Version Control in Crosscutting Framework-Based Development

Maurício Massaru Arimoto
Centro Universitário Eurípides de Marília – UNIVEM
Marília – São Paulo, Brasil, Caixa Postal 2041 -CEP 17.525-901
mauricioarimoto@gmail.com

Maria Istela Cagnin
Centro Universitário Eurípides de Marília – UNIVEM
Marília – São Paulo, Brasil,
Caixa Postal 2041 - CEP 17.525-901
istela@univem.edu.br

Valter Vieira de Camargo
Centro Universitário Eurípides de Marília – UNIVEM
Marília – São Paulo, Brasil, Caixa Postal 2041 - CEP 17.525-901
valtercamargo@hotmail.com

ABSTRACT

Crosscutting Frameworks are a type of Aspect-Oriented Framework, which includes only one crosscutting concern, such as persistence, distribution or security. This type of framework is currently being researched in-depth, however little attention has been given to the version control of these frameworks and of the applications developed with their support. Version control concerning the development based on crosscutting frameworks is more complex than in traditional development. Moreover, it is more complex than in the development based on traditional frameworks, which require suitable techniques to control the evolution of both the frameworks and developed applications. Thus, in this paper we present guidelines and a tool which aims to control the versions of Crosscutting Frameworks, as well as the applications which are developed with their support.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement: *Version Control*;

General Terms

Management, Standardization, Theory

Keywords

Framework, Software Configuration Management, Version Control, Aspect-Oriented Frameworks, Crosscutting Frameworks.

1. INTRODUCTION

Various techniques and tools have been used to improve the software development process. An example is the use of "Frameworks" when developing applications. The purpose is to re-use the software and, consequently, improve the productivity, quality and maintainability [9]. There are authors who investigate the development of applications with object-oriented frameworks (OOF) [9] and others who research aspect-oriented frameworks [5][11][12][14][16][17][18]. Regardless of the technique used in its construction (object-oriented (OO) or aspect-oriented (OA)), a framework is an architecture which can be reused to support the development of new applications giving preference to reusing rather than other techniques of software reuse, resulting in better productivity and quality levels.

Crosscutting Framework (CF) [5, 6] is a term which was created to more accurately represent the most common type of aspect-oriented framework (AOF) found in the literature, i.e., one that includes only one crosscutting concern, such as persistence, cryptography, competition and security.

The Crosscutting Framework-based (CFs) development process [5,6] consists of weaving (or coupling) one or more of these frameworks to the application, whereby each one supports a certain part/functionality of the application. An application developed with the support of CFs becomes dependent on this reuse technology as the architecture of the application includes the framework(s). This also happens with OOF. As there may be evolutions and modifications in both CFs and the application, producing different versions of this software, there must be an appropriate version control which manages the framework - or frameworks - versions, which are related to each version of the application. This situation is more complex than what happens with traditional application frameworks [9], in which an application is generally dependent on only one framework.

Much research has been carried out on CFs and many of these frameworks have been developed [5], [11], [12], [16], [17], [18]. However, little attention has been given to the version control in this particular context. In a previous paper, twelve CFs organized into three families were developed: Persistence, Security and Business Rules [5]. The experiment using these CFs when developing various applications has shown the need for a more refined version control.

This paper presents guidelines to manage versions in the development based on CFs and a support tool. The tool, which is called TOFRA, also controls versions when the development uses conventional application frameworks [1], however in this paper the focus is to show how the version control is done in the CF-based development. The tool was developed in Java and has Web architecture. It can be used by teams who are in different places.

In Section 2, the main difficulties found in controlling CF versions are reported. In Section 3, guidelines to carry out the version control in the CFs are presented. In Section 4, the support tool is described with the class diagram and its functionalities. In Section 5, an example of how to use the tool is presented in order to illustrate its main characteristics. In Section 6, related papers are presented, and in Section 7, the conclusions, limitations and perspectives on future papers are presented.

2. VERSION CONTROL ON CF-BASED DEVELOPMENT

Version control in CF-based development is more complex than in the development based on conventional application frameworks. When developing conventional frameworks, an application is obtained by instantiating the framework, making the application dependent on this framework. Thus, a framework can have various versions and each one may be used to develop various versions of many applications. However, an application version will always be dependent on only one version of a specific framework [1], [9].

