Meta-Data Version And Configuration Management In Multi-Vendor Environments

John R Friedrich, II
Meta Integration Technologies, Inc.
949 Sherwood Avenue, Suite 200
Los Altos, California 94022
650-917-7700
friedrich@metaintegration.net

ABSTRACT

Nearly all components that comprise modern information technology, such as Computer Aided Software Engineering (CASE) tools, Enterprise Application Integration (EAI) environments, Extract/Transform/Load (ETL) engines, Warehouses, EII, and Business Intelligence (BI), contain a great deal of meta-data, which often drive much of the tool's functionality. These meta-data are distributed and duplicated, are often times actively interacting with the tools as they process data, and are generally represented in a variety of methodologies. Meta-data exchange and reuse is now becoming commonplace. This article is based upon the real challenges found in these complicated meta-data environments, and identifies the often overlooked distinctions and importance of meta-data version and configuration management (CM), including the extensive use of automated meta-data comparison, mapping comparison, mapping generation and mapping update functions, which comprise a complete meta-data CM environment. Also addressed is the reality that most repositories are not up to the task of true version and configuration management, and thus true impact and lineage analysis, as their emphasis has been on the development a single enterprise architecture and the concept of "a single version of the truth."

1. 1. INTRODUCTION

Meta-data management has become a sophisticated endeavor. Nearly all components that comprise modern information technology, such as Computer Aided Software Engineering (CASE) tools, Enterprise Application Integration (EAI) environments, Extract/Transform/Load (ETL) engines, Warehouses, EII, and Business Intelligence (BI), all contain a great deal of meta-data. Such meta-data often drives much of the tool's functionality. Additionally, this meta-data are distributed and duplicated, are often times actively interacting with the tools as they process data, and are generally represented in a variety of methodologies.

Harvesting this meta-data is a significant challenge in itself, and many methods and/or architectures for meta-data capture, management, analysis and use have been presented in the literature [10], [11]. Meta-data exchange and reuse is now becoming commonplace, thanks to initiatives like the Object Management Group's Common Warehouse Metamodel[1], especially the XML based XMI[2], and extensive work by many of the current vendors of CASE, ETL and BI tools, as well as meta-data glue providers, such as Meta Integration Technologies, Inc. Much can be learned in the process of deploying such meta-data management environments. In particular, meta-data version and configuration management becomes essential to the success of such implementations.

2. CONFIGURATION MANAGEMENT

Configuration management is a standard part of software (SW) engineering [3]. But what is meant by configuration management (CM)? As an assembly of references to definitions and subtleties of configuration management for software, the author has found no better collection than Brad Appleton's ACME Project page [1], [8]. While the emphasis is on software development, not meta-data, there are a great deal of directly applicable concepts and valuable analogies to be gleaned. Key points include:

- The need to track multiple versions (states in time) and statuses (place in a process) of configuration items (discrete artifacts)
- Managing changes through the software engineering lifecycle
- Impact analysis due to actual or proposed changes (the SW lifecycle is generally better understood than the data or meta-data lifecycle)

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA
Copyright 2005 ACM 1-59593-060-4/05/06 $5.00"

1. See the "Data Warehousing, CWM™ and MOF™ Resource Page" at http://www.omg.org/cwm/
2. See the "CORBA®, XML and XMI® Resource Page" at http://www.omg.org/xml/

- Understanding and analyzing traceability through the life-cycle and interfaces/exchanges
- Managing concurrent engineering activities
- Ensure proper distribution, assembly and deployment of engineering assets

As software engineers, team leads, project managers, and software, data and enterprise architects can attest, this is a difficult, but necessary, set of tasks. Effective software CM requires a CM repository, a repeatable software CM process, the appropriate resources and what is referred to so often as "buy-in", or full conformance with the processes and full integration with the environment, by engineers, managers, designers and integrators.

