Research paper:

Managing Complex Systems – Challenges for PDM and SCM

Contact author:
Ivica Crnkovic
Mälardalen University, Software Engineering Dept.
721 23 Västerås, Sweden
ivica.crnkovic@mdh.se,
tel: +46 21 10 31 83, fax: +46 21 10 14 60

Abstract

Software is being increasingly incorporated in increasingly extensive industrial and other applications. There is a demand for total control of entire applications including their software components. A consequence of this is that the development procedure, production operations and maintenance, previously separate processes, are being integrated in comprehensive process systems.

In the integration of these processes, many difficulties are encountered because of different natures of the processes and the different approaches made to the problem. In the integration process, many activities overlap and much data is duplicated, this making a complex process even more complex. Software Configuration Management (SCM) and Product Data Management (PDM) which are used to solve similar problems in different ways are examples of overlapping processes. Attempts to integrate SCM and PDM systems to obtain a more efficient development process have not proved particularly successful.

This paper analyses the main characteristics of SCM and PDM, development processes that are PDM- or SCM-centered, their common characteristics and their differences. The problems encountered when using both systems are analyzed. An analysis of possible integration of these systems is presented and the potential benefits of and problems involved in such integration are discussed.

Keywords: Software Configuration Management, Product Data Management, Development process, tool and process integration

Categories: Software Configuration Management, Product Data Management, Development process.

Managing Complex Systems – Challenges for PDM and SCM

Ivica Crnkovic
Mälardalen University
Software Engineering Dept.
721 23 Västerås, Sweden
ivica.crnkovic@mdh.se

Annita Persson Dahlkvist
Ericsson Microwave Systems AB
Ground Systems Div
431 84 Mölndal, Sweden
Annita.Persson@emw.ericsson.se

Daniel Svensson
Chalmers University
Machine and Vehicle Design
412 96 Gothenburg, Sweden
daniel.svensson@mvd.chalmers.se

Abstract

Software is being increasingly incorporated in increasingly extensive industrial and other applications. There is a demand for total control of entire applications including their software components. A consequence of this is that the development procedure, production operations and maintenance, previously separate processes, are being integrated in comprehensive process systems.

In the integration of these processes, many difficulties are encountered because of different natures of the processes and the different approaches made to the problem. In the integration process, many activities overlap and much data is duplicated, this making a complex process even more complex. Software Configuration Management (SCM) and Product Data Management (PDM) which are used to solve similar problems in different ways are examples of overlapping processes. Attempts to integrate SCM and PDM systems to obtain a more efficient development process have not proved particularly successful.

This paper analyses the main characteristics of SCM and PDM, development processes that are PDM- or SCM-centered, their common characteristics and their differences. The problems encountered when using both systems are analyzed. An analysis of possible integration of these systems is presented and the potential benefits of and problems involved in such integration are discussed.

Keywords: Software Configuration Management, Product Data Management, Development process.

1. Introduction

Product Data Management (PDM) is the discipline of designing and controlling the evolution of a product design. Software Configuration Management (SCM) is the discipline of controlling the evolution of a software product design. These two domains have long been disconnected. They have shared certain common concepts and techniques, but have lived in different worlds. Historically PDM has been focused on hardware development and SCM has been focused on software development. Vendors drive in the PDM domain and researchers in the SCM domain. A trend in both domains is an understanding, for many reasons, of the need for co-operation, especially on the tool side. The use of embedded software in manufactured products is increasing. Companies need comprehensive control of products and their associated software components. A trend in industry today is to manage the entire product and not the hardware and the software parts separately. For an easier environment for users, companies have tried to integrate different systems such as PDM and SCM. We are thus faced with the need to understand both domains.

PDM vendors have ignored software management in their development activities. Similarly, SCM vendors have until recently concentrated on supporting pure software development. In general there is a lack of knowledge in both disciplines, and exhaustive research is needed to determine which integration and interaction methods are most suitable. PDM and SCM users must also decide what they expect to accomplish from such an integrated system and then put pressure on vendors to deliver those capabilities. For vendors and users, the payoffs are likely to be considerable at relatively low-cost and with minimal investment of resources in software management.

