Towards PLM for Mechatronics System Design Using Concurrent Software Versioning Principles

Matthieu Bricogne, Louis Rivest, Nadège Troussier, Benoît Eynard

▶ **To cite this version:**

Matthieu Bricogne, Louis Rivest, Nadège Troussier, Benoît Eynard. Towards PLM for Mechatronics System Design Using Concurrent Software Versioning Principles. 9th International Conference on Product Lifecycle Management (PLM), Jul 2012, Montreal, QC, Canada. pp.339-348, 10.1007/978-3-642-35758-9_30 . hal-01526128

HAL Id: hal-01526128

https://inria.hal.science/hal-01526128v1

Submitted on 22 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Towards PLM for mechatronics system design using concurrent software versioning principles

Matthieu Bricogne[1], Louis Rivest[2], Nadège Troussier[1] and Benoît Eynard[1]

[1] Université de Technologie de Compiègne,
Department of Mechanical Systems Engineering,
CNRS UMR7337 Roberval
BP 60319, rue du Docteur Schweitzer,
60203 Compiègne Cedex, France
{matthieu.bricogne; nadege.troussier; benoit.eynard}@utc.fr
[2] École de technologie supérieure,
Department of Automated Manufacturing Engineering,
1100 Notre-Dame Street West
Montréal, Québec
Canada H3C 1K3
louis.rivest@etsmtl.ca

Abstract. In this paper, the Change Management (CM) and version control mechanisms used in the mechanical and software domains are presented respectively in the Product Lifecycle Management (PLM) and Application Lifecycle Management (ALM) approaches. Based on their comparison results, this paper discusses branching / merging concepts and presents them as an opportunity for PDM improvements.

Keywords: Mechatronics design, Product Lifecycle Management (PLM), Application Lifecycle Management (ALM), Version control.

1 Introduction

Products are becoming increasingly complex, integrating technologies from several fields, such as mechanical engineering, electronic or electrical engineering and software engineering. Mechanical systems developed since the 80's have thus evolved from electro-mechanical systems with discrete electrical and mechanical parts to integrated electronic-mechanical systems with sensors, actuators, and digital microelectronics. These integrated systems are called mechatronic systems [1].
To design such systems, Product Data Management (PDM) systems can in assist multiple-discipline integration. However, efforts are still required to turn PDM systems into collaborative platforms for mechatronic systems design [2].
This paper is organised as follows: after introducing the processes and systems used to manage data during mechanical and software design, the different change management and version control mechanisms used during the mechanical design and software design processes are presented and compared in the second section. In the

third section, the branching / merging concepts [2-3], derived from the software field, are presented and discussed. The fourth section presents these concepts as an opportunity for PDM improvements. Finally, prototype illustrating the concept usage on Mechanical Computer Aided Design (MCAD) data is proposed in the last section.

2 Change management and version control mechanisms

2.1 Change management and version control during mechanical design process

PDM is generally considered as one of the most important components in Product Lifecycle Management (PLM) implementation. Indeed, PLM is defined as a systematic concept for the integrated management of all product- and process-related information through the entire lifecycle, from the initial idea to end of life [4].
In the mechanical field, PDM systems are used, among others, to [2]:

- Enable collaboration through the entire virtual/extended enterprise (workflow and process management, communication and notifications, secure data exchange, etc.);
- Manage the product structure and its evolution during its different steps (xBOM);
- Track the different modifications performed;
- Help in the project management: organization (roles, tasks, etc.), planning, costs, etc.;
- Facilitate the integration of heterogeneous data derived from diverse design software tools;
- Enhance knowledge capture and reuse;,
- Provide a global Digital Mock-Up (DMU) centralizing the different design contributions.

In this paper, the focus will be on engineering change and systems supporting this activity. An engineering change is defined by Jarratt et al. as "an alteration made to parts, drawings or software that have already been released during the product design process. The change can be of any size or type; the change can involve any number of people and take any length of time" [5]. The authors also provide a generic model (**Fig. 1**) to illustrate the different steps of a change process. First, a request is made with the reason for, the priority, the type and the eventual impacts of the change. The second step is the examination of potential solutions. Impacts and risks associated with this solution are assessed in the third step. The fourth step consists in approving the solution. Usually, a committee is in charge of taking this kind of decision, but some shortcuts could exist depending on the level of emergency. This committee should represent the different stakeholders, such as the product design, manufacturing, marketing, and other teams. The fifth step is the implementation of the change, which can either occur immediately or can be phased in. Finally, the change has to be reviewed to check if initial intents are satisfied by the solution.

