Model Data Management – Towards a common solution for PDM/SCM systems

Jad El-khoury

Royal Institute of Technology (KTH), Mechatronics Division, Machine Design, Sweden
jad@md.kth.se

Abstract. Software Configuration Management and Product Data Management systems have been developed independently, but recently the need to integrate them to support multidisciplinary development environments has been recognised. Due to the difference in maturity levels of these disciplines, integration efforts have had limited success in the past. This paper examines how the move towards model-based development in software engineering is bringing the discipline closer to hardware development, permitting a tighter integration of their data management systems. An architecture for a Model Data Management system that supports model-based development is presented. The system aims to generically handle the models produced by the different tools during the development of software-intensive, yet multidisciplinary, products. The proposed architecture builds on existing technologies from the mature discipline of mechanical engineering, while borrowing new ideas from the software domain.

1 Introduction

Organisations involved in the development of large and complex products need to deal with a large amount of information, created and modified during the development and product life cycle. To support this need, an organisation normally adopts some kind of product management environment. Many such management solutions are currently available, and it is generally the case that each tends to focus on a specific class of products, determined by the major engineering domain involved in the product development. The development of software-intensive products relies on Software Configuration Management (SCM) systems, while mechanical system development uses Product Data Management (PDM) systems.

In the development of products that involve the collaboration of various engineering disciplines, a number of these management environments come into simultaneous use. This is necessary since developers from each discipline require the specific support provided by its corresponding system. An automotive system is a typical such product, where traditional engineering disciplines such as control, software, mechanical and electrical engineering, need to interact to meet the demands for dependable and cost-efficient integrated systems.

Considering the central role these environments take in controlling the development process as well as facilitating the communication between developers, integrating them becomes essential for the successful integration of the efforts of all disciplines involved. In multidisciplinary development, allowing the environments to run unsynchronised creates a source of inconsistencies and conflicts between the disciplines. In other words, it is equally important to provide (where possible) a common set of support mechanisms and principles within, as well as between, the disciplines.

While most of the general facilities provided by these solutions overlap, variations in the details exist due to the differing needs of the domains. This leads to complications and difficulties when attempting to integrate them [1]. In this paper, we discuss how the move towards a model-based development approach in software engineering is bringing it closer to the hardware engineering discipline, allowing for a tighter integration of their management systems. We advocate a common model-based management system that borrows from the technologies of each of these disciplines. In the next section, we discuss the differences between conventional SCM and PDM tools and investigate the effect of adopting model-based development in software engineering in bringing these solutions closer. Section 3 presents a management system architecture that takes advantage of this change. This is followed by a discussion of related work in the area of PDM/SCM integration, before concluding the paper in section 5.

2 Model-based Development – Bringing Software development towards Hardware Development

Model-based development (MBD) refers to a development approach whose activities emphasise the use of models, tools and analysis techniques for the documentation, communication and analysis of decisions taken at each stage of the development lifecycle. Models can take many forms such as, (but not limited to,) graphical, textual and prototype models. It is essential however that the models contain sufficient and consistent information about the system, allowing reproducible and reliable analysis of specific properties to be performed.

With the maturity of the software discipline, the need to move towards a more model-based development approach is being recognized. This need is exemplified in (but certainly not limited to) the OMG efforts [2][3], and the wide range of tools supporting them.

In this section, we will investigate how the adoption of model-based development in software engineering can help bridge a gap between software and hardware development, leading towards a common solution for the data management of multidisciplinary products. In [1], three crucial factors for a successful integration of PDM and SCM are presented: processes, tools and technologies and people. We follow this categorisation in this investigation. New challenges facing such a common solution are also discussed.

2.1 Processes

The difference in the development process of software and hardware products has been most influential in the divergence between their management tools. The more mature hardware development expects support during the complete product life cycle from the early concept design phases down to manufacturing and post-production phases [4]. All product data from all these phases is expected to be handled and related through the PDM system. In comparison, as with any new discipline, early software development occurred in a relatively more ad-hoc manner with no, or little, early design and analysis phases. Consequently, these early phases were beyond the scope of SCM tools [4][5], and SCM was only expected to manage the large amount of source files produced during the implementation phase of software development.

In software engineering, the application of the model-based approach throughout the complete development process implies the need to handle different kinds of documentation from the early design and analysis stages, as well as implementation. Conventional SCM tools have so far incorporated these additional documents by simply treating them as files, without differentiating them from source code files. However, one cannot claim that SCM handles the development process appropriately, since no distinction is made between the types of documents produced during the different development phases. For this to be possible, we argue that the development process itself needs to be reflected in the product information model.