This paper analyses the similarities and differences between SCM and PDM and possibilities for their integration. Chapter 2 gives an overview of SCM, outlining the trends in research and industry. Chapter 3 gives a similar overview of PDM. What are the common SCM and PDM characteristics, and what type of use is typical for software development- or hardware development- oriented organizations? Which activities in both domains address the same problem and require a common approach. These questions are discussed in Chapter 4. Although the processes managed by PDM and SCM are similar, there exist several fundamental differences that cause severe difficulties in integration. Which approach can add quality without introducing new complexity and functional overhead? Possible directions of new, integrated approach are discussed in chapter 5. The paper is rounded off with conclusions and outlines for future work.

2. SCM – State of the Art

SCM is about controlling the evolution of complex software. From a management point of view, SCM directs and controls the development of a product by the identification of the product components and their related documents, product structuring, control of their continuous changes, status accounting and auditing. From a developer point of view, SCM maintains current components, stores their history, offers a stable development environment, builds the products and coordinates simultaneous changes in the product. SCM includes both the product (configuration) and the working mode (methods) and the goal is to make a group of developers as efficient as possible in their common work with the product. In many cases, for example in ISO 1007, the term CM (configuration management) is used which is applicable for both software and hardware products. However when specify SCM we emphasize support specific to software development process, which is often unknown for the hardware developers and typically, PDM users. SCM research was traditionally close to industry and has been actively involved in tool development [2,3,4]. SCM is one of the few software-engineering disciplines which has been utilized successfully in practice. More than 50 [5] different SCM tools are currently available.

– SCM treats first of all the complexity of software products. As software became more complex, the number of SCM functions increase. Many disciplines are covered by SCM today, the most important being:

– *Version Control* - the possibility of storing different versions of a software item and being able subsequently to retrieve and compare them.
– *Configurations/Selection* – the ability to create or select associated versions of different items.
– *Concurrency Control* – concurrent development, by either preventing or by supporting simultaneous access.
– *Build* – mechanisms for keeping generated files up to date, rebuilding only those parts of which sources or the building conditions have been changed.
– *Release management* – managing final products in the form of software packages suitable for distribution and installation, and keeping track of product versions in relation to the building components.
– *Workspace management* – support for developers working in a project. Controlling the working versions of the modules being changed.
– *Change management* – a system supporting the management of the collection of change requests, for example in collaboration with customer support, the generation of error reports, firm change requests, implementation of those changes, documentation of the problem and the solution, and when it is available.
– *SCM Process support* – in recent years, especially as a consequence of the emergence of the CMM [6] emergence, the process aspects of SCM activities, i.e. their planning, executing and measuring, became an important part of SCM.

The disciplines developed earlier are closely related to the software development cycle: Editing and building source code. The products of that process, (i.e. binary files) have traditionally not been apart of SCM, since it has been easy to reproduce them. The objects treated by SCM were mostly source code modules. The software development process is more complex today. In addition to traditional software development, there are many other factors managing different type of objects, and with different internal structures and different types of relations. The implication of this is an increase in requirements for automatic management of completed artifacts. The importance of SCM has extended from the implementation phase to earlier phases (requirement and design specifications) and to later phases (assemble, maintenance and support). The previously dominant version management has been superceded by the configuration/selection discipline. In this respect SCM has become less software-specific and more liked a PDM solution.

Another issue becoming important is document management. The basic level of document management can be covered by pure version management in the same way as any source code, but the more sophisticated support requires more functions: Managing differences on the semantic level (impossible with delta-algorithms), and managing the process is different from the source-code management. In this respect SCM has approached PDM. In general, the number of different processes has significantly increased, which requires different level of support. In this respect also SCM has approached PDM. In general, the number of different processes has significantly increased this requiring a different level of support. In this respect SCM has become a more general approach, valid also for non-software artifacts, i.e. to general CM level.

Although SCM is a successful discipline, there are still a number of problems which are cot completely solved. Most of the SCM disciplines works good for ASCII files recognizing lines as internal structures. SCM manages much less efficiently complex objects, and does not recognize the semantic of the objects. For example, programming language semantics is usually unknown to SCM tool. For this reason the interoperability of SCM is becoming more serious problem in the integration process with other tools.