However, in the CF-based development, the application can be developed with the support of various CFs whereby each one supports the development of a part or functionality of the application. Thus, the problem becomes worse as an application may be developed with the support of various versions of many CFs. For example, version 1.0 of an application may have been developed with the support of version 1.0 of a Persistence CF, of version 1.5 of a Security CF and of version 3.2 of a Competition CF. Within the terminology used in this paper, this set of versions included in the development of an application corresponds to a certain software "configuration".

A detail that also increases the difficulty level of the version control is that CFs can be of "application" level and of "lower" level, depending on the concern. CFs of an application level are the ones that can be only coupled directly to an application, but not to other concerns or CFs. However, CFs of a lower level are usually not coupled directly to the application, but only to other concerns or CFs. For example, in the set of CFs developed in a previous paper [5], Pooling and Caching CFs can only be coupled to a Persistence CF and not directly to the application. On the other hand, a Persistence CF can be coupled only directly to the application. Thus, in a certain configuration, it is important to distinguish the CFs which are coupled directly to the application from the CFs which are coupled to other CFs.

To show the difficulty concerning the version control on the CF-based development, in Figure 1 there is a graph that represents five developed applications, with their respective versions and CFs that were used in the development of these applications. The white vertices represent the applications, the gray vertices represent the CFs and the edges represent the coupling/dependence among them. CFs found at the top of the Figure (part (a)) are CFs of a lower level. CFs that are in the intermediary part (part (b)) are CFs of an application level and in the lower part, (part (c)) the applications can be found.

