RC23613 (W0505-109) May 19, 2005
Other

IBM Research Report

System Data Management: An Inter-Disciplinary Collaboration Architecture for Systems Engineering

José Gomes, Man-Mohan Singh[1], Mila Keren[2], Sai Zeng, Julia Rubin[2], Laurent Balmelli, Ioana Boier-Martin
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[1]IBM Software Group

[2]IBM Haifa Research Laboratory

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

SYSTEM DATA MANAGEMENT: AN INTER-DISCIPLINARY COLLABORATION ARCHITECTURE FOR SYSTEMS ENGINEERING[1]

José Gomes
josegome@us.ibm.com

Man-Mohan Singh
mmsingh@us.ibm.com

Mila Keren
keren@il.ibm.com

Sai Zeng
saizeng@us.ibm.com

Julia Rubin
mjulia@il.ibm.com

Laurent Balmelli
balmelli@us.ibm.com

Ioana Boier-Martin
ioana@us.ibm.com

IBM Research

ABSTRACT

This paper presents a novel approach to integrating systems engineering (SE) artifacts and methods with discipline-specific detailed design artifacts and processes, for the purpose of facilitating inter-disciplinary collaboration. In particular, it addresses the lifecycle management of complex products involving mechanical, electrical, electronics and software aspects, and being designed following a formal product development methodology. The primary motivation of the approach is to capture and maintain the traceability between the concrete artifacts stored in discipline-specific "repositories" and the abstract artifacts used to support system-level decisions as well as system integration. The proposed approach acknowledges the fundamental differences that exist between the various engineering disciplines and therefore favors a loose coupling based on a process-centric management of the artifacts traceability links. These considerations lead to an inter-disciplinary collaboration and infrastructure pattern called "system data management" (SDM), with the role of enforcing the integrity of the inter-disciplinary traceability between artifacts. As a byproduct, this approach suggests a novel perspective on product data management (PDM) and software configuration management (SCM) integration that sharply contrasts with point-to-point integration solutions. The authors have implemented a prototype based on a service-oriented architecture (SOA) and existing PDM and SCM technologies.

Keywords: Inter-disciplinary collaboration; integration of product data management and software configuration management systems, support of systems engineering practices, systems modeling language, conceptual design and detailed design artifacts.

1 Introduction

This paper is concerned with the methods used in the Manufacturing Industry to develop complex products such as passenger vehicles, aircrafts, ships or satellites. Historically, for the sake of economic rationalization, the industry has evolved into complex intertwined networks of original equipment manufacturers, specialized suppliers, and even more specialized suppliers of suppliers. The product development methods themselves have evolved to adjust to this organizational model and rely heavily on component integrations.

Today, the industry continues to evolve in response to changing market conditions. One of the most important changes, currently taking place, is the increasing reliance on embedded software content in new products. Embedded software is increasingly being used to manage complex mechanical and electronic components and to implement unique new product functions and features. This trend is evident from the need of manufacturers to differentiate their products from their competitors and the increase in customer demand for new electronic features. This new trend implies that there is an increased focus on new integrated product development paradigms. In the absence of such paradigms, the manufactures are subject to tremendous risks in costs associated with delays in new product introduction, warranty and liability.

The increased focus on integrated product development is of special interest because, in the recent past, development organizations across various design disciplines have worked in silos and differ in terms of culture, practices, processes, tools, representations, and concrete artifacts. Basic paradigms of software engineering are quite different from those of the traditionally dominant mechanical and electrical engineering disciplines.

[1] **To appear in the Proceedings of ASME IDETC/CIE 2005.**

A noticeable trend in the industry, moving in the direction of integrated product development, is to consider the promises of Requirements Engineering, Systems Engineering or Model-Driven Development disciplines as applied to the development of a "whole" product. Various systems engineering approaches to product development put emphasis on requirements engineering and inter-disciplinary collaboration throughout the product lifecycle while model-driven development emphasizes creation of product models as seen from the perspective of its stakeholders. The product is seen as system delivering measurable value as a whole, rather than as the sum of its constituents.

The proponents of such holistic methods claim that raising the level of abstraction above traditional engineering disciplines has the potential to connect the manufacturer's business drivers to detailed development processes. An additional conclusion that can be asserted is that these new product models, or abstract artifacts, can themselves become valuable reusable assets for the manufacturers.

Objectively, one could also argue that it is not clear why the claimed benefits would counter-balance the cost of managing an even higher number of disciplines, associated artifacts, and processes. A second equally valid concern is that abstract models "on the paper" may not reflect the day to day reality of engineering processes and therefore may not be relevant to effective decision-making processes.

While multiple studies confirm the business benefits of holistic methods [11,21,22], relatively little attention has been paid to the second concern, and this is the focus of this paper. It is clear that abstract models, such as Systems Modeling Language (SysML) models of requirements, products and processes [23, 1], or any other abstract representation of systems, can only be useful if they faithfully reflect the realities of the day. For instance, design tradeoff analysis or impact analysis of design changes are key systems engineering processes for effective decision-making. However, the results of such analysis are only as accurate as the connection between the system model and the associated concrete artifacts representing processes and domain specific knowledge.

In this paper, we call "traceability" the associations between artifacts needed to support specific systems engineering processes through multi-disciplinary collaboration and leading to decisions. We characterize this notion of traceability in detail and describe a conceptual framework for implementing it.

2 State of the art

A central aspect of the holistic approaches to product development involves maintaining the integrity and evolution of inter-relationships of various discipline- or stakeholder-specific abstractions, or artifacts, during the product lifecycle. This is a necessity if one has to effectively apply the systems engineering methods for product development. Maintenance of this traceability of artifacts is achieved through inter-disciplinary collaborations.