3. CM AND META-DATA

Software CM is so important precisely because it is software that drives the functioning of the IT infrastructure. Today, nearly every CASE, ETL, BI and EAI tool has its own repository and designer for meta-data. Now that this meta-data is used to drive much of the modern IT infrastructure, just as with software in the past, similar *meta-data CM* activities must be adopted in order to have effective information management, and specifically meta-data management. In addition, of course, one needs to ensure that the meta-data specific concerns are addressed and that software CM processes are appropriately adapted to the meta-data world.

4. META-DATA SCOPE

For the purposes of this paper, the meta-data considered here shall be limited to that which is directly used by IT infrastructure components or descriptive meta-data directly related to the same. E.g., included are data models, XML schemas/DTDs, Object-Class Diagrams, object interface specifications, ETL schemas and mappings, dimensions, joins, CASE models, ETL and BI process/lifecycle meta-data, and finally meta-data related directly to the organizational standards for these same elements. Excluded in the scope of the article are generally any meta-data elements that are not forward or reverse engineers from the CASE, ETL, BI, tools, except as necessary for mapping, version and configuration management. Some refer to this as "technical meta-data" [5].

Let us begin with a data flow example, extended from a generalization of the meta-data environments referred to above. Generally, one can view various meta-data that drive specific IT infrastructure components in reference to the data movement or flow that they represent, as in Figure 1. This figure depicts many example data flows for a typical information management environment. Each portion of the data flow has its associated metadata, as detailed in Table 1.

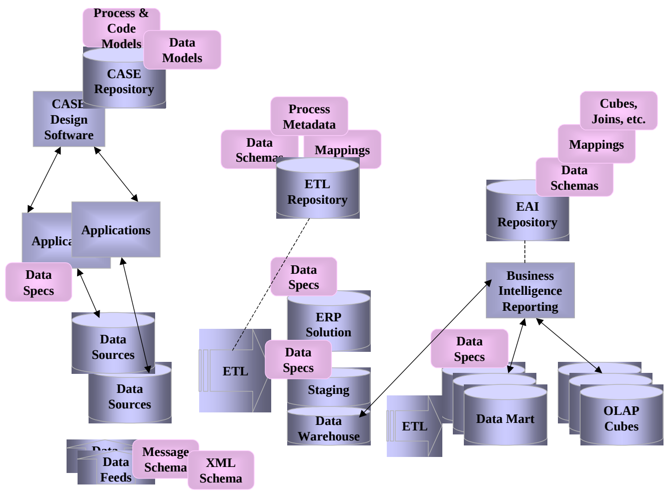


Figure 1. Meta-Data Drives Modern IT Infrastructure.

Table 1. Data Flows and Related Meta-Data.

Data	**Meta-Data**
Operational Data Sources (ODS)	
Data stores	Database specifications, DDL and CASE models (e.g., RDBMS, logical E-R, OODBMS, etc.)
Application software and ERP	Interface specifications, interface definition languages, and CASE models
Other Data sources	Message structure/models and specifications
Extract, Transform and Load (ETL)	
Data Movement and Transform Engine	Source and target data schema, transformation rules, ETL process specifications, and ETL Repository
Data Warehouse (DW)	
Staging and warehouse databases	RDBMS schema, models, CASE models, DDL, and staging to warehouse transforms (e.g., SQL)
Business Intelligence and Reporting (BI)	
Data marts, cubes, views, portals, etc.	RDBMS schema, models, CASE models, BI Repository, view and dimension specifications, aliases, BI transforms, portal interface specifications and models, WSDL, etc.
Standardization	
Generally none	Logical and conceptual schema, database models, CASE models, standards process management meta-data (e.g., steward), data dictionary, mappings to rest of meta-data, etc.

Now, for clarity, let us look at one aspect of the data flows and related meta-data that drives the ETL and BI infrastructure. A simplified diagram is provided in Figure 2. The data flow is a straightforward extract from source RDBMSs using an ETL tool that loads into a warehouse (with an intermediary staging DB). A BI tool then reads and/or extracts data from the warehouse into marts and cubes, and provides reports.