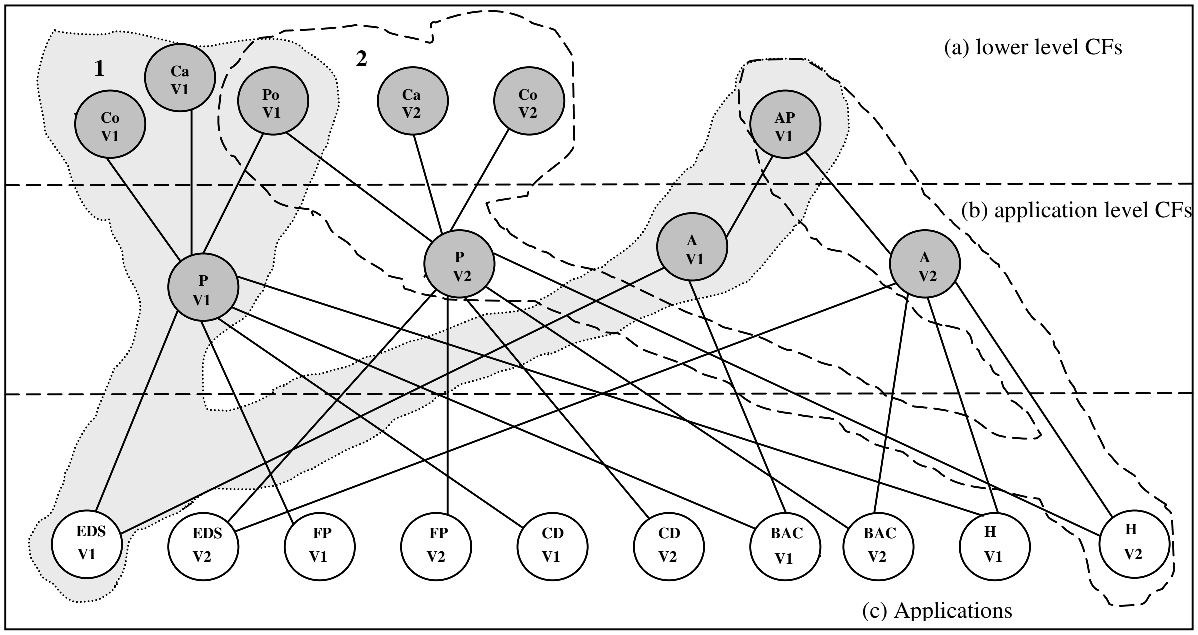


Figure 1. Graph of the Developed Application Configurations

The application "Electronic Devices Shop" in version 1.0 (vertice EDS V1) was developed with version 1.0 of the Persistence CF (vertice P V1), version 1.0 of Connection CF (vertice Co V1), version 1.0 of Pooling CF (vertice Po V1), version 1.0 of Caching CF (vertice Ca V1), version 1.0 of Authentication CF (vertice A V1) and version 1.0 of Authentication Politics CF (vertice AP V1). This set of versions form a "configuration" which is defined by the grayish geometric shape and marked by the number 1. Although CFs of a lower level are coupled only to CFs of an application level, all of them are inside the configuration of this application. The same application, but in version 2.0 (vertice EDS V2) was developed based on version 2.0 of the Persistence CF (vertice P V2), on version 2.0 of the Connection CF (vertice Co V2), on version 1.0 of the Pooling CF (vertice Po V1), on version 2.0 of the Caching CF (vertice Ca V2), on version 2.0 of the Authentication CF (vertice A V2) and on version 1.0 of Authentication Politics CF (vertice AP V1). Likewise, the application "CDs Stores" (CD) follows the same representation outline and the same happens to the applications "Payroll" (FP) and "Banking Account Management" (BAC).

The other geometric shape, marked with number 2, also represents a configuration, but of version 2.0 of the application Hotel (vertice H V2). This configuration involves version 2.0 of the

Persistence CF (vertice P V2), version 2.0 of the Connection CF (vertice Co V2), version 2.0 of the Caching CF (vertice Ca V2), version 1.0 of the Pooling CF (vertice Po V1), version 2.0 of the Authentication CF (vertice A V2) and version 1.0 of Authentication Politics CF (vertice AP V1). Note that version 1.0 of the Pooling CF and version 1.0 of Authentication Politics CF are shared (used) by both applications: Electronic Device Shop and Hotel. Maintaining a registration of this dependence and, consequently, of the CFs that are used by more than one application is very important as a change of requirements in a certain application can result in modifications in the CFs involved in the configuration of this application. Therefore, for a more refined version control to be maintained, one should also update the other configurations that use the same modified versions. Version 1.0 of Pooling CF, for example, is used by all of the versions of the developed applications. Indiscriminate changes in this CF can lead to unwanted side effects in the applications which use it.

3. GUIDELINES FOR VERSION CONTROL IN CF-BASED DEVELOPMENT

The main characteristic of a technique for the version control is to store the CF versions and their variabilities, and also the developed applications and their requirements (functional and non-functional). Afterwards, the relationships of these versions should be determined to create the “configurations”. A specific characteristic is to let relationship 1 for n be established between an application and various CFs - this does not happen in the development based on conventional frameworks [9].

After various configurations are determined, the technique should consult the CF versions which are used for more than one application. This helps to analyze the impact of a modification.

Another important point is to maintain a registration of the allowed dependences among the versions of CFs of application level and of lower level. It is usually necessary because, as CFs evolves, some versions can become incompatible. For example, the graphs shown on Figure 2 shows the version dependences among some of CFs developed by [5]. The vertices of the graph represent versions of CFs and the edges represent that the coupling among the versions is allowed. For example, on the part defined with the letter (a) on Figure 2, it is shown that the version 1.0 of Persistence CF - represented by the vertice P V1 - can only be used with the versions 1.0 and 2.0 of Caching CF - represented by the vertices Ca V1 and Ca V2 - and not with the version 3.0. That same version of Persistence CF can also make use of the versions 1.0 and 2.0 of Pooling CF - represented by the vertices Po V1 and Po V2. Yet, on part (b) of Figure, it is shown that the version 3.0 of Caching CF can only be used with the version 2.0 of Persistence CF, but the versions 1.0 and 2.0 of Pooling CF can also be used in that version of Persistence CF.

It is also interesting to keep a registration of the mandatory dependences among CFs of the application level and of the lower level. For example, Persistence CF cannot be used in any configuration without Connection CF. Although the dependences among the versions of CFs are being represented as graphs, other alternative ways can be used, for example, a feature diagram [10].

Note that the edges of the graphs shown in Figure 1 and Figure 2 have different meanings. In the graph in Figure 1, the edges show the couplings already performed in a certain configuration. However, in Figure 2 the edges represent the allowed coupling.