In the traditional product lifecycle management (PLM) approach to product development, product data management (PDM) or engineering data management (EDM) is the discipline of designing and controlling the evolution of a product from its mechanical, electrical, and electronics engineering aspects, whereas, software configuration management (SCM) is the discipline of controlling the evolution of software engineering aspects of the product. The artifacts of requirements engineering are either managed in PDM or SCM systems, or more often, are not afforded formal treatment. At a conceptual level, there are many similarities between PDM and SCM; however, there are also major differences [10, 26, 6, 7] because of the different nature of the supported artifacts and processes.

When applying a systems engineering approach to product development, there have been attempts to manage software engineering artifacts within PDM systems and, similarly, to manage mechanical and electrical engineering artifacts in SCM systems but these solutions have not been very satisfactory or successful. PDM tools have been on the market for decades with a long tradition and standardized product evolution control know-how. They are strong in product modeling, data representation (metadata and data are separated), data modeling (object-oriented data models), and workflow, process and document management. However, they have weak support for concurrent engineering and workspace, build and configuration management [10]. SCM tools are more recent than PDM tools. They are good at managing files and directories and supporting concurrent engineering. They are good at version, build, configuration, and workspace management. However, they are weak in product modeling and release and document management [10].

An alternate approach that has been tried is the point-to-point integration between specific PDM and SCM products by invoking import/export functionality of the underlying databases to maintain integrity of the two different product data representations [7]. However, such integration efforts have turned out to be very complex in nature and there is a high risk that the data integrity across the two systems cannot be guaranteed.

Looking at the problem from a model-driven development standpoint, yet another approach seems natural. For example, assume that artifacts are kept within their native development environments, for instance, hardware in PDM and software in SCM. Further, assume that the overall product is described in a system model; it is tempting to implement inter-disciplinary traceability by directly associating each system model element with the relevant concrete artifacts in other disciplines. One limitation of this solution, though, is that it is difficult *a priori,* to decide which concrete artifacts are associated with a given system element. In reality, the associations mainly derive from the specific product development practices in use. There are potentially as many different associations of very different nature as there are different processes involving the same system model element. Additionally, using the system model of the product as the root of all traceability links prohibits re-use of the model. For instance, part of a system model could be reused in different parts of a product, or even in different products. This situation makes it clear that the system model itself cannot be the root of the traceability to the concrete artifacts.

Even though inter-disciplinary collaboration is the central tenet of systems engineering, the state of the art to achieve such collaboration is too weak to be effective. We therefore propose

an alternative approach to supporting inter-disciplinary collaboration.

3 System Data Management Concept

System data management (SDM), as defined in this paper, is a collaboration and infrastructure pattern. Its role is to support inter-disciplinary collaboration processes so as to maintain the integrity of a system that is specified by a set of product development artifacts concurrently authored by multiple teams. This concept and its hypotheses are described in this section.

3.1 Intra and inter-disciplinary collaboration

The authors assume that product development processes may be classified as intra-disciplinary or inter-disciplinary. An intra-disciplinary process is executed by a development team within a single discipline and involves authoring artifacts of this discipline only. Figures 1 and 2 represent a typical intra-disciplinary product development infrastructure and collaboration pattern at a level of detail sufficient to introduce the SDM concept. Development artifacts are under the revision control of one or more discipline-specific repositories. This is, for instance, the basic pattern implemented by both PDM and SCM systems. The development environment of a team includes a workflow client used to involve the team in a prescribed product development process, a repository client used to reserve and commit artifacts to the common repository, and authoring tools used to create or modify artifacts in the workspaces of team members.

In contrast, an inter-disciplinary process is executed by a multi-disciplinary team and involves the coordinated concurrent authoring of artifacts belonging to different disciplines so as to guaranty the integrity of the system being developed. Examples of inter-disciplinary processes are those prescribed by systems engineering best practices. Specific examples are requirements coverage analysis ("Is this requirement being taken into account in the design?"), requirements conformance analysis ("Is this requirement actually satisfied by the design?"), impact or tradeoff analysis ("What if?"), and system-level requirements verification. Unlike intra-disciplinary collaboration, there exists no *de facto* inter-disciplinary collaboration and infrastructure pattern.

The relevance of this classification of development processes (*i.e.* intra and inter-disciplinary) arises in particular from growing evidence in the industry that most quality concerns of modern products primarily result from poor inter-disciplinary collaboration [3].

3.2 Process-centric support of inter-disciplinary collaboration

SDM therefore exclusively aims at supporting inter-disciplinary collaboration and is a collaboration and infrastructure pattern. The authors assume that the pattern depicted in Figures 1 and 2 adequately supports intra-disciplinary collaboration within any individual discipline. To understand the concept of SDM, it is necessary to know the reasons that this pattern is not appropriate for inter-disciplinary processes in practice. There are at least two reasons.

A careful examination of existing PDM and SCM systems [10, 26, 6, 7] shows how qualitatively different discipline-specific implementations of this pattern are in practice. The detailed data models, evolutionary models and interfaces of each repository arise from specific needs and development paradigms of its discipline. These are often not known by developers of other disciplines, neither should they need to be known in general. In these circumstances, it might be unrealistic to reconcile several data models and associated practices into a universal kind of repository and practice, as required by this pattern. This motivates the need to investigate inter-disciplinary collaboration and infrastructure patterns that will respect these multiple *de facto* discipline-specific infrastructures and leverage them "as is", if possible.

Another argument is the existence of strategic investments in PDM and SCM implementations in organizations traditionally dominated by a specific engineering discipline. The investment in skills and processes which use these infrastructures provides additional reasons to consider an integration approach that does not violate, but instead leverages, the discipline-specific infrastructures in place.