In [1], it is argued that the life cycle processes of the software and hardware development should be integrated for the successful integration of PDM/SCM. The challenges for such integration and a simple solution are then suggested. What seems to be missing in the discussion is how process integration would be beneficial for the integration of PDM and SCM systems. Studying the functionalities of PDM/SCM, one can see that such systems simply provide the infrastructure to enforce a given process (see section 2.2) and play no direct role in integrating the development processes. Instead, PDM/SCM functionalities focus on the product data produced. For this reason, while process integration may be desired within an organisation, for the purpose of integrating PDM/SCM systems, it is even more important to focus on the integration of the outcomes/artefacts produced at each phase of the product lifecycle. The ultimate goal is the tight integration of the hardware and software components of the final product, and not the process of getting there.

2.2 Tools and Technologies

This category is further divided into six basic functionalities expected of PDM/SCM systems: data representation, version management, management of distributed data, product structure management, process support and document management.

Data Representation A major difference between PDM and SCM lies in the kind of data that the support tools are expected to handle [6]. In hardware development, the need to provide a seamless workflow from design to manufacturing phases has forced PDM systems to not only handle the documents produced, but much of their internal contents (metadata) as well. A detailed information model of the product data is an integral part of a PDM system [7]. Software development, on the other hand, has so far adopted a file-based approach, only managing the files produced during development, and where the only relations handled between the files is that of the file system itself (a small amount of meta-data is also handled such as file author and modification date). The internal structure of these files and the semantical relationships between them has so far been outside the scope of SCM tools. PDM can be interpreted as managing product representations, while SCM manages the final product itself [8].

With the maturity of the software discipline, and its move towards a more model-based development approach, many documents (analysis models, uses cases, etc.) will be produced during development. These documents act as models representing certain aspects of the product and will not necessarily end up in the final product. Nevertheless, the different types of documents need to be identified in the management system and related to specific development stages.

The information stored in the documents is interrelated. For this reason, SCM systems supporting model-based development would need to, not only manage the files storing the models, but also the internal content of these models, allowing fine-grained relationships between the document contents to be setup. An information model of the complete information space contained in the models need to be an integral part of a SCM system.

In a model-based development approach, developers need to be shielded from the file structures used to store the models built, allowing them to focus on the models and their structures. This strategy is adopted by many modern modelling tools that may use database systems to store and hide models, and a modern management system should follow in this track.

Version Management In PDM systems, revisions of an object are manually managed by the user and form a sequential series, with no possibility of performing parallel changes. In contrast, versions in SCM systems form a graph structure, with the possibility to perform branching in the development, followed by merging of the branched tracks. Due to these differences, the later approach facilitates concurrent engineering, which is limited in the former.

Accepting that SCM systems need to focus on modelling items, and not only the files storing these items, it becomes essential for version management functionality in SCM systems to similarly focus on the contents provided in these files. Instead of differentiating between the lines of text in different versions of a file, it is differences between the modelling items in different versions of a model that need to be identified and managed.

Since conventional SCM systems do not handle the internal semantics of files, it has also been out of its scope to ensure that parallel changes to the same item (file) are consistent upon a merge. SCM simply provides the mechanism to branch and merge changes made to unrelated lines of text. The burden is placed on the user to ensure that merged changes from different development tracks are consistent semantically. It was hence relatively easy to provide such semantic-free functionality.

Model-based version management becomes a challenge for SCM systems. Complexity arises due to the different kinds of modelling items that may exist in a model compared to the single type (lines of text) that are conventionally handled. It is no longer possible to provide the exact versioning functionalities for all kinds of documents in the system. In the best case, customisation of a generic mechanism will allow the reuse of much of this functionality.

An additional challenge is to ensure consistent parallel changes to the models stored in the files during version management. While lines of text in a file can be treated individually, modelling items in a model are generally tightly interrelated. Changes to one item may have implications on other items in the model. This implies that even though each individual set of changes in two parallel change tracks is semantically valid, merging these changes into a consistent set is not as simple as the union of the changes since the relations between the modelling items need to be taken into account. For example, in a class diagram, one track of changes may have deleted a certain class, while in another track a new association is created between that class and another. In merging these changes, it is first necessary to establish if the deleted class needs to be reintroduced before allowing the presence of the new association.

In dealing with this problem, an SCM system can adopt the approach of PDM of disallowing parallel changes and in this way preventing the problem from occurring in the first place. Another approach is to develop branch/merge mechanisms that work on model structures, maintaining support of concurrent development of models for software developers. A successful implementation of the latter approach can also be beneficial for hardware development, where the possibility to concurrently develop models becomes possible, leading the way for new development processes.