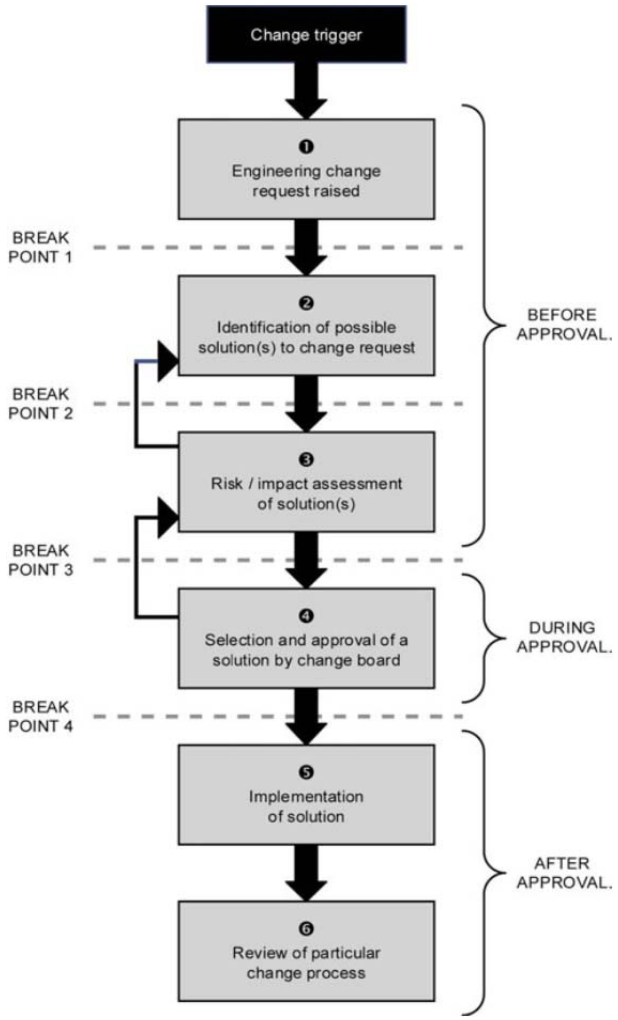

Fig. 1. A model of a generic change process [5]

This process can be considered as a formal and structured engineering change process. It can be supported by PDM thanks to the workflow functionality, where roles are attributed to the different stakeholders for each design project. However, as identified in the definition, this kind of procedure is used only if data have already been released. Otherwise, no formal procedure is used, and the data are sometimes not stored on PDM but on the local machines of designers. One aspect leading to this situation is a poor control of data before they are released.

This first part presents the different existing processes for change management in the mechanical field. In the second part, existing processes in software fields are presented.

2.2 Change management and version control during software design process

PDM was compared to Software Configuration Management (SCM) by Crnkovic et al. in the 00's [2]. They list several functionalities for both systems and try to distinguish the overlap and the differences, as illustrated in **Fig. 2**. However, even for functionalities with the same name, they note that they could be implemented differently.

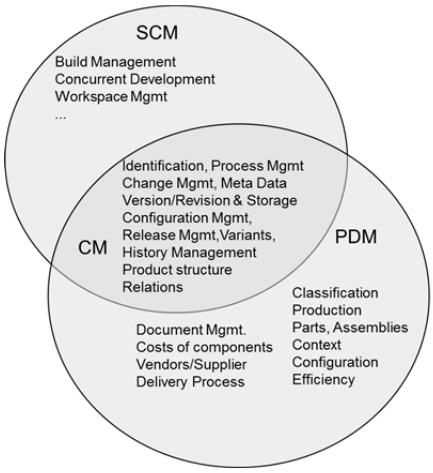

Fig. 2. SCM and PDM: similarities and differences in functionalities [2]