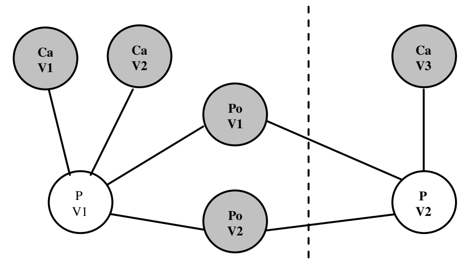


Figure 2. Graph of the Allowed Couplings among CFs

It is worth mentioning that a version control technique in CF-based development should allow one to take control at three different moments: when the framework is being developed, when the application is being developed/modified and when requests are responded to concerning modifications in the application.

In the first case, the framework developer must register the initial version of the developed framework and establish the dependence of versions between him and other existent CFs. In the second case, the application developer must register the version of the application and the version of each selected framework when it is being developed/modified. In this case, the selection of the framework to support the development of the application cannot violate the dependences of versions among CFs which have previously been established.

In this last case, it should be decided whether the modification should be made in the application or in the frameworks. This decision is supported by a framework evolution process, called PREF (Cagnin *et al.*, 2004). If the decision is to modify the application, it is necessary to modify it and register only the new version of the application. On the other hand, if the decision is to modify one or more frameworks, an impact analysis should be made previously in order to verify whether the modification to be accomplished can affect (or not) the behavior of applications previously created with the support of the frameworks in question.

4. TOFRA: TOOL OF VERSION CONTROL IN CF-BASED DEVELOPMENT

In Figure 3, the conceptual class diagram of the tool is presented. The top of the diagram represents the control of the user’s access (classes, `User`, `Role` and `Rights`). This control allows only authorized users to modify the content of the database. The `Application` and `Framework` classes represent the applications and CFs which can be registered, while the class `Domain` represents the crosscutting concern covered by each CF, for example, persistence, distribution or access control. The `FrameworkVersion` and `ApplicationVersion` classes are responsible for controlling the CFs versions and the application versions, respectively. By means of the relationship “is vinculated to”[1], the dependence/coupling between the application versions and the CFs versions is registered. The self-relationship “can be coupled” in the `FrameworkVersion` class represents the dependences allowed among the CF versions while the relationship “is coupled to”, in this same class, represents the dependences that are actually established.

[1] The term “vinculated” is being used to generalize the relation between applications and frameworks. Thus, when an application is “vinculated” to a framework, it means that the framework can be either OO or OA.