3. PDM – State of the Art

PDM is used to handle artifacts in hardware design, primarily meta-data i.e. data which describes real objects. PDM systems are used to perform CM of hardware, but the scope of PDM is wider than this: "Product Data Management (PDM) is a tool that helps engineers and others to manage both data and the product development process. PDM systems keep track of the masses of data and information required to design, manufacture or build, and support and maintain products" [13]. The PDM systems used today have their origin in the management of documents and product structures. A product structure is used to identify the parts in a product. The first systems used to manage product structures were manufacturing systems, but subsequently design departments began to apply their own systems. When the use of CAD systems expanded, a need to manage CAD models more efficiently became apparent. The vendors of CAD systems offered applications for this. A product structure was used to structure the information in these systems. This was the beginning of modern PDM systems, which can manage not only various kinds of documents and product structures, but also development processes.

- The most important user functions of a PDM system are [13]:

- *Data Vault and Document Management* – documents must be stored in an organized manner. Information about the documents (meta-data) is stored in a central database. Document management routines are used to manage release and change of documents.
- *Workflow and Process Management* – routine processes can be monitored and controlled by a PDM system.
- *Product Structure Management* – product structures, which include description of products parts, are defined and changes in them are controlled.
- *Part and Component Management (Classification & Retrieval)* – standard parts can be classified to support re-use.
- *Project Management* – a large project can be broken down into sub-projects. The progress of a project can be tracked.

- Central utility functions are:

- *Data Transport and Translation* – design is often performed in a heterogeneous environment, with various design tools, on different platforms and at various locations. A PDM system must therefore be able to communicate with various applications and to transfer data between locations. Data created in different applications may need to be translated.
- *Image Services* – most design tools, such as CAD, are only used by the designers. An automatic translation to a neutral format possible to view from any desktop PC makes it possible for anyone in a company to view the geometry.
- *Administration* – hardware design involves numerous participants and many document types. Different users have different access rights to documents which must be handled by the PDM system.

The introduction of PDM systems has not been easy. Many companies are still in the process of selecting and implementing PDM systems. A PDM system will have impact on many of the processes in a company; therefore a company's overall business strategy must be taken into consideration when introducing a PDM system [10]. It is also important to consider organizational issues when introducing PDM [11] in a company.

If researchers drive the SCM (Software Configuration Management), PDM is driven by the needs of industry and by the vendors of PDM systems. There are few researchers in the PDM field as compared with SCM. PDM systems are usually very large and complex and require considerable administration efforts. The usability of such systems is often limited. Their deployment with efficient customization and improved usability is now demanded by industry. Another requirement trend is Internet access and a web-based interface to data to permit people outside a company to access data.

4. Changing paradigms in both SCM and PDM

The characteristics of the two systems emerge from the nature of the artifacts developed. In life-cycle models, PDM is focused more on the design phase and later on the production and maintenance/support phase. The development phase as seen from the software development point of view is less significant. On the contrary, in the software product life cycle, the development phase is usually the most intensive part (despite the intention of software engineering to move more activities to the beginning of the product life cycle), The tools consequently bring into focus support for the corresponding processes.

Figure 1 shows schematically support provided by these tools during the product life-cycle. From the functional point of view, the SCM and PDM tools fit together to completely cover the entire product life cycle. A possibility of integration is even more attractive as the trends in both systems are enlargement of the area of control already covered by the other system. For example,

the CM part is becoming more important than pure version management for the software developers. SCM becomes more similar to PDM due to the structuring and configuration of complex products. On the other hand the development phase, due to extensive use of CAD and simulation tools, becomes more important for PDM users.

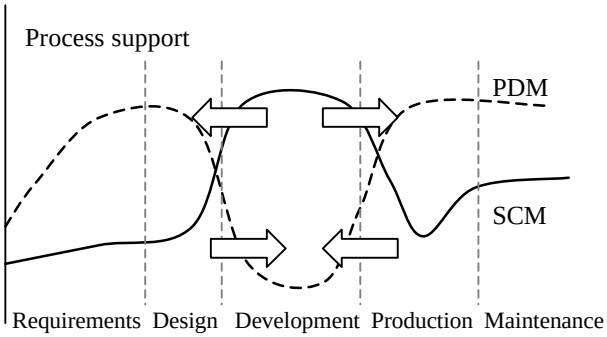

Figure 1. PDM and SCM Process support

In practice, there exist many problems. First, the sharing or exchanging of data between different tools from the same domain but from different phases. Neither system has yet solved this problem completely. A more serious problem arises when data must be shared or exchanged between the tools from these two domains. The second problem is the choice of the tools and methods from the overlapping areas. Even if a particular tool provides excellent support within one domain, it does not mean that it is suitable or well integrated within the second domain.