In order to represent this data flow in a meta-data management environment, meta-data in the form of data schemas/models is "stitched" together across tools. For example, models of the source RDBMS data structures or design CASE models are extracted, as are the source schemas from the ETL tool. Neither "knows about" each other until they are stitched together in the meta-data management environment. Additionally, extracts for the ETL tool will provide meta-data describing transformation mappings and ETL process information to the target schemas (the staging DB). These target schemas are then stitched to the staging data models and those to the warehouse data models. Finally, the warehouse models are stitched to the reference data schemas from the BI tool and additional mappings are provided by that tool to the dimensions, joins and views.

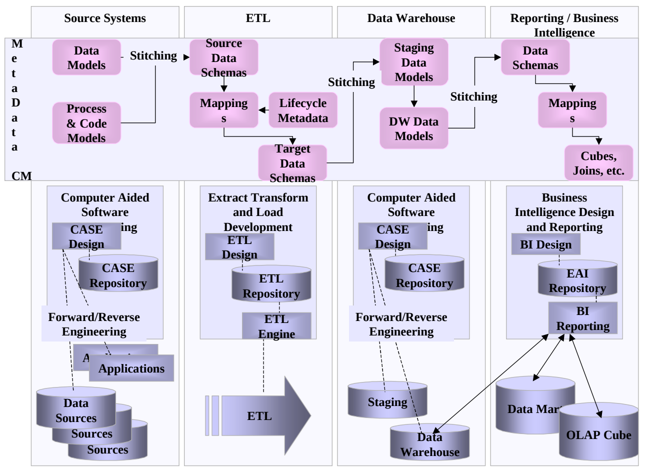


Figure 2. ET, DW and BI Data Flow Related Meta-Data.

After going through this process, it is possible to answer some very powerful questions. Two obvious questions include:

- Given a change to the meaning of an element in the source system data schema, what is the impact on reporting using the BI tool (and every step in between)?
- Given that a report field is thought to be incorrect, what is the data flow *lineage* of systems and transformations that lead to that field's value? Or the Sarbanes-Oxley question: How do I certify that this financial report is indeed accurate, i.e., where did this data come from?

Additionally, some more sophisticated, call them meta-data business intelligence, questions include:

- What transformations are reproduced, perhaps, redundantly in the ETL and BI solution? How? Are they consistent? Should they be?
- What DW data is unused by BI queries? What source system data?

Remember these questions as they will be used to determine much of the specialized requirements for version and configuration management of meta-data. Now, let us follow through the analogies with software engineering CM. These topics will include:

- Multiple Versions and Multiple Configurations
- Managing changes through the meta-data management lifecycle
- Impact analysis
- Lineage Analysis
- Understanding and analyzing traceability through the life-cycle
- Managing concurrent engineering activities
- Ensure proper distribution, assembly and deployment of engineering assets.

5. MULTIPLE VERSIONS AND MULTIPLE CONFIGURATIONS

As a software CM concept, a configuration is a set of versions of source and object modules, most likely associated with a build of the system. The meta-data example above considered *one configuration* only. In reality, there are several dimensions of possible versions to consider, and thus a multitude of configurations. These dimensions include:

- Multiple deployed versions of each of the source systems, ETL schemas and transformations, warehouse and staging (distributed warehouse), and BI reports and design
- Multiple design, developmental, beta, etc., versions of all of the above
- Multiple version of standards and/or reference models, standard code sets, code set mappings, look-up and reference tables, etc.
- Multiple versions of data migration transformations for new versions of data systems (i.e., data migration for the accounts receivable source system from version 4.1 to 4.2).
- Other more subtle ones which are not within the scope of this paper.

5.1 MANAGING CHANGES THROUGH THE META-DATA MANAGEMENT LIFECYCLE

Much of the well-defined methods used for software CM can be applied directly, such as ensuring that configuration items are versioned, version history is maintained, configurations of versions are identified and maintained, proper milestones for versioning of meta-data are defined, etc. An example of such a meta-data management process is provided in Figure 3.