The need for concurrent changes to the same source code files partly originates from the less mature adhoc development of earlier software systems before software "engineering" became a discipline. It is argued that a structured model-based development approach would reduce the need for parallel access to the same product data and hence the former approach becomes more appropriate. In the case where concurrent changes remain a necessity, the latter approach needs to be supported.

Nevertheless, branch/merge mechanisms in SCM remain a necessity for the management of product variants. However, in model-based development, this implicit management of variants should be made more explicit, by representing variants in the product information model.

As discussed in section 4, the model-based approach to versioning and branching/merging is gaining ground in the SCM community. In this paper, we advocate taking advantage of this new trend in the integration of PDM and SCM systems.

Management of Distributed Data The need to manage geographically distributed data seems to be common for both disciplines, with the difference being in the technical solution provided by the management systems. PDM systems provided a more limiting functionality by not allowing concurrent access to distributed data. This difference is closely related to that discussed in the previous subsection, and synchronising the earlier difference will naturally lead to the synchronisation of this functionality. Technically, a common solution will choose either the currently adopted PDM or SCM solution based on whether concurrent access is desired or not.

In a model-based approach to distributed data management, the functionality would focus on the management of distributed fine-grained model data items and not the files storing these items.

Product Structure Management In hardware systems, the physical structure of the final product is the single predominant structure. This structure is used throughout the development phases as a basis for the information model to which all other information is related. Conventional SCM systems do not explicitly support the structure of the product, focusing instead on the directory structure of the files it manages.

In a model-based approach to software development, an SCM system would need to focus on the internal structures of the models stored in the files instead. Unlike hardware products, when using models throughout the development phases, the software structure will vary widely, and hence the product structure management functionality of a model-based SCM needs to handle many different parallel structures. Relationships between these structures will also need to be taken into account.

Given the possibility to manage multiple structures, it becomes easier to also manage products resulting from the integrated effort of hardware and software development. Each discipline would be able to maintain its own structure. The possibility to set up relationships between the structures results in a tight integration of hardware and software components.

Process Support As mentioned in section 2.1, it is necessary to integrate the process support functionalities of PDM and SCM systems. Software and hardware development would need to follow different development processes, and this functionality should be able to support each of the chosen processes, yet based on common fundamental mechanisms: workflow management, user assignment, approach rule mechanisms, etc. As mentioned in [1], such functionality is already quite similar in PDM and SCM systems.

Document Management Document management is an integral part of PDM systems, and such functionality is missing in conventional SCM systems. The need for document management by software developers is apparent, and hence a common efficient support ought to be technically feasible.

2.3 People and Cultural Behaviours

In [9], some of the differences in the terminologies used by software and hardware engineers are highlighted. These differences are attributed to the differences in the development phases generally focused on by these disciplines. For example, in software engineering, “design” is traditionally defined as building a model of the system up to the point at which coding begins. In hardware development, however, “design” would also include broader activities such as requirements and testing activities.

In adopting a model-based approach in both disciplines, and as a by-product of integrating the outcomes of each of the phases of the development processes as advocated earlier, it becomes necessary to integrate the meaning of some of the terminology used.

An important function of models is communication. While models are domain-specific and can only be understood in details by engineers of the specific disciplines, such models can be still used to communicate certain aspects of the design to other engineers, if presented at the right level of abstraction. If models from the various disciplines can be successfully interrelated to form a consistent whole view of the system through a common management system, such interrelations can also act as interaction points between the disciplines, reducing any risks of inconsistencies and conflicts.

2.4 Conclusion

The fundamental differences between SCM and PDM systems stem from the different needs of the disciplines they aim to support. As software development becomes increasingly model-based, and requires support throughout its development life cycle, its needs become closer to those of hardware development. In particular, the process management and information modelling functionalities expected of SCM systems come closer to those provided by PDM systems for hardware development.

This leads the way for an easier and more effective integrated management platform satisfying the needs of both disciplines using a common set of mechanisms. The management functionality ought to take advantage of the commonality between the disciplines – the use of models – in the development process by focusing on models and their internal content as central entities. This allows the same model-based functionalities to be used by both disciplines. We term such an approach as Model Data Management (MDM).

3 Model Data Management

In this section, we present an architecture for a Model Data Management (MDM) system that aims to generically support and control different kinds of models produced from a set of different tools and disciplines.

3.2 Tool Architecture

The envisaged architecture is shown in figure 1. The platform consists of two main parts: A set of tool-specific adaption layers and a data repository with mechanisms to handle this data. The data repository stores the data for each of the tools. To perform this role in a generic way, the data from the different tools is expected to be presented in a neutral form, and this functionality is provided by the adaption layer. Triggered either by a tool or the repository, the corresponding adaption layer permits the data flow between a tool and the repository, in a predefined format. The following subsections will further discuss these components.