To understand the concept of SDM, it is also necessary to know its positioning with respect to systems engineering artifacts and practices. The discipline of systems engineering, in many regards, does support inter-disciplinary collaboration. In effect, its artifacts, such as models of systems, allow the joint expression of mechanical, electrical, electronics, software and requirements aspects in a common language such as, for instance, SysML. This offers a basis for reasoning, and therefore making decisions, on the systems structures and behaviors which will best satisfy all the product and product development requirements. However, from the artifacts lifecycle management perspective, systems engineering is merely another discipline. Let us illustrate this point. The set of all development artifacts associated with a given product typically contains artifacts representing its individual components, and artifacts representing the integrated system. The first subset might contain for instance Computer Aided Design (CAD) or Computer Aided Engineering (CAE) documents, specifications, software code, or test cases. The second subset might contain SysML system models or any other description of how all these components *should* fit together to form the whole.

However, an important piece of information still missing is the set of concrete links existing between all these artifacts, if they are seen as different resources kept under revision control in specific repositories. The SysML model does not contain this information and, from this perspective, is merely such as resource itself. This missing *traceability* information is necessary so that the abstract systems model can accurately reflect all the aspects of the product and lead to correct product development decisions.

3.3 SDM conceptual model

This section introduces the SDM conceptual model of collaboration, beginning with a formal description and supporting it with an example.

Under the SDM data model, discipline-specific artifacts remain under the management of their respective teams and repositories. The inter-disciplinary traceability information is explicitly captured and managed in an additional discipline-

agnostic repository called the *SDM Repository*. Within this repository, the traceability information is organized and managed according to prescribed inter-disciplinary processes associated with the system being developed. For each such process, the SDM repository maintains references to the set of multi-disciplinary artifacts (seen as concrete resources) needed to execute it. The integrity of the traceability information is guarantied through the enforcement of a collaboration protocol between discipline-specific repositories and the SDM repository.

For illustration, assume that a given testing process *P* consists of "loading" a specific version of a software component *S* on a specific revision of a hardware component *H*, executing the software program in the context of a specific version of a use case *UC* and verifying that some specific versions of associated performance requirements *PR* are satisfied. This process is assumed to be prescribed by the methodology of the development organization and needs to be executed multiple times during the development phase. Each specific version of *P*, *S*, *H*, *UC* and *PR* takes the form of a file document, database records, or any other digital form. These artifacts are possibly stored in different repositories and managed by different organizations. Since each of these constituents participate to the specification of a system, the logical relationships between *S*, *H*, *UC* and *PR*, as intended by a systems engineering team, are specified in a SysML model *Sys*, or any other systems modeling language. Seen from the perspective of the SDM data model, *Sys* is yet another artifact, stored in yet another repository and managed by the systems engineering team. Next, taking the perspective of an inter-disciplinary team in charge of executing the testing process *P*, it is necessary to know how to access and reserve each of these artifacts in their individual repositories. Therefore, for the sake of both efficiency and correctness, there is a need to record the association between the process to be executed and the artifacts required to execute it, so as to enable the team to involve the "right" artifacts at the "right" time. This is illustrated in Figure 5, in which each lower circle represents all the information needed to access an artifact in its own repository, *e.g.,* PDM or SCM.

The set of references to the artifacts in their respective repositories (or *artifact proxies*) that are needed to execute any inter-disciplinary process is called the *artifact tuple* associated with the process, in our example the testing process. This artifact tuple is stored in the SDM repository, along with other tuples associated with other processes. From a lifecycle management perspective, the artifacts referenced from the artifact tuple pursue their individual lifecycle, *e.g.,* go through their next revisions. Therefore, the references stored in the tuple need to be updated accordingly. The artifact tuple itself is updated whenever any of its referenced artifacts is changed. Therefore, the artifact tuple needs to be kept under revision control in the SDM repository. In other words, the artifact tuple is an artifact itself. This new artifact, now part of the set of artifact describing the system, captures process knowledge regarding the development of the system. Indeed, the artifact tuple will facilitate the re-execution of this process in the future. Furthermore, since the artifact tuple *is* an artifact, it might be referenced from a parent artifact tuple as well. For instance, in a different scenario, a "large" testing process may involve not just direct concrete artifacts of the system, as in our prior example, but may also need as input the outcome of a number of smaller tests. In this case, a parent tuple associated with the parent test process may reference child artifact tuples associated with child testing processes. Therefore, hierarchies of artifact tuples, or more general graphs of them, capture arbitrarily complex process knowledge of development processes associated with a system. This process knowledge is now part of the set of development artifacts that describe the system.

In particular, this can be particularly useful to raise the level of granularity at which reuse is performed from the level of individual components to that of entire subsystems. Indeed, if care is taken in packaging the set of artifacts associated with a subsystem so that this process knowledge always remains attached with it, then artifact tuples may facilitate the re-execution of customization and integration processes associated with that particular sub-system and therefore effectively support its reuse in different contexts and products. In some sense, SDM defines system reuse as a process.

Note that, *a priori*, organizing the artifacts traceability in such a process-centric manner is not the only possibility. For instance, one could "directly" capture the information that component *H* and component *S* are associated by creating and managing a link from *H* to *S*. However, the role of this relationship is not clear as one considers practical use cases. In effect, the same hardware component *H* could, in principle, be as well associated with another software component S_2 in other products of the portfolio. Then, by induction, any component artifact could be "directly" linked to an arbitrary number of other components artifacts. It is not clear then how these links may be distinguished from one another by a team in charge of executing a process involving these artifacts. The same argument would apply if one would attempt to "directly" capture the traceability between *Sys* and the discipline-specific artifacts. In practice, traceability links always depend upon a specific process context, and their only reason to exist is to support this process. This motivates the process-centric approach taken by the authors. Let's now turn to the challenges of this approach.

