, to include this new function. During the instantiation process, GREN-Wizard detected the change done before in the `windowLabel` method and opened a special screen to solve the conflict between the method inherited from the framework and the method changed directly in the application, as shown in Figure 9. The application engineer is responsible for solving this conflict, through the screen presented by the *GREN-Wizard Differator Tool* (Figure 9), which is automatically opened by GREN-Wizard at runtime during the instantiation.

Figure 9 also presents some screens of the library system new version: "Book Loan", with the updated window label, and "Fine Rate" [6], resulting from the functionality added through GREN-Wizard, as well as its SQL script.

[6] that is, "Taxa de Multa" in Portuguese.

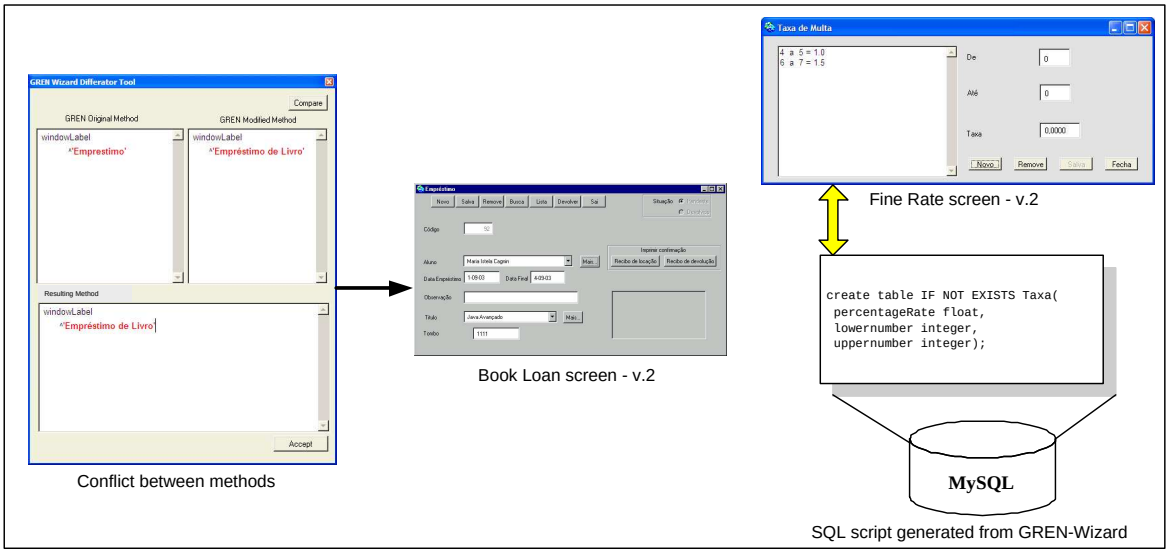

Figure 9: Part of Version 2 functionality for the library system

5 Conclusions

The *GREN-WizardVersionControl* tool supports both reengineering and development processes based on the GREN framework, using an incremental approach, as it allows the application engineer to have an historical log of all changes done in each new system version. These changes are automatically incorporated to new versions generated by GREN-Wizard, with minimum intervention of the application engineer.

As we cannot guarantee that the source code generated by GREN-Wizard is always in the same order, existing tools for version control (for example, CVS) are difficult to use to merge the application versions. Besides, these tools do not have control mechanisms neither to detect source code conflicts at instantiation time, nor to deal with database maintenance of the new application version. These facts have motivated the development of the *GREN-WizardVersionControl* tool.

Other case studies should be performed with the tool to refine the tests already done and to add improvements. Moreover, it is necessary to use it in other contexts, for example, software maintenance. Although the tool presented in this paper is specific for a particular framework, its conceptual modeling and architecture could possibly be used to implement other tools with the same goals for other existing frameworks.

It should be observed that *GREN-WizardVersionControl* is classified as an SCM system, as it fulfills most of the desirable features (Midha, 1997) for this type of system: it is easy to use; it allows the user to interact with its functionality as part of his daily job; it is easy to manage; and it is integrated to GREN-Wizard.

References

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2002). Agile software development methods. review and analysis. ESPOO (Technical Research Centre of Finland)' 2002. VTT Publications n. 478. `http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf`. Accessed: December,2003.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd edition.

Braga, R. (2003). *A Process for Construction and Instantiation of Frameworks based on a Domain-Specific Patterns Language*. PhD thesis, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo. (in portuguese).

Braga, R. T. V., Germano, F. S. R., and Masiero, P. C. (1999). A pattern language for business resource management. In *PLOP'1999, Conference on Pattern Languages of Programs*, pages 1–33.