Given its maturity, we aim to base the proposed MDM system on a configurable PDM system. The major advantage of using a PDM system is the possibility to define information models, with a high level query language to access and modify the model data in the repository. These facilities generally do not exist in conventional SCM systems. In addition, it is envisaged that the development of the remaining MDM functionalities is made easier given the already developed functionalities of PDM such as the support for distributed development, change management, workflow control, etc. The adoption of a PDM system is not indispensable and one can envisage building an independent MDM that supports both disciplines.

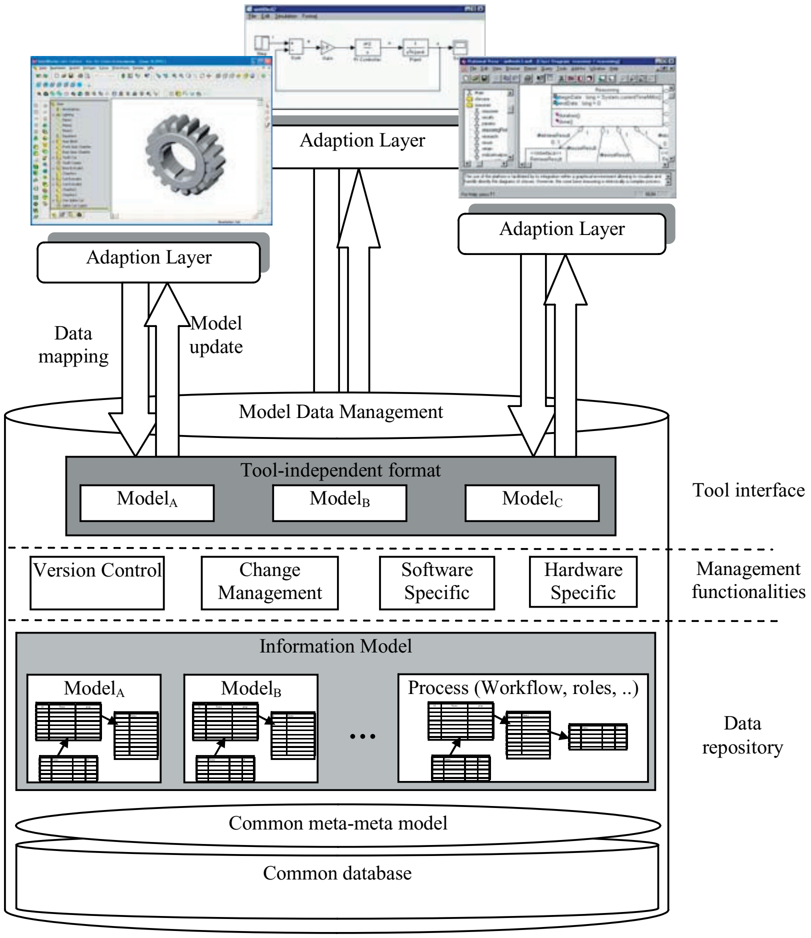

Fig. 1. The major components of the MDM architecture. (Note that the graphical tools are mock-ups shown here for illustration purposes only.)

Data repository The data repository stores the data from each of the tools integrated into the platform. Tool data can be separated into graphical and model data [10] and both types of data need to be managed by the system, giving full control over the models.

It is important to note that the data repository is not expected to be the primary storage medium for each integrated tool, and to which each tool implementation needs to conform. Similar to the a-posteriori approach in [24], an integrated tool is self-sustained, and is only a-posteriori integrated through an adaption layer (See next sub-section).

The content of a model is generally defined using a specific meta-model that reflects its internal structure and constraints of how modelling elements can be combined to form a valid model. In many tools such as in Simulink [11], a meta-model is implicitly assumed, while others, such as any UML tool [3], are strongly based on a given meta-modelling framework.

This meta-model acts as a basis for the data schema used by a tool to internally manage and store the model contents. Similarly, the MDM system managing an integrated model needs to map the corresponding meta-model onto the data schema of the repository. Since different types of models assume a different meta-model, each model type would occupy a separate space in the repository with a different data structure. However, in order to simplify the specification of a schema for each integrated model, a meta-meta-model is adopted as a basis for the repository. This meta-meta-model is instantiated to reflect a given meta-model, which is then further instantiated when mapping the internal data of its tool to the information model of the repository.