The overlapping functions are many. Figure 2 depicts the most important functions from both domains. As the figure shows, there are many functions supporting the same or similar process.

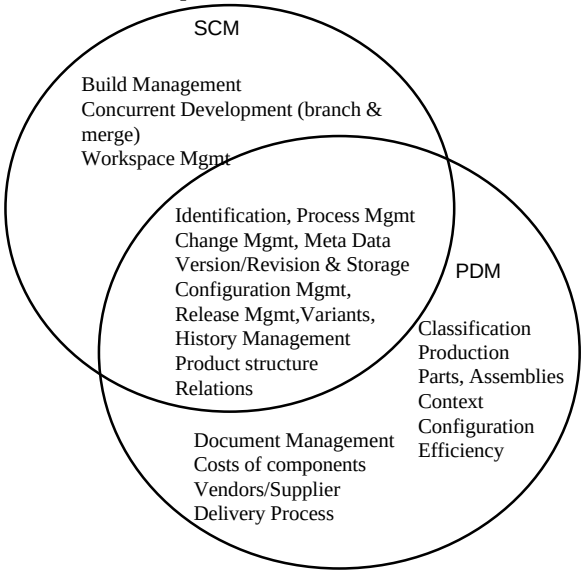

Figure 2. The main functions of SCM and PDM

A further problem is that software developers do not understand the hardware developers and their need of a PDM system, and the hardware developers do not understand the need for SCM tools. The two groups have different opinions on what CM is. There is a need for a common terminology and the semantics to ensure mutual understanding. To find out the real possibility for the tools integration, the analysis beyond the functional level must be done, which is done in the next section.

5. Common Characteristics and Possibilities of Integration

5.1 Common characteristics and differences

Estublier [1] concludes the both domains appear to be very similar, but only on the principle level, while the implementations are very different. To analyze the similarities and the differences the following categories can be compared:

- The product model (data model, configuration)
- The evolution model (versioning)
- The process model

The *product model* includes mechanisms for specification of complex systems. The lowest level of product modeling is data modeling. PDM follows STEP/EXPRESS standard [7, 12]. EXPRESS is an Object-Oriented modeling language defining the static characteristics of complex artifacts. SCM systems do not provide explicit data modeling, but are most often based on an operating system model, i.e. files and directories. The basic principle of product modeling in PDM is the composition relationship in the form of tree structures (examples *part list* or *bill of materials*). Traditionally SCM uses tools such as *make* [8] where relations (dependencies) between different components and the procedure to build them are specified. Data modeling in SCM is very weak. The differences in approach come from fundamental differences in the nature of hardware products: PDM, the product has a physical existence and consists of physical parts. For that reason the product structure is always its part structure. In software there is no such real structure; parts are arbitrary abstractions with loose relationships, and the product structure (application, operating system, platform) exists only in the development phase.

The *evolution model* manages changes during the product life cycle, and is related to version management. The PDM data model, EXPRESS includes no concept of versioning, while this concept is fundamental for SCM data modeling. The focus on version management in SCM

and the neglect of this concept in PDM arise from the differences in the products' natures: Since software may be changed more easily than hardware, SCM must manage versioning in a more sophisticated way than CM for hardware [9]. PDM recognizes three different concepts: historical versioning, logical versioning and domain versioning. Historical versioning is conceptual and similar to SCM versioning, dealing with revisions/versions of a product. Logical versioning manages versions of parts as alternatives, possible substitutes or options. Finally domain versioning is not actually versioning but more the generation of different views of product structures. Similar concepts are included in SCM with emphasis on flexibility of configuration and visibility of differences between versions. The concept of "view" exists in many SCM tools and is related to the selection of a specific configuration. The versioning models in PDM and SCM are tending to become more alike since the objects of management have become similar. Software artifacts, for example documentation and the results from different modeling/design tools, are being managed more and more by PDM. Documentation management is often treated separately in PDM and SCM versioning models are then applied. The problem remaining is the fact that neither PDM nor SCM have solved sophisticated versioning of objects with a complex internal structure (such as documents created by word-processors).

The process model is conceptually similar for both SCM and PDM and can be described by *State Transition Diagrams*. Although STEP should define the standard for PDM systems, PDM providers commonly use extensions to this standard, since STEP is a rather static model, oriented to the final products rather than to the development process. There is however a significant difference between the processes: In PDM, there is a clear distinction between the design and the production process. In software the design and the product are almost the same, and the majority of effort is concentrated to the development/design phase.