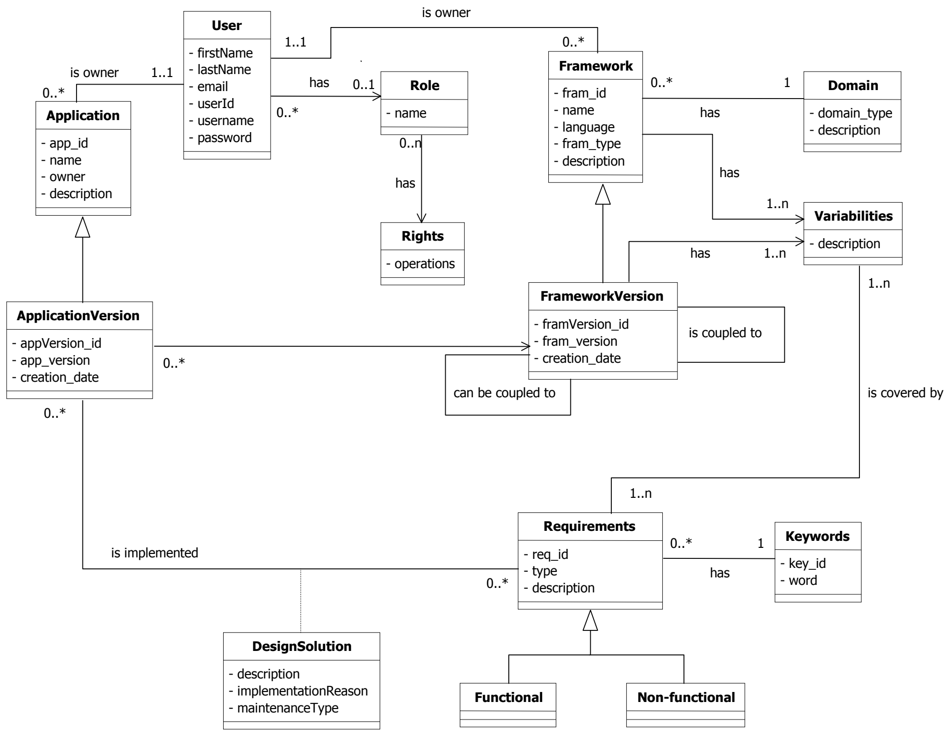


Figure 3. Conceptual Class Diagram

The `Variabilities` class maintains the variabilities of each CF, for example, Connection CF allows the connection to be made by means of Windows ODBC or by means of a native driver. Another example is regarding the Authentication Politics CF, which provides two alternative ways to control the wrong attempts to access the system: by a specific time or by a number of sequential mistakes.

The `Requirement` class manages the requirements of each application version. The relationship between this class and the `Variabilities` class manages the application requirements covered and not covered by each version of a CF present in a specific configuration.

Some of the tool functionalities are commented next. The functionality "Manage Framework Versions" in the Manage Frameworks module makes it possible to register the CF versions. All of the information related to the framework versions, such as: the framework name, the current framework version, dates of creation of the version and the lower level CF versions are dealt with by this functionality.

The functionality "Manage Application Versions" in the Manager Applications module not only registers the application versions, but also the association of these versions to specific versions of one or more frameworks. By means of this functionality, it is possible to store, in a disciplined way, all of the information regarding the application versions based on frameworks. The functionality "Manage Requirements Covered by Framework" in the Manage Requirements module makes it possible to register the requirements (functional or non-functional) covered by the framework, i.e., the application requirements implemented/attended by each version of the framework. All of the information regarding these requirements is dealt with by this functionality.

The functionality "Check Requirement History" in the Manage Consultations module consults the Requirements History to visualize a list which contains the requirements covered and non-covered by each version of the framework. The functionality "Check Versions of Applications Dependent on a Framework Version" identifies all of the application versions developed with the support of a certain framework version.

5. CASE STUDY

To exemplify the use of the tool, we show some interfaces that exemplify its use in the version control of the "Electronic Device Shop" application, which is denoted by the geometric shape (Configuration 1) presented in Figure 1. The interfaces shown below consider the initial registration of the Persistence CF version and of the application. They also show how the version control is done after a change request when the application is already in operation.

In Figure 4, the Persistence CF registration is exemplified. As this CF is at an application level, it is not necessary to choose any CF in the "sub level of" box. However, during the registration of lower level CFs, it is necessary to do this in order to register the dependences/coupling allowed among the CF versions. The frameworks version registration interfaces are not shown as they are simple and they do not have important details.

After CFs and their versions are registered, it is possible to register the developed applications and their respective versions. Suppose that two versions of the "Electronic Device Shop" application are registered. Therefore, the “Manager Application Versions” functionality can be used to create dependences between the application version and the CF versions used in its development. In Figure 5, it is shown how version 1 of the Electronic Device Shop application can be associated to a certain version of some CF. For other CFs to be associated with the same application, the interface should be used again.

To simulate a maintenance scenario, after the “Electronic Device Shop” application was in operation, we chose to add a new requirement. Cryptography. As this new requirement was not covered by the Authentication CF, it was implemented directly to the application. Thus, it had to be registered in the Requirements History by the “Requirements non-Covered by the Framework” functionality, whose interface was not presented here.

Although only the context of the development and evolution of the “Electronic Device Shop” application was shown, all of the other applications and CFs in Section 2 were registered and managed by the tool. In Figure 6, the applications which were developed based on each CF and their respective versions can be clearly seen. As shown in Figure 1, version 1.0 of the “Electronic Device Shop” application is related to all of the CF versions shown in that figure. Some other applications, shown in Figure 1, are also present in Figure 6.

As the number of application versions increases, the difficulties to control these versions also increase. There are more difficulties using CFs, and consequently the variety of framework versions that can be associated to only one application, which could become problematic, in case this control is manually performed.

Following the guidelines presented in Section 3 and with the support of the developed tool, it was observed that the control of application versions and of the frameworks presented in Figure 1 (Section 2) can be performed suitably.