We adopt a simple meta-meta-model which generalises among established meta-meta-modelling languages such as MoF [12], Dome [13] and GME [14], and based on a broad survey of modelling languages for embedded computer systems [15]. A model can be generally viewed as consisting of a hierarchical structuring of modelling objects that may possess properties; ports defining interfaces of these objects; and relationships (such as associations, inheritance and refinement) between ports. Modelling languages differ in the kinds of objects that can be specified, their relationships and the kind of properties they possess. When integrating a particular model, a meta-model is instantiated by defining the kind of objects, ports and relations that exist in a model. (Note that the main aim is not to suggest yet another meta-meta-model that claims to cover any modelling language. A simple, generalised meta-meta-model was adopted, allowing focus to be placed on the PDM/SCM integration aspects of the platform.)

Figure 2 shows a UML class diagram of the object types, attributes and relations defining the generic meta-meta-model. As an example, the lower part of the figure illustrates the meta-model of a Data Flow Diagram (DFD) [21] model as interpreted by the Simulink tool [11], which is defined by specialising the generic objects.

In this approach, the granularity at which the MDM system operates on the models is controlled by the definition of the meta-model, implemented in the adaption layer. MDM mechanisms will understand the model semantics down to the level at which the elements, ports and properties are defined. Finer semantics within these entities are not the concern of MDM. For example, if a property of an element is defined as a blob of text, an MDM functionality cannot be expected to interpret the detailed semantics of this property.

Adopting a common meta-meta-model between the models is not sufficient if there is a need to integrate the various model contents into a whole. For this to be possible, a unified information model of the set of models is necessary, specifying more detailed semantics of the models and their interrelations. While such information models are standardised for hardware products [16], no such standard model is currently available that also encompasses models from the software discipline. In [17], ongoing work on how such integration can be achieved is presented.

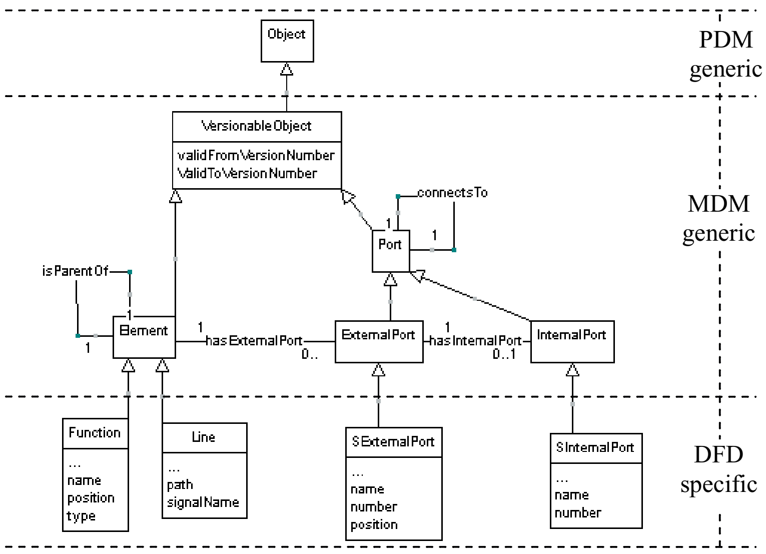

Fig. 2. The PDM information model implementing the meta-meta model, and showing how a tool-specific meta-model is defined.

Tool interface Access to the tool data and the mapping of this data to the repository is performed by an *adaption layer*. An adaption layer is developed for each tool to be integrated into the MDM system. This layer isolates the tool-specific issues allowing MDM to operate generically on many tools implementing different technologies. The adaption layer fulfils three purposes. As discussed in the previous section, it maps the specific meta-model of its designated tool onto the repository.

Second, the adaption layer maps the tool-specific format used internally to manage the model data to a generic format of the repository. In this way, the management functionalities can operate uniformly using a single format.

Different technologies are available for a tool to internally store its model data. A tool can use either a computer file system to store model data in a file, or a database management system. Various standards exist that specify how data should be handled using these technologies, yet one cannot assume that tools will not implement their own solutions.

In a set of tools in which the tools adopt a combination of technologies (standard or not), it becomes necessary to translate these technologies onto a common format, in order to make the interface to the MDM platform generic. This role is fulfilled by the adaption layer, making the tool-specific data technology transparent to the rest of the platform. The adaption layer translates the format used by its designated tool to the chosen format of the repository. In the platform advocated in this paper, we adopt the data neutral XML format to interface the adaption layer to the repository.

Third, the adaption layer accommodates the different techniques used to gain access to the tool's internal data. Different tools use different technologies to provide automated access to its internal data. In the simplest case, the adaption layer can access and interpret the textual file produced by the tool. A tool can also provide 'export' functionality, an Application Programming Interface (API), or a query language.