To sum up, we can conclude that there are many similarities on the conceptual level between PDM and SCM, but the emphasis on different moments is quite different. The implementations are also different. While PDM is complaint with STEP, SCM does not have established standards. Being too static and too much oriented to the final product management, STEP could not be applied in SCM.

5.2 Integration Possibilities

As many companies are faced with a situation requiring the use of both systems, the question is which kind of integration or cooperation can be achieved with these two systems? A full integration can be achieved by using a common infrastructure, common interfaces and common data. This means that we need a common product model, common evolution model and common process model. A common support for the process model can be used with the present tools, but other models with today's functionality are too different for use as common models.

Another possibility of integration is weak integration with separate infrastructures and data, but a well-defined and efficient interface between the systems.

Figure 3 shows the simplest integration method, building a common application user interface to manage both SCM and PDM functions and present them via common interface to the user.

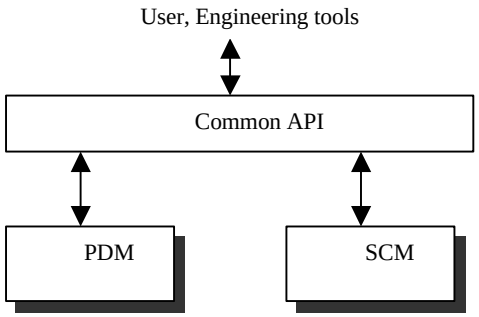

Figure 3. SCM and PDM integration – Common API

This model unfortunately cannot work well with the tools available on the market today. Most of them have a poor API, which provides incomplete (if any) functionality. Another challenge for both tools, independently of each other, is interoperability with other engineering tools. Interoperability requirements will lead to the emergence of better and clearer API's. The new, component-based technology also encourages the use of API's.

As an API for SCM and PDM tools does not provide full functionality, the solution shown in Figure 3 will appear in practice as shown on Figure 4. This solution may generate problems as certain manual actions may introduce inconsistent states for the SCM/PDM combination.

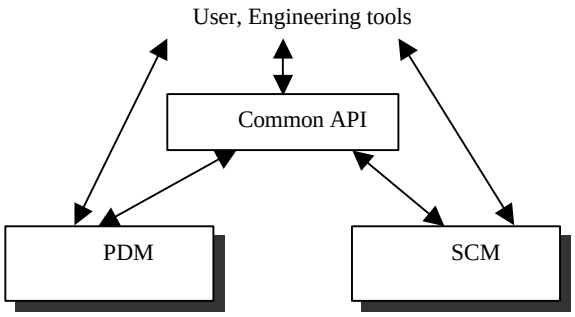

Figure 4. Direct and indirect use of tools

For more robust and efficient integration, SCM and PDM vendors should provide the integration. As PDM covers a larger part of the total product life-cycle and as PDM deals with meta-data (i.e. description and structuring of data), it is natural that communication with the user is via PDM, as shown in Figure 5. Early attempts at this integration, such as integration between Metaphase [15] and Rational ClearCase [14], are already in progress. SCM tools can be integrated with PDM tools as other engineering tools (such as Integrated Development Environment tools) are integrated. PDM tools use API from SCM. Users communicate only via PDM, which is responsible for updating information for both PDM and SCM data. This model provides better control of the consistency of duplicated data. However a similar problem remains. There is already integration of different development tools and SCM tools and it is unrealistic to assume that such integration will not be used independently of PDM integration. This means that there will always be a possibility of data in one database only being modified thereby introducing an inconsistent state. To avoid such possible inconsistencies, a database synchronization process must be included between the databases on a periodical or interrupt/trigger base.

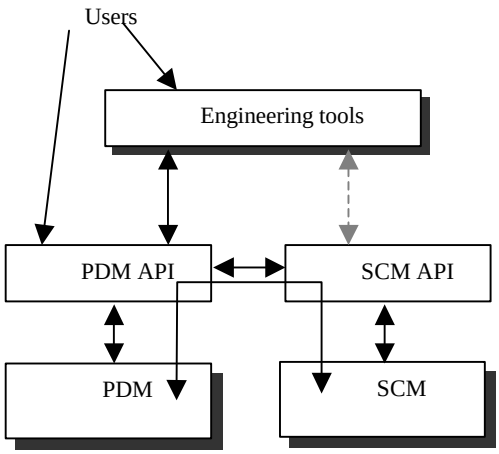

Figure 5. Partial direct SCM/PDM integration

Another problem, which already exists in both systems, becomes more acute in the integration process. Both tools are complex and as a consequence have complex and often user-unfriendly interfaces. When integrated, the system user-interface will be even more complex.