Supposing that artifact tuples are stored in the SDM repository and their referenced resources are each managed in their own repositories, an obvious challenge is to keep the links up to date as each referenced artifact goes through its next steps in its evolutionary model (cf. Figure 5). "How does the SDM repository know that component *H* has now a new version?" This problem can be solved if communication takes place between the SDM repository and the discipline-specific repository owning component *H*. Without this transfer of information, *P* might be performed using outdated artifacts. In SDM, the integrity of artifact tuples is guarantied by a simple, but strictly enforced request/update collaboration pattern between the SDM repository and the various discipline-specific repositories. This requires a minimal but necessary alteration to each of the discipline-specific development environments: the addition of a light infrastructure component providing "SDM Client Services" that enforce the communication protocol each time "Check In" or "Check Out" requests are made by a development team. Each instance of this component handles the negotiation with the SDM repository that exposes its "SDM Core Services". This alteration can be made transparent to the team members by programmatically routing requests to reserve

or commit artifacts to the SDM Service, instead of to the repository client. In the case of a non-integrated development environment (IDE), the SDM Client Services may as well be exposed to end-user using an additional trivial graphical user interface (GUI).

In more detail, and referring to the sequence diagram of Figure 4, the request/update collaboration pattern functions as follows. Whenever a team member requests to reserve an artifact, its "local" instance of the SDM Client Services needs to request authoring permission to the SDM Core Services, before it may actually access the artifact within its own "local" repository. As a result of this request, the SDM Core Services return detailed access information that is used by the SDM Client Services to reserve the artifact in its own "local" repository. Similarly, when the same team member requests to commit the artifact after having updated it, the same instance of the SDM Client Services first commits the artifact to its own "local" repository and then needs to notify the SDM Core Services of the new access information to be used for the next access of this artifact. The SDM Core Services update the traceability data kept under revision control in the SDM Repository. This alteration of the infrastructure and collaboration pattern is depicted in Figures 3 and 4.

A second challenge is that artifacts in different disciplines might *a priori* follow different evolutionary models. This could be a problem since the details of the negotiations between the SDM Client Services and the SDM Core Services that track these evolutions would in this case differ, depending on the discipline. For this reason, SDM assumes the most general evolutionary model. One observes indeed that there are two basic evolutionary models: linear models (mostly adopted by PDM) and tree models (mostly adopted by SCM). The motivation for tree models is to have multiple lifecycle branches running simultaneously and allowing, for instance, multiple teams to concurrently update the same artifact. Historically, this evolutionary model became plausible in software engineering thanks to the possibility of analyzing incremental differences between two text files, or for that matter software source code, and merge them into a third file containing all the differences. The difference and merge capabilities have been extended to many other formats, including that of visual models but is not a standard operation for most artifacts of other disciplines. However, since the tree evolutionary model is a generalization of the linear one, in SDM, it may be assumed, without loss of generality, that all the artifacts follow a tree evolutionary model. Possible applications of such a general lifecycle model at the system-level include the simultaneous investigation of design alternatives or the simultaneous resolution of independent design issues across multiple disciplines. For instance, two pairs of software and hardware teams could, in principle, simultaneously investigate two different ways to solve a system design issue involving changes in both software and hardware. It must be noted that although branching models are not supported by traditional PDM practices, the actual underlying technologies of PDM support may support arbitrarily general evolutionary models thanks in particular to the notion of "objects primary identifiers", which may be used to give multiple identities to the "same" object.

4 System Data Management Proof Of Concept

The authors have developed a prototype SDM infrastructure based on a service-oriented architecture (SOA) and existing PDM and SCM technologies. The purpose of this prototype is to provide an early validation of the concept of SDM in the context of hypothetical use-cases, prescribed system-engineering methodology and discipline-specific product development environments. This prototype is not intended as a reference solution but merely as an example. In effect, each development organization has its own prescribed product development methodology and environments. Therefore, SDM must be customized accordingly. SDM being technology-agnostic and methodology-agnostic, its hypotheses of applicability are largely met by modern development organizations.

4.1 Use-cases

The authors have defined a set of automotive use-cases focused on the reconciliation of system-level requirements through inter-disciplinary collaboration. The sub-systems of interest are the brake system and the adaptive cruise control, with a focus on mechanical analysis and embedded systems. The addressed requirements include hypothetical cost requirements and the publicly available regulatory requirements of the National Highway Traffic Safety Administration (NHTSA) of the United States for passenger vehicle brake systems [13]. According to one use-case, for instance, a team of systems engineers evaluates possible design changes targeting a brake system cost reduction. This use-case involves the work of mechanical engineers in charge of verifying brake system thermal-stress requirements, of software engineers in charge of adapting existing embedded software of various functions (Anti-lock brake system (ABS), electronic stability program (ESP), traction control system (TCS) and adaptive cruise control (ACC), [7]), and of requirements engineers in charge of managing the corresponding cost and safety requirements.

4.2 Prescribed methodology

The assumed prescribed methodology is the Rational Unified Process for Systems Engineering [17]. Note that any methodology explicitly prescribing inter-disciplinary collaboration processes is an equally good candidate to apply SDM and since artifact tuples may be created on-demand, detailed processes need not be defined in advance.

4.3 Discipline-specific environments

Each engineering discipline has it own specific development environment, in agreement with Figures 1 and 2. The environments used in the prototype are summarized in Table 1. The authors have installed, configured and populated these environments in their laboratory so as to approximate a representative production environment as closely as possible.

4.4 System data management deployment

The authors have implemented the data model of Figure 6 using PDM technology, used as a basic combination of object-oriented and database technologies. This data model is logically independent from that of usual PDM data models such as those used for the Bill of Material (BOM). Indeed, from a logical perspective, PDM systems are the intra-disciplinary repositories of choice for mechanical and electrical engineering, while the SDM repository is inter-disciplinary. From a physical deployment perspective, though, SDM can be implemented using existing technologies, and even use the same PDM engine as for the BOM. Other candidate technologies for implementing the SDM data model are object-oriented SCM, stellation technology or object-oriented stores persisting data in relational databases, such as in the Enterprise Java Beans technologies [18, 4].