While SCM may be considered as a whole software development environment, including build and release management, for example, the basic functions it provides are version control (or revision control, or source control), concurrent development, configuration selection and workspace management. Focusing on these functions, SCM is very similar to PDM, as the Application Lifecycle Management (ALM) can be compared to PLM. ALM "has emerged to indicate the coordination of activities and the management of artefacts (e.g., requirements, source code, test cases) during the software product's lifecycle" [6]. ALM comes from the Configuration Management (CM) domain, which traditionally provides storage, versioning and traceability between all artefacts. It provides extra functionalities to support communication, reporting, traceability and tool integration, such as requirements and defects management, build and test facilities, etc., as illustrated in **Fig. 3** [6].

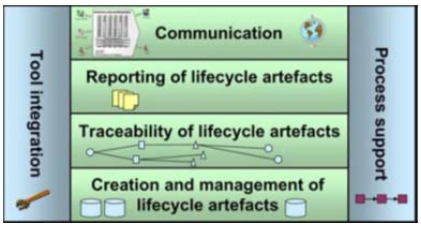

Fig. 3. Main elements of Application Lifecycle Management [6]

In this paper, the focus is on engineering change and systems supporting the activity. During software design, as in mechanical design, designers generally have a great deal of freedom before data are used in software that is sold or comes close to being sold (convergence step). This could be compared to the release mechanism, although several differences still remain in the way change is managed between the software and mechanical domains.

First, even during the design step, the data manipulated[1] are kept under the control of SCM. While such data are considered to be private, they are however generally not locked: other developers can import them, for integration purpose, for instance. The data are likely to change rapidly, generating a great number of versions of the same data. However, the data are all stored in SCM, using differential techniques to store only modified portions.

Secondly, software usually evolves from one version to another much more rapidly than do mechanical products. This implies that different versions of the same data can be involved in different versions of software. The effect of this is that several versions are modified at the same time without affecting one another.

These are just two out of many examples that illustrate the flexibility induced by the use of SCM; however, it also highlights the need to manage several "branches" containing different concurrent versions of the data and the need to merge different versions for reconciliation. These two concepts, which in this paper are called "branch and merge" [3], [2], will be presented in the third section.

After presenting some major differences between change management and version control in the mechanical versus the software domains, the next section illustrates the opportunities provided by using the branch / merge concepts in PDM.

2.3 Opportunities for PDM and methodology

In software engineering, version control systems allow multiple developers to edit the same file synchronously, thanks to the branching concept. After the first developer "checks in" his/her version to the central repository, the system provides facilities to merge further changes, preserving the work from the first developer. This second operation, based on the merging concept, is presented as an opportunity for PDM improvements.

This paper will try to identify the reasons why these concepts have so far not been adopted in PDM as well as the prerequisites for their implementation.

To answer these questions, the use of these concepts will be described in the software domain, after which the concepts will be adapted to the mechanical domain and illustrated, in an attempt to find solutions to the different obstructions.

---

[1] In the Configuration Management community, the data manipulated in CM are often named Configuration Items (CI)[3]

3 Description of branching / merging concepts in software design process

Branch and merge concepts have been presented in several papers. For example, Do et al. specify that "SCM supports branch/merge operations on its source codes (…) however, most PDM do not support the branch/merge operations on their documents" (see **Fig. 4**). This comparison, initiated by the software CM community, leads to the conclusion that PDM should be able to manage software data if PDM allows "SCM version control operations, such as branch/merge and code comparisons" [7].