6. RELATED WORK

Currently, there are various Configuration Management tools. Each one has its particularities with different functionalityes and applied to different contexts. The open-source tools which are worth mentioning are: *Revision Control System* - RCS [17], *Concurrent Versions System* - CVS [7] and *Subversion* [8]. None of these tools match the need of controlling the framework versions and the versions of the applications which are developed based on these frameworks, as well as the dependence of versions among CFs.

The *GREN-WizardVersionControl* [4] tool was developed to control only the application versions based on the *GREN* framework [2]. At certain moments, these applications need to be adapted to the source code and, if the framework is instantiated again to include a new system requirement, the whole code included previously in the system is lost. Regarding the *GREN* framework, the *GREN-WizardVersionControl* tool prevents this from happening.

7. FINAL REMARKS

The main contribution of this paper refers to not only the supply of a tool, but also to guidelines that can be followed to control versions during the development of applications based on CFs. The tool helps on the control of the changes performed and on the reduction of problems identified in the version control, contributing to the software quality warranty. Another characteristic to be highlighted refers to its architecture, that allows to be made available on the Web and, consequently, accessible to people geographically dispersed, facilitating its use in the distributed development of software.

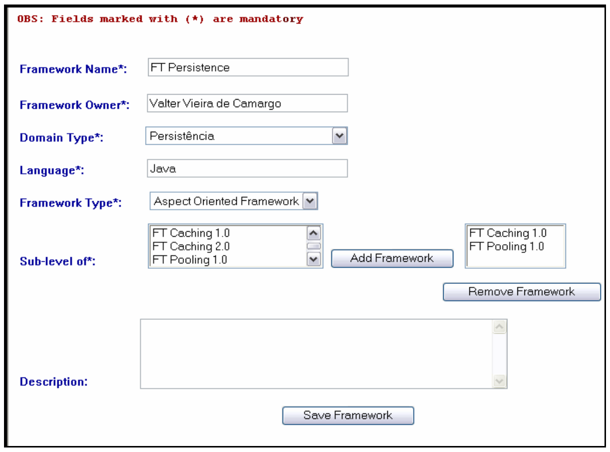
Although the guidelines shown in this paper were created based on CF’s context, we believe they can also be applied in component and library-based development, as well as managing package, as for example, Linux distribution.

A point that needs to be improved in the tool is regarding the registration of the allowed and mandatory dependences among CFs, not among CFs versions. Currently, just the allowed and mandatory dependences among the CF versions are implemented to the tool, allowing the mandatory dependences among CFs to be identified. Only the mandatory dependences among CFs cannot be obtained in the current version of the tool.

As future works, we can mention a better evaluation of the tool regarding to its usability and the adaptation and use of the tool in product lines of crosscutting frameworks [6].

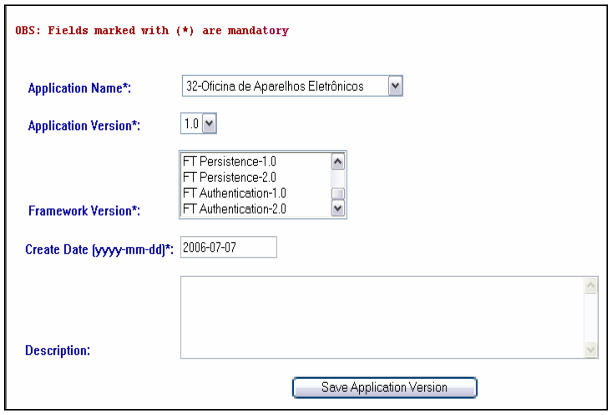
ACKNOWLEDGEMENTS

This work is supported by CAPES and CNPq: Projeto PROSUL-Proc (Latin-AOSD) 490478/2006-9.