Figure 4 defines functional interfaces between the various environments. It is therefore natural to select the SOA integration paradigm [19], which focuses on the definition and deployment of such interfaces and not in the specifics of their implementations. Each discipline-specific environment and PDM repository exposes "web services", or possibly "enterprise services" that invoke one another.

4.5 Comment

Based on the experience gained with this prototype, the authors conjecture that the services (or network interfaces) needed in deployments of the SDM architecture in actual production environments may be classified as in Table 2. This classification provides additional guidance for future SDM deployments.

Table 1 Disciplines considered in SDM prototype.

Discipline	**Role**	**Discipline-specific authoring tool**	**Discipline-specific repository**
Systems engineering	Conceptual design of brake system and adaptive cruise control using SysML	Rational Software Architect [15]	Rational ClearCase [14]
Requirements engineering	Management of brake system and adaptive cruise control requirements	Rational RequisitePro [16]	Rational RequisitePro database
Mechanical engineering	CAD/CAE design and mechanical analysis of disk brake system	Dassault Systèmes CATIA [5]	SmarTeam as PDM with DB2 [20]
Software engineering	Design of embedded software for brake system and adaptive cruise control	Websphere Studio Application Developer [25]	Rational ClearCase [14] as SCM

Table 2 Conjectured classification of SDM Interfaces.

Class	**Members**	**Role**
SDM Core Services	Session Control and Credentials Services	Handles context of negotiation session between discipline-specific environments and SDM repository
	Systems Engineering Artifacts Authoring Services	Handles authoring of information residing in the SDM repository (*e.g.* artifact tuples)
	Systems Engineering Traceability Services	Handles algorithmic traversals of inter-disciplinary traceability links for purpose of system analysis
SDM Client Services	Inter-disciplinary Services	Handles negotiations between discipline-specific repository client and SDM Core Services
	Discipline-Specific Introspection Services	Handles algorithmic traversal of intra-discipline traceability links for purpose of finer system analysis

5 Conclusion

In this paper, the authors have presented an approach which supports inter-disciplinary collaboration, called "system data management" (SDM) and have contrasted it with the prior art. The principle underlying the SDM is to keep discipline-specific development artifacts where they naturally belong and manage the inter-disciplinary traceability links separately. Such links support associated prescribed inter-disciplinary collaboration processes. The integrity of the traceability is guarantied by a simple, but strictly enforced, request/update protocol collaboration pattern between the individual disciplines and the SDM repository.

From a deployment perspective, this process-centric approach is less invasive than those of prior art and is lighter, therefore more scalable. In particular, it respects existing infrastructures and may also be implemented using their underlying technologies. Moreover, the SDM architecture captures development process knowledge in a reusable way since artifacts are linked only from the perspective of (repeatable) processes. The applicability of the SDM architecture relies on the existence of prescribed inter-disciplinary collaboration processes. However, the wider adoption of such systems engineering practices is an industry trend.

The authors have implemented a prototype in their laboratory that provides an early validation of the feasibility and value of the SDM concept, and they are currently involved in an industrial validation study.

REFERENCES

1. Balmelli, L. and Moore, A. (2004), 'Requirement Modeling for System Engineering using SysML, The Systems Modeling Language', *In Proceedings of DETC, Computers & Information in Engineering Conference*, Salt Lake City.

2. Bosh, R. (1996), 'Automotive Handbook', Fourth Edition, BOSH, *Horst Bauer (Editor).*

3. Burkett, M., Mixer, K., Carrillo, L. and Asgekar, V. (2004), 'Don't Let Increased Software Features Derail Your New Product Development and Launch Goals', AMR Research Report.

4. Chu-Carroll, M.C., Wright, J. and Shields, D. (2002), 'Supporting aggregation in fine grained software configuration management', *ACM SIGSOFT Software Engineering Notes*, Volume 27, Issue 6.

5. CATIA, http://www.catia.com, Dassault Systemes.

6. Dahlqvist, A.P., Crnkovic, and Larsson, M. (2001), 'Managing Complex Systems - Challenges for PDM and SCM', Software Configuration Management, SCM 10, 23rd ICSE, Toronto, Canada.

7. Crnkovic, I., Asklund, U., and Dahlqvist, A.P. (2003), 'Implementing and Integrating Product Data Management and Software Configuration Management', *Artech House Publishers*.

8. Dahlqvist, A.P., Crnkovic, I., Hedin, A. and Larsson, M., Ranby, J., and Svensson, D. (2001), 'Product Data Management and Software Configuration Management – Similarities and Differences', *The Association of Swedish Engineering Industries*.

9. Dahlqvist, A.P., Asklund, U., Crnkovic, I., Hedin, A., Larsson, M., Ranby, J., and Svensson, D. (2001), 'Product Data Management and Software Configuration Management – Similarities and Differences', *The Association of Swedish Engineering Industries*.

10. Estublier, J., Favre, J.M., and Morat P. (1998), `Toward PDM / SCM: integration ?', In Proceedings of the 8th International Workshop on Software Configuration Management, Lecture Notes in Computer Science 1439, pp. 75--95, Bruxelles, Belgium, Springer Verlag.

11. Gruhl, W. (1992), "Lessons Learned, Cost/Schedule Assessment Guide", NASA Comptroller's Office.

12. International Council on Systems Engineering (INCOSE), *http://www.incose.org*.

13. National Highway Traffic Safety Administration, (NHTSA), *http://www.nhtsa.dot.gov*.

14. Rational ClearCase, 1990-2005, *http://www-306.ibm.com/software/awdtools/clearcase*, International Business Machines Corporation.