Which parts of the tools can be integrated depends on the tools. The minimal integration required is on the version and configuration level. As PDM does not have flexible mechanisms for version management it is suitable for file versioning to be under the control of the SCM tool. From the PDM point of view, it is more interesting to keep information about specific versions of files collected in a configuration or in a baseline. This means that a list of files (source and executables) containing the pointers to the actual files is saved, as shown on Figure 6.

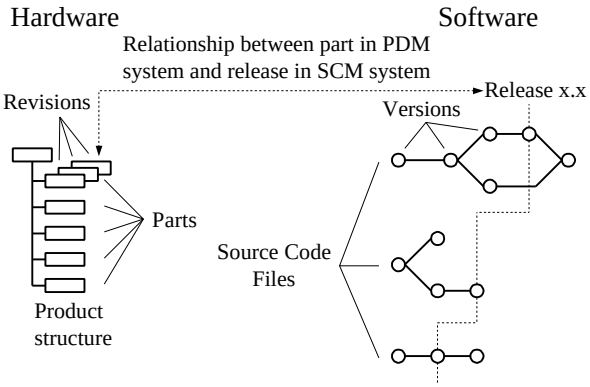

Figure 6. SCM configurations saved in PDM

Workspace management is very important in SCM and in more advanced tools tightly integrated in the entire process. This part must also remain under SCM control, which implies direct interaction between users and SCM, and which deviates from the general intention that one tool should control.

Change management and process management in general can be kept under PDM control. This implies that the change management part in SCM tools should be hidden from users in form of process and action initiation, but kept as triggers to actions and information status inside SCM.

6. A new integration approach

Interaction and integration of the existing PDM and SCM tools meet many problems and it is likely that only partial results can be achieved. The integration will introduce even more complexity and problems with interpretability. The reliability of such integrated system would decrease. As both domains are recognized by

industries as important parts in the development process, and both domains still struggle with usability, interoperability, scalability and cost efficiency, it is not likely that much effort will be applied to integration unless the pressure from users becomes obvious.

The proper solution would be to define a common model. As the processes on the highest abstraction level are very similar, a common data and process model including evolution and interoperability models would be a natural development. Many other engineering tools could use the same model. It should be much easier for one vendor providing several tools to use this approach, but it has been shown that it is a difficult task even for one vendor. A tendency to use this approach can be seen in Rational products, in particular in Rational SoDA [14] in which the results from different tools are collected in one common documentation.

One of the biggest problems encountered in the creation of a common model is the complexity and diversity of objects of management and the inflexibility of the tools which make it impossible to develop consistent coherent models.

New technologies such as middleware, XML, and component-based development are promising in the sense that they permit the building of flexible structures and well-defined interface specifications, allowing different tools to retain their internal structures and processes. There is ongoing work using this approach in the process industry (ABB Objects and Aspects) [16] and research [17], in which two main objectives are considered: flexible and scalable data modeling and a general application interface model. The flexible modeling is achieved by building complex objects with attributes grouped in so-called *aspects*. Each aspect describes a specific role in a certain context of the system. For example, the aspects of a car as an object can be production documents, part lists, web pages, test results, any kind of relevant information. The introduction of aspects (known also as object roles) provides the possibility of obtaining a partial view of an object and the extraction of related information only. The aspects have their own methods for providing information or for performing an action. An aspect can be implemented for example as the document, CAD/CAM drawing, a video sequence, an optimization function or a process control loop. The most important fact is that the aspect's properties need not to be known at system building time, but they are invoked dynamically.

The aspects from different objects (i.e. a configuration of object aspects) can be grouped and managed together. Figure 7 shows an example of structuring aspects, in which the same aspect of an object shares two different structures.