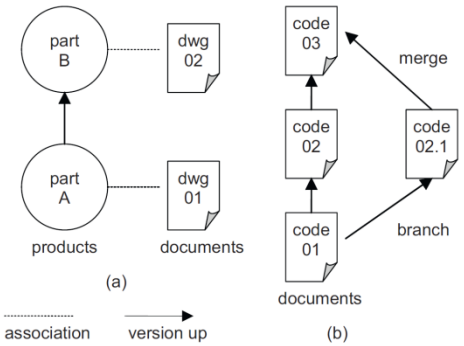

Fig. 4. Comparisons between PDM (a) and SCM (b) on document management [7]

To understand this claim, with which this paper fully agrees, it is necessary to understand how it is used in software engineering. First, it is essential to be familiar with the fact that several versions of software application could co-exist. For example, if the 2008 and 2010 versions of a software application have been released and are still maintained, all the bugs found on these versions will have to be corrected, and the correction should be carried forward on the future versions of the software.
Likewise, if the software is available on several Operating Systems (OS) such as Windows, iOS, Linux or even on smartphone OS, some components of the software will be present on all these OS and a modification performed on one of these shared components should impact all the versions.
Another example is the case of the addition of a new functionality to the software. In that case, the developer begins his/her modifications from a version of the Configuration Item (CI). Throughout the development cycle, the modifications are kept under the control of SCM by keeping them private. Other modifications can occur on the same CI for different reasons, such as maintenance, other developments, etc. When the new functionality is finally ready, the developer shall integrate the different modifications performed concurrently before being allowed to deliver his/her code.
These three examples are illustrated in **Fig. 5**, where the 47^{th} version of the CI has been created from the 43^{th} version to fix a bug found on the 2008 release. The modification has been carried forward to the 50^{th} version of the CI (a) to fix the bug in the 2010 release. However, a new functionality has been created for the 2010 release

which impacts the CI. To avoid a merge when this new functionality is delivered to the 2010 release (c), the developer has decided to reconcile the 47th and the 49th versions using a merge operation (b). This operation is performed through a "3-way merge" whose characteristics are as follows:

- Common ancestor version: 43rd version
- First version to integrate: 46th version
- Second version to integrate: 47th version
- Resulting version: 49th version

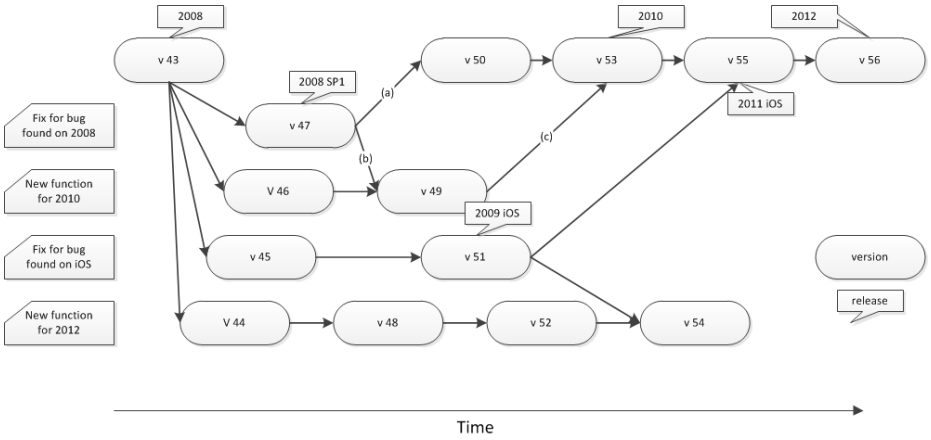

Fig. 5. Example of versioning graph of a Configuration Item

This example illustrates how version control systems allow multiple developers to edit the same file synchronously, thanks to the branching concept. In this example, the versions' numbers are given sequentially, but different strategies can be chosen for numbering, merge operations, etc…

The ability to work synchronously with several versions of the same CI is provided by the *diff* and *merge* tools. The diff tool computes the differences between files, while merge tool reconciles the different versions of the same object.

Software engineering is mostly based on text files (source code), and in its case, standard diff and merge tools are relevant. However, since the establishment of object-oriented programming, source code can no longer be considered as simple text. Reengineering, much like displacement of classes, requires "intelligent" diff and merge tools.

In the mechanical domain, most files are binary files, which has always been presented as a barrier for the adoption of concurrent versioning mechanisms. But structured files, such as XML files, can also be merged [8]. In the next section, conditions for the adoption of these mechanisms by PDM are presented, as well as opportunities for mechanical design.