For a potential tool to be integrated into the MDM system, specific automation support is expected. In order to allow fine-grained accessibility to parts of models and the manipulation of models, a modelling tool whose models are to be managed need to:

- Provide access to the model data either through an API or using parsable text [16].
- Provide fine-grained mechanisms for the construction and modification of models through an API.

Again, in this a-posteriori approach, an integrated tool needs neither to conform to the meta-meta-model, nor format, nor data access approach adopted by the MDM platform. Such demands would create a tight, undesired, dependency between the integrated tools and the platform. It is the adaption layer's role to map these technologies to those of the platform.

Management Functionalities MDM functionalities ought to generically store and handle models from the various tools and disciplines. The functionalities of the union of typical SCM and PDM tools would include: Version management, product structure management, build management, change management, release management, workflow and process management, document management, concurrent development, configuration management and workspace management [4].

The model-based approach to data management unifies the disciplines by unifying the kinds of objects it manages – models. The management functionalities should focus on the models and their contents, transparent of the file structure used to store them.

While as much of the functionalities can be shared by the disciplines, discipline-specific functionalities need still to be supported such as build management for the software discipline. In certain cases, it may also be desired to provide different solutions of the same functionality for different disciplines. For example, software development might require the complex version control mechanisms and concurrent development normally provided by SCM systems, while hardware development is satisfied with sequential revision control. There should be no problem providing different solutions in MDM, depending on the kind of data items the functionality operates on. It would however be advantageous to base the different solutions on generic mechanisms for reusability purposes. The different solutions ought to be also based on the same user interface and terminology.

In order to test the proposed architecture, we have investigated in details the version management functionality of models. This functionality is termed Model Version Control (MVC). While version control is needed in both domains, the functionality differs between SCM and PDM systems (section 2.2). This allows us to investigate how far such mechanisms can be aligned between the disciplines. Version control is also critical since it will put to the test the other crucial factors discussed in section 2, such as the possibility of having a common product structure and data representation. A short summary of MVC is presented in section 3.4.

3.4 Model Version Control Implementation

The MDM platform is currently developed using the Matrix PDM system [19]. As shown in figure 2, it was necessary to specialise the meta-meta model provided by the tool in order to perform the desired version control algorithm.

A simple model version control functionality (MVC) has been implemented. This should be seen as an exploration of the integration possibilities of model-based development. The implementation borrows from the general ideas from the fine-grained version control algorithms such as [5] and [20]. However, instead of using conventional databases, we base our implementation on the MDM architecture.

The algorithm supports the versioning of any model that can be mapped to the meta-meta-model assumed in the platform. In the current implementation, data flow diagram (DFD) [21] models from the Matlab/Simulink [11] tool and Hardware Structure Diagram models [22] in the Dome [13] tool, are handled.

Even though each tool's models contain a different kind of modelling objects, with different set of properties, MVC operates generically on all kinds of models, owing to the adoption layer which presents the model instances using a common format and structure.

Compared to version control mechanisms in conventional SCM systems, the major difference with the model-based approach is that entities have relations between them that also need to be handled. Such relations do not exist between files in the file-based approach. In file-based version control, the versioning of an entity (file) is done independently, and does not affect the versioning of other entities in the system, since no relations exist between them. In contrast, the versionable objects of a model are interrelated and creating a new version of an object might influence the versions of others.

Similar to file-based SCM systems, by only saving the changes between versions of a model, this algorithm maintains efficient storage in the repository and avoids the duplication of information. In addition, comparison between different versions of a model can be efficiently deduced.

MVC provides mechanisms that allow a user to save and extract any part of the system model. In a 'checkin' operation, changes to the model since the last checkin operation are saved in the repository. When performing a 'checkout' operation, the specified element is reconstructed for a given version, together with its subparts, forming an XML document of the information in the repository. This document is then further transformed by the adaption layer to create a tool-specific format that can be used by the tool. The details of these operations are performed transparently to the user, allowing him/her to interface with the modelling tool's interface and format. Further details on the implemented algorithm can be found in [18].

3.5 Integration vs. Unification

As an alternative to integrating PDM and SCM systems as proposed in [4] and [6], the MDM architecture ought to be interpreted as a unified solution that aims to support the needs of both disciplines, assuming model-based development.

The need to move from the file-based approach of SCM to focus on models instead, makes much of the mechanisms currently available technically obsolete. So, the only advantage of maintaining both systems using the integration approach would be to maintain the user interface and terminologies software engineers are accustomed to. Integration techniques struggle with trying to synchronise and balance between the two disciplines.