5.2 IMPACT ANALYSIS

Impact analysis in the meta-data world is analogous, but not identical, to the software engineering parallel. In the case of meta-data, impacts generally fall into one of the following categories:

- A change in data or interface specification, type, or meaning
- A change of schema or relationships among data elements and interfaces
- A change of transformation, validation or generation rules.

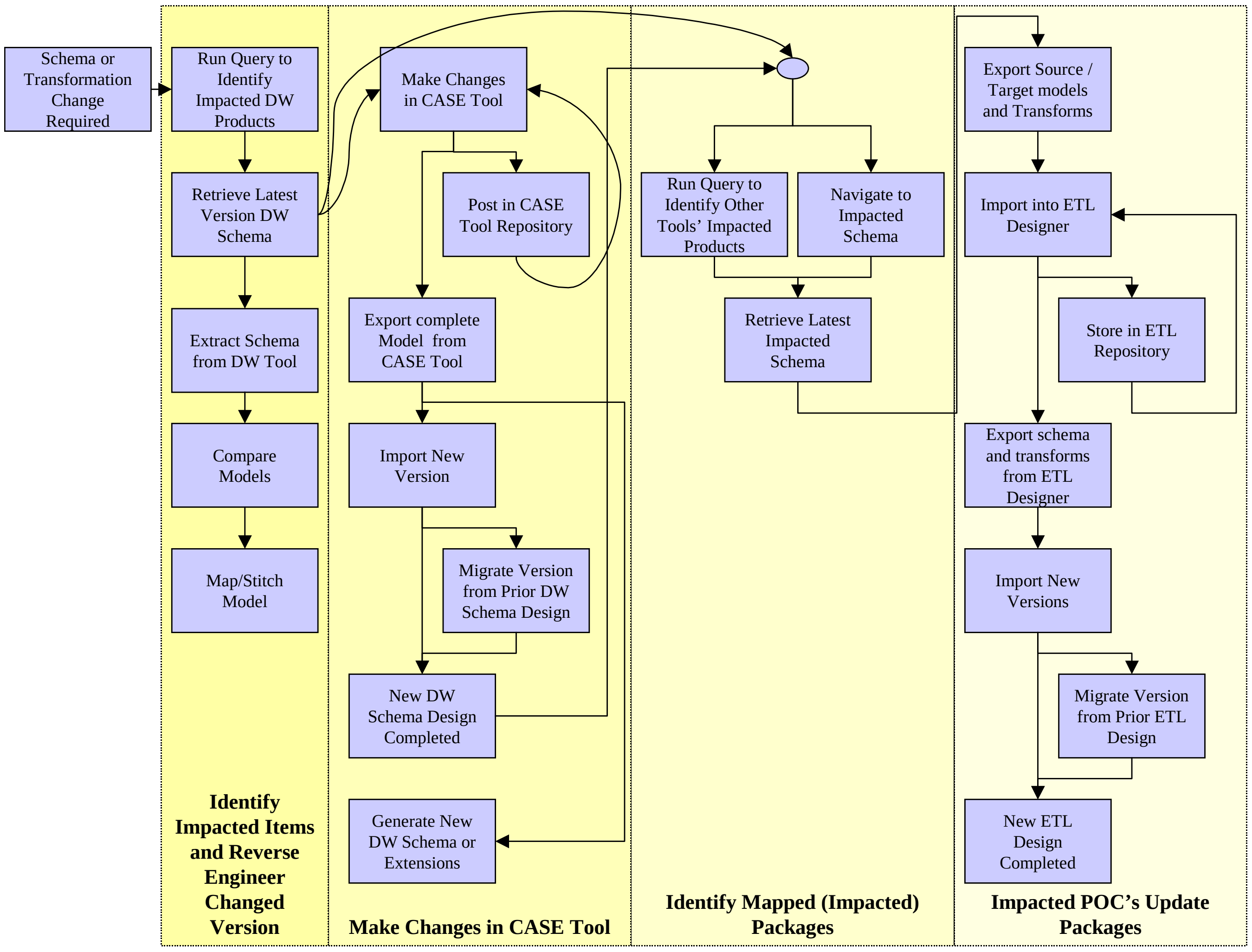
Impacts can then cascade for any of the following reasons:

- Data-flow impacts down-stream
- Additional data inputs required for the change to be successful (lineage impacts)

- Standards impacted by change.