OBS: Fields marked with (*) are mandatory

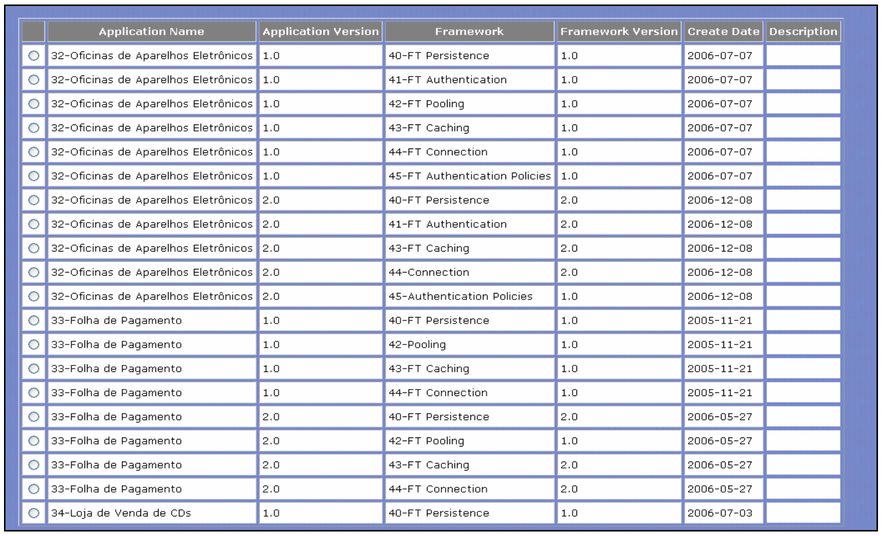
Framework Name*: FT Persistence
Framework Owner*: Valter Vieira de Camargo
Domain Type*: Persistência
Language*: Java
Framework Type*: Aspect Oriented Framework
Sub-level of*: FT Caching 1.0 / FT Caching 2.0 / FT Pooling 1.0 — FT Caching 1.0 / FT Pooling 1.0
Description:

Figure 4. Framework Registration Interface


OBS: Fields marked with (*) are mandatory

Application Name*: 32-Oficina de Aparelhos Eletrônicos
Application Version*: 1.0
Framework Version*: FT Persistence-1.0 / FT Persistence-2.0 / FT Authentication-1.0 / FT Authentication-2.0
Create Date (yyyy-mm-dd)*: 2006-07-07
Description:

Figure 5. Interface of Registration of Versions of Applications



	Application Name	Application Version	Framework	Framework Version	Create Date	Description
○	32-Oficinas de Aparelhos Eletrônicos	1.0	40-FT Persistence	1.0	2006-07-07	
○	32-Oficinas de Aparelhos Eletrônicos	1.0	41-FT Authentication	1.0	2006-07-07	
○	32-Oficinas de Aparelhos Eletrônicos	1.0	42-FT Pooling	1.0	2006-07-07	
○	32-Oficinas de Aparelhos Eletrônicos	1.0	43-FT Caching	1.0	2006-07-07	
○	32-Oficinas de Aparelhos Eletrônicos	1.0	44-FT Connection	1.0	2006-07-07	
○	32-Oficinas de Aparelhos Eletrônicos	1.0	45-FT Authentication Policies	1.0	2006-07-07	
○	32-Oficinas de Aparelhos Eletrônicos	2.0	40-FT Persistence	2.0	2006-12-08	
○	32-Oficinas de Aparelhos Eletrônicos	2.0	41-FT Authentication	2.0	2006-12-08	
○	32-Oficinas de Aparelhos Eletrônicos	2.0	43-FT Caching	2.0	2006-12-08	
○	32-Oficinas de Aparelhos Eletrônicos	2.0	44-Connection	2.0	2006-12-08	
○	32-Oficinas de Aparelhos Eletrônicos	2.0	45-Authentication Policies	1.0	2006-12-08	
○	33-Folha de Pagamento	1.0	40-FT Persistence	1.0	2005-11-21	
○	33-Folha de Pagamento	1.0	42-Pooling	1.0	2005-11-21	
○	33-Folha de Pagamento	1.0	43-FT Caching	1.0	2005-11-21	
○	33-Folha de Pagamento	1.0	44-FT Connection	1.0	2005-11-21	
○	33-Folha de Pagamento	2.0	40-FT Persistence	2.0	2006-05-27	
○	33-Folha de Pagamento	2.0	42-FT Pooling	1.0	2006-05-27	
○	33-Folha de Pagamento	2.0	43-FT Caching	2.0	2006-05-27	
○	33-Folha de Pagamento	2.0	44-FT Connection	2.0	2006-05-27	
○	34-Loja de Venda de CDs	1.0	40-FT Persistence	1.0	2006-07-03	

