An investigation on the approaches for version control systems

Vladimir Jotov

Abstract: *The article introduces the major flows in version control area. Here we present and classify the major features and challenges and their approaches in the problem area. Furthermore a roadmap for further elaborations is outlined in this article.*

Key words: *Software engineering, Version control, Source code management systems, Software configuration management.*

INTRODUCTION

Version Control Systems (VCS), known also as source code management systems, software configuration management or revision control, are the backbone in the software engineering area [4]. These systems allow the big software projects to achieve successful results. In this approach we will try to present the main flows and challenges in the VCS sphere.

We introduce in the next section the major principles used in repository architecture of VCS. In the third section we can find a presentation of the major versioning approaches. The fourth section is accentuates on the minor internal methods used in VCS. The next section is focused on an element in the versioning – the granulation of versioned objects. In the last section we present the challenges that a VCS shall meet as a topic for the future work.

SYSTEM REPOSITORY'S ARCHITECTURE

The repository is the core VCS subsystems. It is responsible for the storage of all versioned objects. As a versioned object we can define that it is an object (source code file, class, resource picture, etc.) that is under version control and every change of its attributes is registered in the VCS.

There are different types of repositories' architectures and on that point we can classify the VCS to client-server, distributed and hybrid [6].

The client-server architecture uses the principles of client-server technology. In this case the repository is at one place. This allows to use simpler maintain strategies to be used. The developer extracts a copy of the versioned object from the repository and after he finishes with changes he puts back to the repository the new object version. The basic representatives of this architecture are CVS, Subversion, and IBM Rational ClearCase [8].

In distributed VCS the repository is decentralized. It is more common to be used the definition "set of repositories" instead of only one. This architecture allows asynchronization of the software development process even using peer-to-peer networks [4]. Every developer has its own repository. The synchronization between repositories is a separate step [8].

In hybrid architecture we can see client-server architecture where the server part is distributed. Special strategies are used to synchronize the geographically separated servers [10]

VERSIONING STRATEGIES

If we look at the system interaction point then we can notice the following strategies: Snapshot; Changeset [7].

The initial strategy and the one easier to implement is the snapshot strategy. Here we have good system performance. The system keeps every object's version change independently. Unfortunately in this strategy has a bigger disk space demand.

In the Changeset strategy the main point is that the repository keeps only the difference between the versions also known as delta version [2]. As a consequence we could notice that the repository size becomes much smaller as we keep only the deltas between two versions in the repository. The disadvantage of this strategy is that in order to retrieve the current version the system is pushed to calculate all the changes from the first version.

In order to avoid the big memory demands of the Snapshot approach and the big performance leak of the Changeset, in [2] they solve it as combining the two approaches. The calculation of several change sets over specific snapshot is an attempt to find the best solution for the memory demands and the system load.

There are cases when the Changeset strategy cannot be used (Ex. versioning of none vector graphical [5], audio or video resources). The calculation of the delta in these cases is very difficult and can overload the system even for a small change. So for that type of objects the Snapshot strategy is preferable.

VERSIONING MODELS

In [1] are represented two approaches that are the most common used in VCS. They are Lock-Modify-Unlock and Copy-Modify-Merge. The first one is easy to be implemented but its main weakness is that the locked resource is untouchable for the co developers. There are some practices developed that helps us to avoid this [1] but they do not solve the problem as it is.

The Copy-Modify-Merge approach is based on update only parental (base) version and merges the code otherwise. At this point some researches [2, 5] mention three main level of merging:

- Textual merging is applied for plain text files. This merge approach detects only "physical" differences between the local and the base versions. This merging method is the most common used;
- Syntactic merging is more complicated than the textual. This method uses syntactical rules in order to achieve more intelligent merge results;
- Semantic merging is the approach with the greatest complexity. It performs sophisticated analysis of the versioned objects. They are so complex that this approach is used only into the practice only for source code or structured data as hypermedia.

GRANULATION LEVELS OF VERSIONED OBJECTS

How big is the versioned object? This question does not have an unambiguous answer. The versioned object can be as the whole software product as well as a single method or a field. The granularity of the versioned objects is a decision of company or project strategy level. In the practice the most common size is the source code file. And if we envisage that in the modern programming languages the file contains a class then the granulation is on class level. Further more from granulation of the objects depends the size of the lock or the synchronize size in the used versioning model [3].

We examine the granulation because defining the scope of one versioned object we can easily predict the VCS performance and system needs in different strategies. The smaller granulation is giving us the possibility to prepare more flexible to the customer needs versions instead to keep a large set of individual versions.

CONCLUSIONS AND FUTURE WORK

Using an appropriate versioning policy it can propose better software development, support and maintenance. The fundamental point for this is to make a strong balance in repository architecture, versioning strategy, versioning model, and to present a new approach for granulation of the versioned objects. All this will lead us to make a flexible model of version control system that will allow using of different methodologies in different projects [9].

REFERENCES

[1] Collins-Sussman Ben, Brian W. Fitzpatrick, C. Michael Pilato, Version Control with Subversion, Draft Revision 9837, http://bebas.vlsm.org/v06/tools/svn-book.html , 2004 (accessed 18.11.2007).

[2] Conradi, R. and Westfechtel, B. 1998. Version models for software configuration management. ACM Comput. Surv. 30, 2 (Jun. 1998), 232-282.

[3] Estublier, J. and Garcia, S. 2005. Process model and awareness in SCM. In Proceedings of the 12th international Workshop on Software Configuration Management (Lisbon, Portugal, September 05 - 06, 2005). SCM '05. ACM Press, New York, NY, 59-74.

[4] Jones, M. Tim, Version control for Linux, http://www.ibm.com/developerworks/ linux/library/l-vercon/, 2006 (accessed 18.11.2007).

[5] Lee, B. G., Chang, K. H., and Narayanan, N. H. 1998. An integrated approach to version control management in computer supported collaborative writing. In Proceedings of the 36th Annual Southeast Regional Conference ACM-SE 36. ACM Press, New York, NY, 34-43.

[6] Pressmann, Roger S., Software Engineering: A Practitioner's Approach, Chapter 27 – Change Management, 739-768, McGraw-Hill Professional, 2005.

[7] Wikipedia contributors, "Comparison of revision control software," Wikipedia, The Free Encyclopedia, Wikipedia, The Free Encyclopedia, 11 April 2008, 22:54 UTC http://en.wikipedia.org/w/index.php?title=Comparison_of_revision_control_software&oldid= 205021471 (accessed April 14, 2008).

[8] Wikipedia contributors, "List of revision control software," Wikipedia, The Free Encyclopedia, Wikipedia, The Free Encyclopedia, 13 April 2008, 16:42 UTC http://en.wikipedia.org/w/index.php?title=List_of_revision_control_software&oldid=2053553 64 (accessed April 14, 2008).

[9] Алистэр Коуберн, Каждому проекту своя методология, 2005, http://www.citforum.ru/SE/project/meth_per_project/ (accessed April 14, 2008).

[10] Новичков Александр, ClearCase - система конфигурационного и версионного контроля, 2000, http://www.citforum.ru/programming/rational/ clearcase.shtml (accessed April 14, 2008).

ABOUT THE AUTHOR

Vladimir Jotov, PhD student at Veliko Tarnovo University “St. Cyril and St. Methodius”. E-mail: vjotov@acm.org.