15. Rational Software Architect (RSA), 2005, http://www-306.ibm.com/software/awdtools/architect/swarchitect, International Business Machines Corporation.

16. Rational RequisitePro, 2000-2005, *http://www-306.ibm.com/software/awdtools/reqpro*, International Business Machines Corporation.

17. Rational Unified Process for Systems Engineering, *http://www-128.ibm.com/developerworks/rational/library/2766.html*, International Business Machines Corporation.

18. Render, H. and Campbell, R. (1991) `An object-oriented model of software configuration management', *Proceedings of the 3rd international workshop on Software configuration management*, pp.127—139, ACM Press, New York.

19. Service-oriented architecture (SOA), *http://www.service-architecture.com*.

20. SmarTeam, 2000-2005, *http://www.smarteam.com*, SmarTeam Corporation.

21. Systems Engineering Center Of Excellence (SECOE) (2002), "Impact of SE at NASA", International Council on Systems Engineering (INCOSE).

22. Systems Engineering Center of Excellence (SECOE) (2003), "Value of SE", International Council on Systems Engineering (INCOSE).

23. Systems Modeling Language (SysML) Partners, http://www.sysml.org.

24. The Unified Modeling Language, *http://www.uml.org*, the Object Modeling Group (OMG).

25. WebSphere Studio Application Developer (WSAD), http://www-306.ibm.com/software/awdtools/studioappdev/support, 2004, International Business Machines Corporation.

26. Westfechtel, B. and Conradi, R., "Software Configuration Management and Engineering Data Management", In Proceedings of the 8th International Workshop on Software Configuration Management, Lecture Notes in Computer Science 1439, pp. 96--106, Bruxelles, Belgium, Springer Verlag.

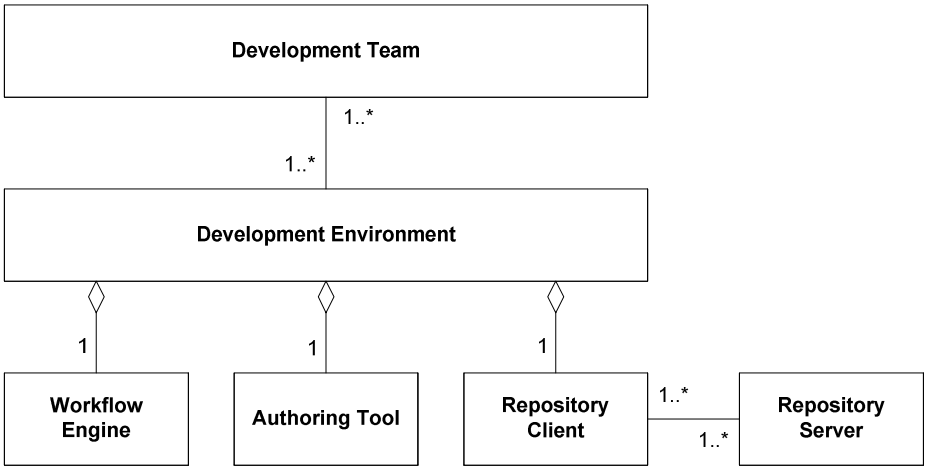

Figure 1 Intra-disciplinary infrastructure pattern represented according to the Unified Modeling Language (UML) convention [24] (*cf.* Section 3.1).

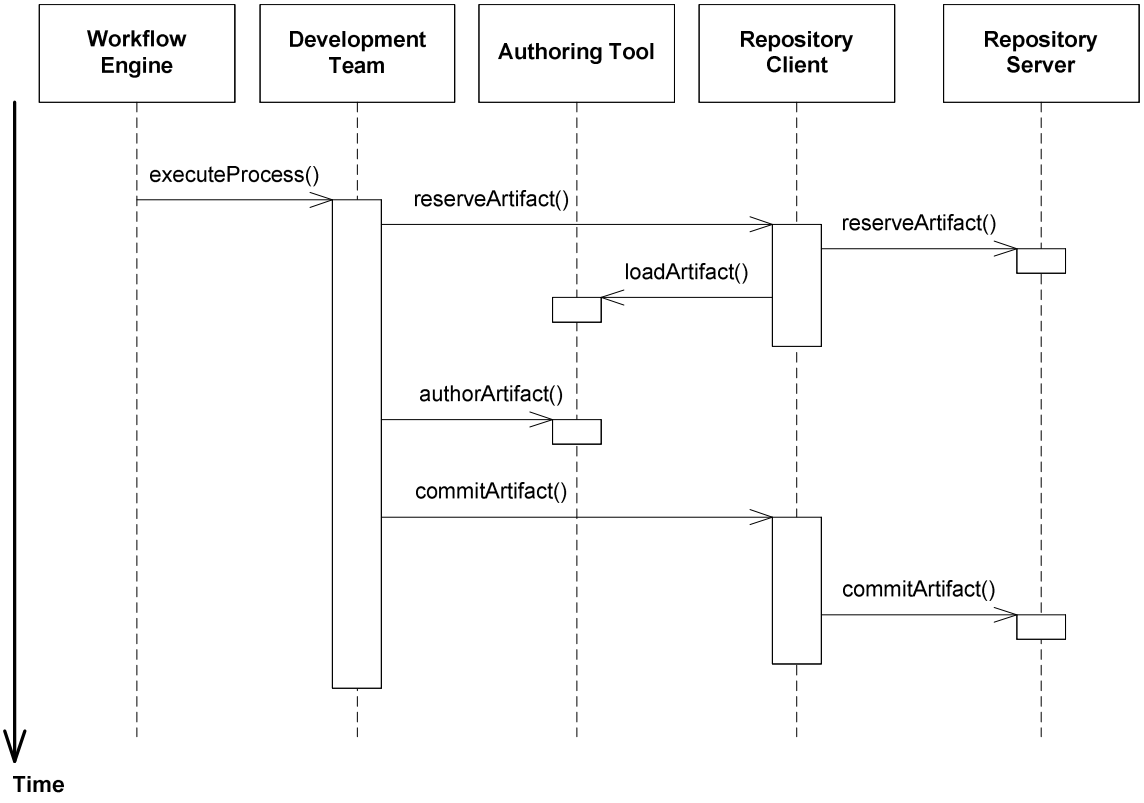

Figure 2 Intra-disciplinary collaboration pattern (*cf.* Section 3.1).