Instead, the unification approach imposes new common mechanisms with common terminology that are expected to be accepted by both disciplines. Naturally, this approach faces more resistance from established developers and disciplines. However, the shift to model-based development would require a paradigm change that the software community may have to face anyway.

Failures in PDM/SCM integration efforts due to cultural differences [1] ought to be seen as integration problems in the organisation itself that have to be dealt with. In the best case, a unified approach can only bring the conflicts to the surface to be dealt with appropriately.

Accepting the resistance and time it takes tool vendors to change, integration may be the first step, but the future is unification.

4 Related Work

SCM systems targeting models, instead of file objects, are increasingly appearing in the literature ([5], [20] and [23]). In these approaches, an information model of the documents to be handled is assumed, allowing for the management of the internal information stored in the documents, as well as the specification of relations between information from different documents. While focused on software models, these approaches are helpful since the mechanisms can be extended to apply to any kind of models throughout the development lifecycle. The MVC implementation advocated in this paper is inspired by these approaches, broadening their use for more general model types. More importantly, basing the implementation on the facilities already available in PDM systems, instead of using conventional databases, helps in the integration with the mechanisms in the discipline of hardware development.

In [4], three techniques of integrating PDM and SCM systems are proposed. Of these, the full integration technique was considered ideal and most desired. In the full integration solution, the systems' functionalities are separated from their own repositories, and reintegrated into a common repository with a common information model. A common user interface is also built on top, in order to give all users a common look and feel. However, it is argued that full integration is difficult to implement using today's tools due to the tight integration of the tools' components. All the suggested approaches accept the status quo of software and hardware development and consequently needed to deal with fundamental differences. This lead to limited integration success. Rejecting the status quo and focusing on the commonality between the disciplines (model-based development), should instead lead to a smoother integration.

In [6], a configuration management system is suggested that can be applied to both software and hardware design documents. The system also allows for relationships, such as dependencies, to be established between documents. However, the entities handled by the system are file documents with no fine-grained management of their content.

5 Conclusion

In multidisciplinary development, the integration of the various management systems used by different disciplines is of critical value. An integrated environment allows the efforts of all developers to be well communicated and reduces any risks of inconsistencies and conflicts between them.

Due to the difference in maturity levels of these disciplines, such integration efforts has had limited success in the past. Specifically, the implementation-centred development approach of software systems expected a coarse-grained support from SCM systems, where documents are the smallest entities managed, while ignoring the internal model semantics contained within them. In comparison, mechanical development expects the handling of the detailed product data by their corresponding PDM systems using standard information models.

However, with the move towards model-based development, where the use of models becomes the central activity in making, communicating and documenting design decisions, disciplines share a common need to handle the same kind of entities – models. In this way, management systems can be brought closer together.

This paper presented an architecture for a Model Data Management (MDM) system that aims to provide the support functionalities expected of a model-based development environment. The system aims to generically handle and control the various kinds of models produced by the different tools during the development of software-intensive, yet multidisciplinary, products. The proposed architecture builds on existing technologies from the more mature discipline of mechanical engineering, while borrowing new ideas from the software engineering discipline.

To illustrate the MDM solution, an initial implementation of a Model Version Control (MVC) functionality was performed, allowing for the fine-grained version management of two types of models from two different tools. MVC permits stakeholders to perform design activities in terms of models, where they can organise, share and modify their models, transparent to the underlying file structure. A simplified version control functionality has been realised. The ability to perform branches and merges in the changes of an element is a very important feature of version control, specifically desired in software development. This is needed in order to study different design alternative, provide product variants, or deal with a bug fix from an earlier release. MVC needs to handle this functionality in the future.

The major aim of the current platform implementation has been to experiment and illustrate the concepts discussed in this paper. While the current implementation has only been validated through the use of a small case study, a more commercial size case study would be needed to appropriately validate the usability of this approach. This remains to be done in the future.

The advantage of MDM over conventional PDM/SCM systems is the inclusion of the internal content of its supported models, allowing for a tighter integration of the design information between different models. In addition, functionalities are generically applicable for many kinds of models, simplifying the process of adding new tools into the toolset. However, an initial effort is required to integrate new models in the development of the adaption layer. The fine-grained management of models is bound to require more computational effort that the coarse-grained approach.

The development process of software and hardware products will always differ due to the nature of the products themselves. However, in a unified approach the same mechanisms ought to be used to support these differing processes. Moreover, by providing different strategies for different kinds of models, the development needs of both disciplines can be satisfied, using variants of the same basic mechanisms in a unified management system. It is essential however to base the strategies on the same basic mechanisms and user interface, allowing the reuse of basic components and preventing confusion in terminologies.