Braga, R. T. V. and Masiero, P. C. (2003). Building a Wizard for Framework Instantiation Based on a Pattern Language. In *OOIS'2003, 9th International Conference on Object-Oriented Information Systems*, pages 95–106, Geneva, Switzerland. Lecture Notes on Computer Science, LNCS 2817.

Cagnin, M. I., Maldonado, J. C., Germano, F. S., Chan, A., and Penteado, R. D. (2003a). A reengineering case study using the PARFAIT process. In *SDMS'2003, Simpósio de Desenvolvimento e Manutenção de Software da Marinha*, Niterói, RJ. CD-ROM, (in portuguese).

Cagnin, M. I., Maldonado, J. C., Germano, F. S., and Penteado, R. D. (2003b). PARFAIT: Towards a framework-based agile reengineering process. In *ADC'2003, Agile Development Conference*, pages 22–31. IEEE.

Carnegie Mellon University (2002). Capability maturity model integration - version 1.1. Technical Report CMU/SEI-2002-TR-003, Carnegie Mellon University, Software Engineering Institute.

Chan, A., Cagnin, M. I., and Maldonado, J. C. (2003). Application of PARFAIT agile reengineering process in a library control legacy system. Working Document, ICMC-USP.

Cincom (2003). Visualworks 5i.4 non-commercial. `http://www.cincom.com/`. Accessed: February, 2003.

CVS (2004). Concurrent Versions System - The open standard for version control. `http://www.cvshome.org`. Accessed: December, 2003.

Fayad, M. and Schmidt, D. C. (1997). Object-oriented application frameworks. *Communications of the ACM*, 40(10).

Fowler, M. and Scott, K. (1997). *UML Distilled - Applying the Standard Object Modeling Language*. Addison-Wesley, first edition.

FreeVCS (2004). Free Version Control System. `http://www.thensle.de`. Accessed: April, 2004.

IBM (2004). Clearcase. `http://www-306.ibm.com/software/awdtools/clearcase`. Accessed: April, 2004.

ISO 9000 (1993). International Organization for Standardization. `http://www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html`. Accessed: April, 2004.

ISO/IEC 15504 (1998). International Standard for Software Process Assessment. `http://isospice.com/standard/tr15504.htm`. Accessed: April, 2004.

Johnson, R. E. and Foote, B. (1988). Designing reusable classes. *Journal of Object Oriented Programming – JOOP*, 1(2):22–35.

Microsoft (2004). Visual SourceSafe. http://www.msdn.microsoft.com/vstudio/previous/ssafe. Accessed: April, 2004.

Midha, A. K. (1997). Software configuration management for the 21st century. *Bell Labs Technical Journal*, 2(1). http://citeseer.nj.nec.com/midha97software.html. Accessed: February, 2004.

MySQL (2003). Mysql reference manual. http://www.mysql.com/doc/en/index.html. Accessed: December, 2003.

OMG (2003). Unified Modeling Language Specification. Version 1.5, formal/2003-03-01. http://www.omg.org/technology/documents/formal/uml.htm. Accessed: April, 2005.

Paulk, M. C. (1993). Capability maturity model for software - version 1.1. Technical Report CMU/SEI-1993-TR-24, Carnegie Mellon University, Software Engineering Institute.

Pressman, R. (2001). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 5th edition.

Quma Software Inc. (2004). QVCS – Quma Version Control System. http://www.qumasoft.com. Accessed: April, 2004.

Reichenberger, C. (1991). Delta storage for arbitrary non-text files. *ACM*, pages 144–152.

Soares, M. D., Fortes, R. P. M., and Moreira, D. A. (2000). VersionWeb: A Tool for Helping Web Pages Version Control. In *IMSA'2000, 4th International Conference on Internet Multimedia Systems and Applications*, pages 275–280, Las Vegas, Nevada, USA.

Synergex (2004). PVCS. http://www.pvcs.synergex.com. Accessed: April, 2004.

Taligent (1997). Building object-oriented frameworks. http://www.ibm.com/java/education/oobuilding/index.html. Accessed: February, 2002.

Telelogic (2004). Continuus/CM. http://www.telelogic.com/continuus.cfm. Accessed: April, 2004.

Tourwé, T. (2002). *Automated Support For Framework-Based Software Evolution*. PhD thesis, Department of Computer Science, Vrije Universiteit Brussel, Brussel.

Turk, D., France, R., and Rumpe, B. (2002). Limitations of agile software processes. In *Third International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'2002)*, pages 43–46, Alghero, Sardinia, Italy.