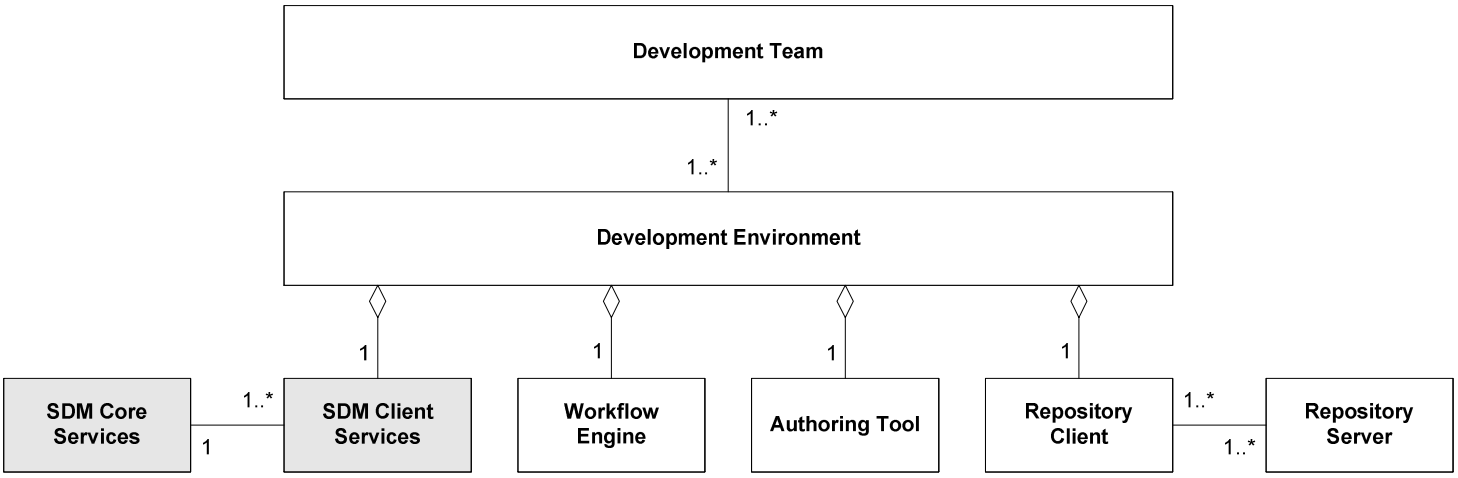

Figure 3 Inter-disciplinary infrastructure pattern (*cf.* Section 3.3).