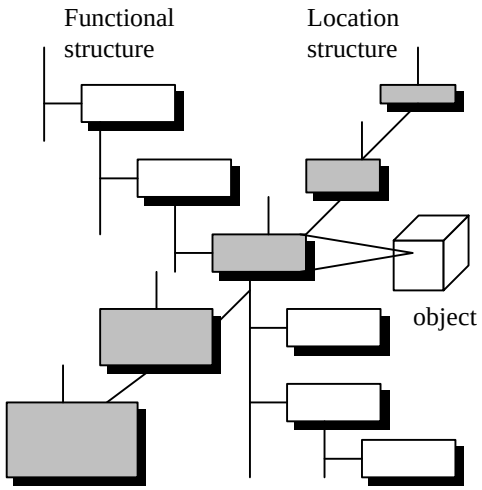

Figure 7. Aspect structures

The concept is similar to implementation of PDM structures, but in this case the structures can be built dynamically. The flexibility of structures is a new feature, absent from previous PDM systems. A second feature is the possibility of communicating with objects by invoking objects' or aspects' methods. The objects provide those methods. Using a standardized framework such as COM/DOCM or JavaBeans makes it possible to dynamically load, invoke and provide results from the objects without knowing in advance the format of data. This approach enables the building of complex systems which are highly dynamic and flexible.

7. Conclusion

Although the trends in system development are toward an integrated approach, in which products are built from both software and hardware, these processes are still separated. One of the reasons is the inadequate integration of tools managing hardware and tools managing software. SCM and PDM systems differ too much to be easily integrated. As the data models and the processes are too different, it is very difficult to achieve a strong integration of the existing systems. This means that instead of having a common database, the repositories will continue to be separate. The integration can be achieved by exchanging data using import/export functions triggered by change of state in databases or invoked through API from users of other engineering tools. When using this approach there is always a risk that data in two systems will not be synchronized. A new approach, covering functions of both systems is required, in which flexibility of data structures and the possibility of dynamically adding new services are

the most important requirements. Middleware technology can be used to achieve the dynamic integration.

8. References

[1] J. Estublier, J-M Favre and P. Morat: “Toward SCM/PDM Integration?”, System Configuration Management, SCM-8, Lecture Notes in Computer Science 1439, Springer, pp. 75-94

[2] J. Estublier, R. Casallas: “The Adele Software Configuration manager”. Configuration Management, Edited by W. Tichy, J. Wiley and Sons, 1994, Trends in Software

[3] G.M. Clemm, Replacing Version Control with Job Control, *Proceedings of the 2bd International Workshop on SCM*, pp 162-169, ACM SIGSOFT, 1989

[4] D. B. Leblang, “Managing the Software Development Process with ClearGuide”, Software Configuration Management SCM-7, Springer, 1997, pp. 66-80

[5] CM Yellow pages, http://www.cmtoday.com

[6] A Systems Engineering Capability Maturity Model, CMU/SEI-95-MM-003, Carnegie Mellon, Software Engineering Institute,

[7] STEP Part 1: Overview and fundamental principles”, ISO TCI194/SC4/WG5, November 1991.

[8] S. Feldman, “Make – A program for maintaining computer programs” – *Software-Practice and Experience*, 9:255-265, April 1979

[9] B. Westfechtel, R. Conradi: “Software Configuration Management and Engineering Data Management: Differences and Similarities”, System Configuration Management, SCM-8, Lecture Notes in Computer Science 1439, Springer, pp. 96-106

[10] S. B. Harris, "Business strategy and the role of engineering data management: a literature review and summary of the emerging research questions", Proceedings of the Institution of Mechanical Engineers: Part B: Journal of Engineering Manufacturing, 210: pp. 207-220, 1996.

[11] P. Pikosz: "Product Data Management in the Product Development Process". Licentiate thesis, Machine and Vehicle Design, Chalmers University of Technology, Göteborg, Sweden, 1997.

[12] D. Schenck, P. R. Wilson: "Information modeling the EXPRESS way". Oxford University Press, New York, NY, USA.

[13] CIMdata: "Product Data Management: The Definition", CIMdata Inc., Ann Arbor, MI, USA, 1998.

[14] Rational ClearCase, http://www.rational.com/products/clearcase/index.jsp

[15] SDRC Metaphase, http://www.sdrc.com/metaphase

[16] ABB Automation, Industrial IT – Objects and Aspects, http://www.abb.com/automation

[17] Erik Gyillenswärd et al: Information Organizer: A Framework for Business Integration, Technical report, Mälardalen University, January 2000