In the case where development is not (completely) model-based, MDM facilities may still be used. Any product data inputted into the platform is restructured and interpreted to form model data. For example, a MDM system can manage the files of a Java project by reinterpreting each file as a class model, extracting and managing the attributes and methods contained within each file as fine-grained structured data.

The approach is currently implemented using a PDM system. It is our ideal vision that with the acceptance of model-based development, one no longer needs to discuss the integration of PDM and SCM systems. Instead, a truly unified approach to model data management can be used by both disciplines.

6 Acknowledgements

This work has been supported by the Swedish Strategic Research Foundation, through the SAVE project.

References

1. Dahlqvist, A.P., Crnkovic, I. and Asklund, U., Quality Improvements by Integrating Development Processes, 11th Asia-Pacific Software Engineering Conference, 2004.
2. OMG, Model Driven Architecture Specification, MDA Guide Version 1.0.1, Document Number: omg/2003-06-01, June 2003.
3. OMG, Unified Modeling Language (UML) Specification, V1.5, March 2003.
4. Crnkovic I., Asklund U. and Persson Dahlqvist A., Implementing and integrating product data management and software configuration management, Artech House Publishers, 2003.
5. Ohst D. and Kelter U., A fine-grained version and configuration model in analysis and design, Proceedings of the International Conference on Software Maintenance, 2002.
6. Westfechtel B. and Conradi R., "Software Configuration Management and Engineering Data Management: Differences and Similarities" Proceedings 8th International Workshop on System Configuration Management, Springer-Verlag, pages 95-106, 1998.
7. Kemmerer S. J. (editor), "STEP, the grand experience", National Institute of Standards and Technology, special publication 939, 1999.
8. Estublier J., Favre J. M. and Morat P., Toward SCM / PDM integration?, International Workshop on Software Configuration Management, (SCM8), Springer Verlag, 1998.
9. Kruchten, P., Casting Software Design in the Function-Behavior-Structure Framework, IEEE Software, Volume 22, Issue 2, 2005.
10. Ohst D., Welle M. and Kelter U., "Differences between Versions of UML Diagrams", Proceedings of the joint European software engineering conference (ESEC) and SIGSOFT symposium on the foundations of software engineering (FSE-11), 2003.

11. Simulink, Mathworks, http://www.mathworks.com/products/simulink/, accessed March 2005.
12. OMG, Meta Object Facility (MOF) Specification, V1.4, April 2002.
13. Dome, “Dome Guide” Version 5.2.2, http://www.htc.honeywell.com/dome/index.htm, 1999.
14. GME, A Generic Modeling Environment, GME 4 User’s Manual, Version 4.0, Institute for Software Integrated Systems, Vanderbilt University, 2004.
15. El-khoury J., Chen D. and Törngren M., “A survey of modelling approaches for embedded computer control systems (Version 2.0)” Technical report, ISRN/KTH/MMK/R-03/11-SE, TRITA-MMK 2003:36, ISSN 1400-1179, Department of Machine Design, KTH, 2003.
16. Kemmerer S. J. (editor), STEP, the grand experience, National Institute of Standards and Technology, special publication 939, 1999.
17. El-khoury J., Redell O. and Törngren M., A Tool Integration Platform for Multi-Disciplinary Development, to be published, 31st Euromicro Conference on Software Engineering and Advanced Applications, 2005.
18. El-khoury J and Redell O., A Model Data Management Architecture for Multidisciplinary Development, Internal Technical Report, Mechatronics Lab. Royal Institute of Technology, Stockholm. 2005.
19. MatrixOne, Matrix10, http://www.matrixone.com/, accessed April 2005.
20. Nguyen T. N., Munson E.V., Boyland J.T. and Thao C., Flexible Fine-grained Version Control for Software Documents, 11th Asia-Pacific Software Engineering Conference, 2004.
21. Cooling J., Software Engineering for Real-time Systems. Pearson Education Limited, ISBN 0201596202, 2003.
22. Redell O., El-khoury J. and Törngren M., The AIDA toolset for design and implementation analysis of distributed real-time control systems, Microprocessors and Microsystems, Volume 28, Issue 4, 2004.
23. Chien S. Y., Tsotras V. J., Zaniolo C., Version Management of XML Documents, Third International Workshop WebDB 2000 on The World Wide Web and Databases, 2000.
24. Becker S. M., Haase T. and Westfechtel B., Model-based a-posteriori integration of engineering tools for incremental development processes, Journal of Software and Systems Modeling, Volume 4, Number 2, Springer, 2005.