4 Proposal for branching / merging concepts transposition to mechanical design process

As seen in the previous sections, mechanical data are managed differently before and after release. Generally, after release, the data are often not modified, and the change management process is already formally in place and supported by PDM workflows. Conversely, before release, change management could be much more structured and supported by PDM improvements based on an SCM versioning system. As illustrated in the previous section, the major capabilities offered by SCM are based on the diff and merge tools. Some "intelligent" diff and merge tools exist in the software domain, i.e., tools which can better understand the content or the files' semantic.
The same "intelligent" tools could be envisioned for the mechanical domain. A generic tool for all the mechanical technical data seems unrealistic, but tools dedicated for specific data and provided by creation tool publishers seems rational. The choice of the appropriate tool would be performed by PDM depending, on the data type. Even for merging the same data, several tools could be used, e.g., geometric diff and features tree diff. To illustrate this kind of solution, **Fig. 6** presents the geometric diff and merge functionalities proposed by the CATIA® software[2].

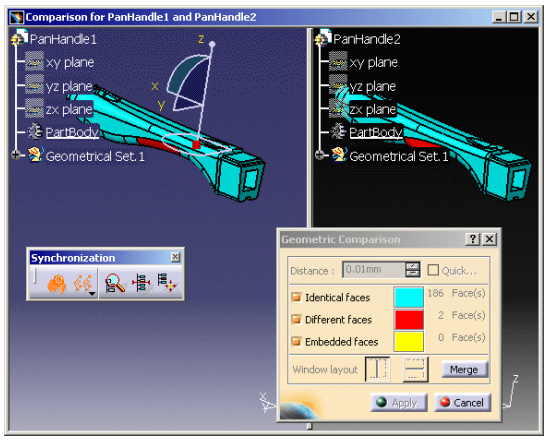

Fig. 6. Geometric diff and merge tool example: CATIA® "compare parts" functionality

This paper does not focus on 3D CAD models comparison. Brière-Côté et al. however, do present different methods and tools for such comparisons [10].
In this section; the obstructions to the opportunity for PDM improvements based on the branching / merging concepts have been presented. The need for dedicated diff and merge tools has been emphasized and illustrated. In the next section, the results of a prototype to prove that 3D CAD models can be managed with SCM concepts is presented.

---

[2] CATIA V5, http://www.3ds.com/products/catia/, Dassault Systèmes inc.

5 Illustration of SCM version control mechanisms on mechanical data

In order to illustrate that the change management for unreleased mechanical data can be improved using mechanisms derived from software engineering, an SCM system named TortoiseHg[3] was configured in order to be able to manage 3D CAD models. Thanks to the plug-in capability, STEP and CATIA® parts and product files were stored in SCM. Some of these files were checked out by several users, modified synchronously and checked back in, creating several versions / branches. A merge operation was then initiated from the SCM user interface, launching the appropriate merge tool. **Fig. 7** illustrates the SCM user interface resulting from these different operations and the choice offered to the user when he/she commits his/her change: create a new branch or merge files.

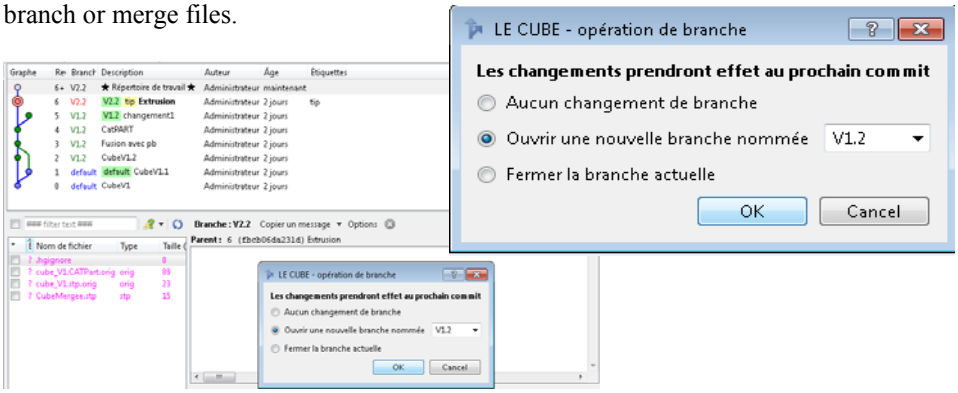

Fig. 7. 3D CAD Models management in an SCM system

Several merge tools were tested: classical 3-way merge for STEP files, as illustrated on **Fig. 8**, and dedicated software for commercial 3D CAD models.