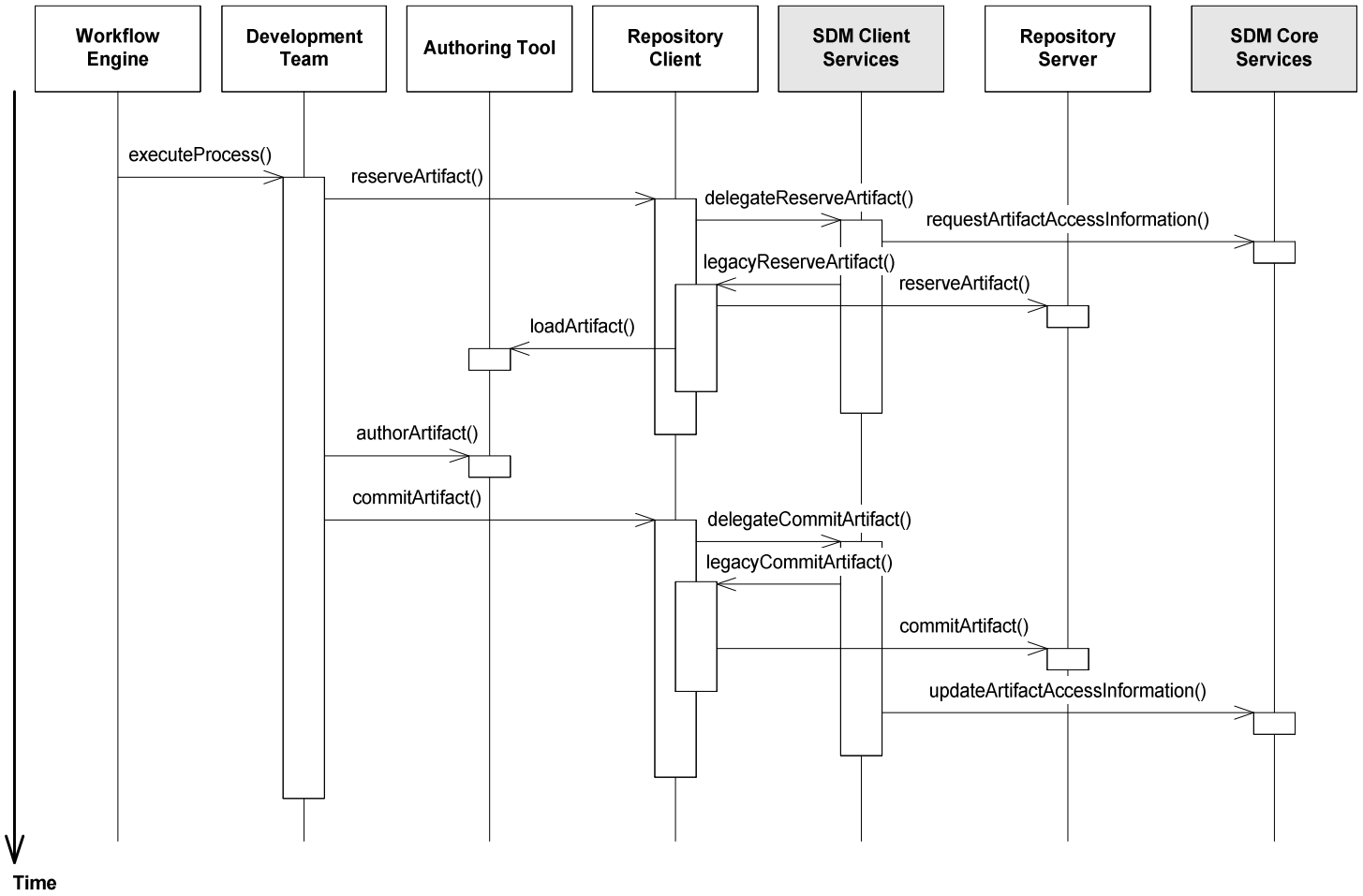

Figure 4 Inter-disciplinary collaboration pattern (*cf.* Section 3.3).

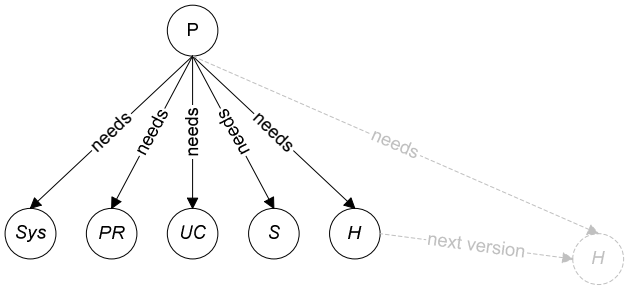

Figure 5 Set of multi-disciplinary resources needed to execute a given process (*cf.* Section 3.3).

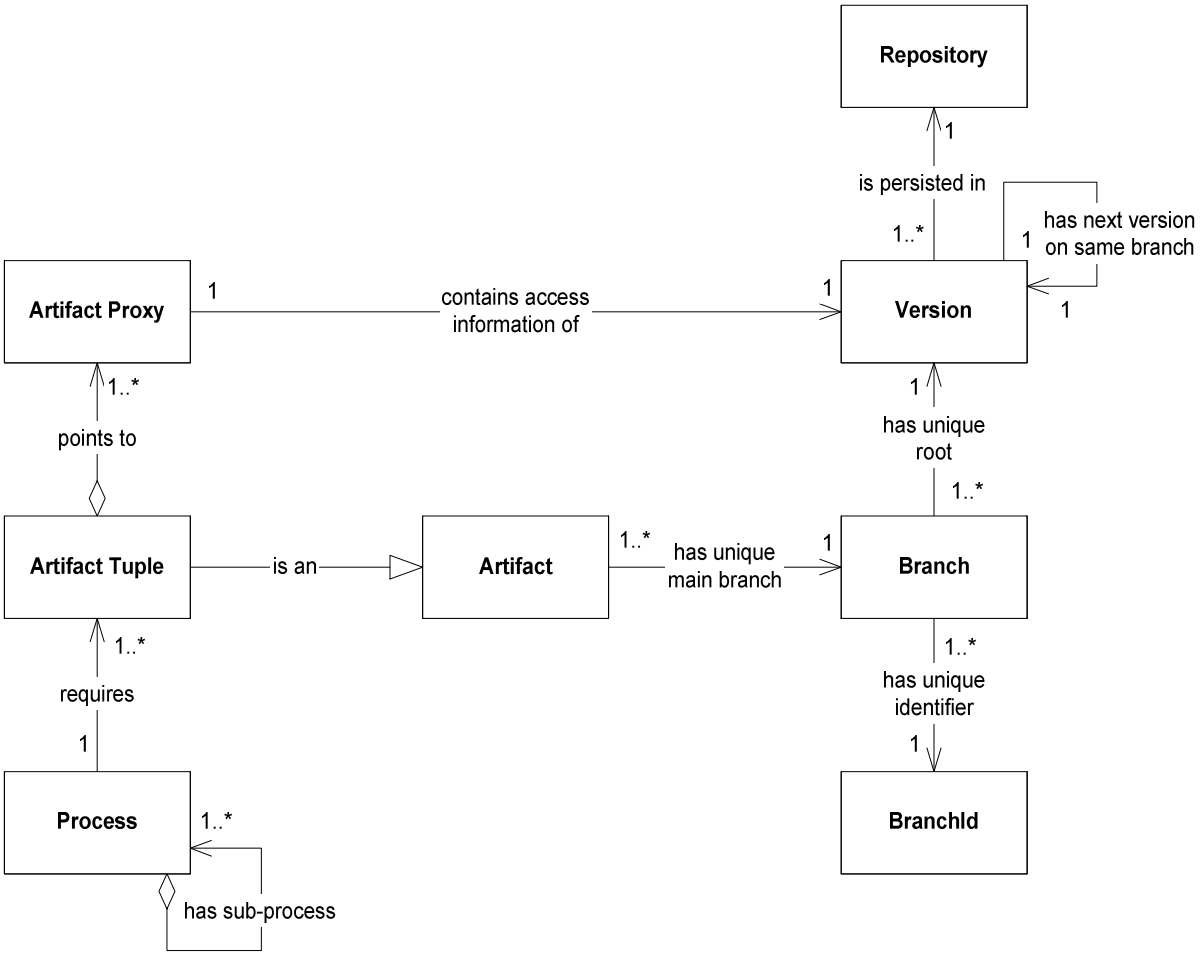

Figure 6 SDM data model (*cf.* Section 3.3).