Figure 6: Interface of application version visualization

8. REFERENCES

[1] Arimoto, M. M., Camargo, V.V., Cagnin, M.I. A Version Control Tool for Frameworks. In: Proceedings of the Latin-American Conference of Informatics (CLEI 2007), October, San José, Costa Rica, 2007. (*In Portuguese*)

[2] Braga, R. T. V. (2003) A Process for Building and Instantiating of Frameworks based on a Specific Domain Pattern Language. PhD Thesis – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo. (In Portuguese)

[3] Cagnin, M. I.; Maldonado, J.C.; Masiero, P.C.; Braga, R.T.V.; Penteado, R. An Evolution Process for Application Frameworks. In: Workshop de Manutenção Moderna de Software, em conjunto com o XVIII Simpósio Brasileiro de Engenharia de Software, Brasília, DF, 8p, 2004.

[4] Cagnin, M. I.; Braga, R.T.V.; Penteado, R.; Germano, F.; Maldonado, J.C. (2005) A Version Control Tool for Framework-based Applications. Clei Electronic Journal, vol. 8, pp. 1-13.

[5] Camargo, V.V., Masiero, P.C. Aspect-Oriented Frameworks. In: Proceedings of 19º Simpósio Brasileiro de Engenharia de Software, Uberlândia – MG, Brasil, pp. 200-215, October, 2005. *(In Portuguese*).

[6] Camargo, V.V., Masiero, P.C. A Pattern to Design Crosscutting Frameworks. In Proceedings of the Annual ACM Symposium on Applied Computing (ACM-SAC 08), Fortaleza, Brazil, 2008.

[7] Cederqvist, P. Version Management with CVS. (1993) Official manual for CVS. Disponível em: <http://www.cvshome.org/docs/manual>. Acessed in 10 mar.

[8] Collins – Sussman, B., Fitzpatrick, B.W., and Pilato C.M. (2004) Version Control with Subversion: for Subversion 1.0. O'Reilly & Associates, Sebastopol, Califórnia, June. 2004. On-line book. Available on: <http://svnbook.red-bean.com>.

[9] Fayad, M.E., Schimidt, D.C., and Johnson, R. (1999) Building Application Frameworks: Object-oriented Foundations of Framework Design, John Wiley & Sons.

[10] Gomaa, H. (2004) Designing Software Product Lines With UML – From Use Case to Pattern-Based Software Architectures, Addison-Wesley.

[11] Hanenberg, S., Hirschfeld, A., Unland, R. and Kawamura, K. (2004) Applying Aspect-Oriented Composition to Framework Development – A Case Study. In: Proceedings of the 1st International Workshop on Foundations of Unanticipated Software Evolution, Barcelona, Spain, March.

[12] Huang, M., Wang, C. and Zhang, L. (2004) Towards a Reusable and Generic Aspect Library. In: Proceedings of AOSDSEC'04 (AOSD Technology for Application-Level Security). Workshop of the Aspect Oriented Software Development Conference, Lancaster, UK, March, 23.

[13] Pressman, R. S. Software Engineering: a practitioner's approach. 5th ed. McGraw-Hill, 2001.

[14] Rashid, A. and Chitchyan, R. (2003) Persistence as an Aspect. In: Proceedings of the 2nd International Conference on Aspect Oriented Software Development (AOSD), Boston–USA, March.

[15] Rausch, A., Rumpe, B., Hoogerdoorn, L. (2003) Aspect-Oriented Framework Modeling. In: Proc. of the 4th AOSD Modeling with UML Workshop (UML Conference 2003) October.

[16] Shah, V. and Hill, V. (2004) An Aspect-Oriented Security Framework: Lessons Learned. In: Proceedings of AOSDSEC'04 (AOSD Technology for Application-Level Security). Workshop of the Aspect Oriented Software Development Conference, Lancaster, UK, March, 23.

[17] Soares, S., Borba, P., Laureano, E. (2006) Distribution and Persistence as Aspects. Software: Practice & Experience, v. 36, n. 6, p. 711-759.

[18] Tichy, W. F. (1985) RCS – A System for Version Control. Software - Practice and Experience, vol.15, nº 7, pp.637-654.