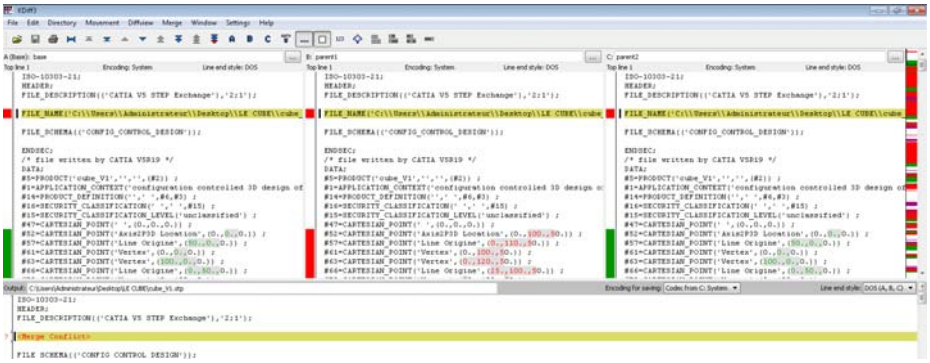

Fig. 8. Classical 3-way merge tool used on STEP files

[3] ToitoiseHg, http://tortoisehg.bitbucket.org

6 Conclusion

To design mechatronic systems, Product Data Management (PDM) systems can assist multiple-discipline integration. However, efforts are still required to turn PDM systems into collaborative platforms for mechatronic systems design. The first evolution proposed is to allow SCM version control operations, such as branch / merge, in PDM systems. Thus, software engineers can manage their source code items in PDM and mechanical engineers can work simultaneously on the same data. To achieve such functions, adapted diff and merge tools are required and PDM versioning mechanisms have to be modified. Further work will consist in continuing the specification of changes for PDM systems in order to transform them into collaborative platforms for mechatronic systems design.

References

1. Isermann, R.: Mechatronic design approach. The Mechatronics Handbook. Bishop, R.H. (Ed), CRC Press, Boca Raton (2002)
2. Bricogne, M., Troussier, N., Rivest, L., and Eynard, B.: PLM perspectives in mechatronic systems design. In Proceedings of Advances in Production Management Systems, APMS2010, Cernobbio, Como, Italy (2010)
3. Crnkovic, I., Asklund, U., and Dahlqvist, A. P.: Implementing and integrating product data management and software configuration management. Artech House Publishers (2003)
4. Estublier, J., Favre, J.-M., and Morat, P.: Toward SCM/PDM integration? In System Configuration Management, Vol. 1439, B. Magnusson, Ed. Springer-Verlag, Berlin/Heidelberg, pp. 75-94 (1998)
5. Saaksvuori, A. and Immonen, A.: Product Lifecycle Management. Springer, Berlin (2008)
6. Jarratt, T. A. W., Eckert, C. M., Caldwell, N. H. M., and Clarkson, P. J.: Engineering change: an overview and perspective on the literature. In: Research in Engineering Design, vol. 22, no. 2, pp. 103-124 (2010)
7. Kääriäinen, J. and Välimäki, A.: Applying Application Lifecycle Management for the Development of Complex Systems: Experiences from the Automation Industry. In: Software Process Improvement, vol. 42, Eds. Berlin, Heidelberg: Springer, 2009, pp. 149-160 (2009)
8. Lindholm, T.. A three-way merge for XML documents. Proceedings of the 2004 ACM symposium on Document engineering - DocEng'04. New York, USA: ACM Press (2004)
9. Do, N. and Chae, G.: A Product Data Management architecture for integrating hardware and software development. In: Computers in Industry, vol. 62, no. 8-9, pp. 854-863 (2011)
10. Brière-Côté, A., Rivest, L., and Maranzana, R.: Comparing 3D CAD Models: Uses, Methods, Tools and Perspectives. In: Computer-Aided Design & Applications (2011)