Each of these changes creates another version of the mapping in the meta-data management environment.

Note, however, that the software CM approaches are *not* sufficient (do not address) the issue of mappings (e.g., the stitching between the CASE tool models and the source ETL schema). In order to answer any of the questions identified in that last section the meta-data CM environment must be able to identify the version of each of the *mappings* among the meta-data, not just simply what versions exist of the schemas and transformations themselves.



Schema or Transformation Change Required
Run Query to Identify Impacted DW Products
Retrieve Latest Version DW Schema
Extract Schema from DW Tool
Compare Models
Map/Stitch Model
Identify Impacted Items and Reverse Engineer Changed Version
Make Changes in CASE Tool
Post in CASE Tool Repository
Export complete Model from CASE Tool
Import New Version
Migrate Version from Prior DW Schema Design
New DW Schema Design Completed
Generate New DW Schema or Extensions
Make Changes in CASE Tool
Run Query to Identify Other Tools' Impacted Products
Navigate to Impacted Schema
Retrieve Latest Impacted Schema
Identify Mapped (Impacted) Packages
Export Source / Target models and Transforms
Import into ETL Designer
Store in ETL Repository
Export schema and transforms from ETL Designer
Import New Versions
Migrate Version from Prior ETL Design
New ETL Design Completed
Impacted POC's Update Packages

Figure 3. Sample Meta-Data ETL Version Control Process for Meta-Data CM.

To better understand this meta-data specific requirement, consider the configurable items, versions of those and specific file formats required for managing the meta-data for an RDBMS to Informatica ETL [9] to an ERwin [4] designed warehouse DB data flow. In this example there are multiple versions of a source system RDBMS and of the Warehouse (as changes have created new versions). These configurations then require multiple versions of both of the source and target schemas in Informatica, and many of the permutations of sources and targets represented as versions of the ETL mappings and transforms in Informatica. In turn, the existence of multiple source and target versions then requires an equivalent number of versions of the mappings from the RDBMS schemas in the ERwin to the Informatica source schemas, and the same for the target side.

Successful meta-data impact analysis also requires a *meta-data comparison* facility, much like those in existing CASE tools, and a *mapping or transformation comparison capability*, unique to the meta-data management environment. Analogous with the software engineering "diff" type tools, this meta-data comparison tool would provide a report of the differences between two models, for example. Unique to the meta-data view, a mapping comparison facility would provide a report on the differences between two mappings, and thus the impacts to the downstream meta-data and mappings.

Finally, just as in ETL or BI tools, it is important to automate, to as great a degree as possible, the generation of the mappings (stitching, standards mapping, etc.) that are not directly reverse engineered from a tool. Here, the meta-data comparison tool's output could be used by a *meta-data mapping tool* to generate mappings, again based upon some make-file or wizard type rules and specifications. In most cases, the stitching would be nearly automatic as the data names, structures, definitions and types are generally the same across tools that are referring to the same data (e.g., the target schema in the ETL tool and the data warehouse RDBMS model).

5.3 LINEAGE ANALYSIS

The Sarbanes-Oxley question in the earlier section represents a type of analysis requirement, often times referred to as *lineage analysis*. Lineage analysis requires that one can navigate "backward" in terms of the data flow in order to determine all the data sources, transforms, interfaces and relationships that were used to create a data element or report field toward the end of the data stream. These can be divided into the following types of analysis:

- Based upon the current operational configuration
- Based upon a configuration of historically operational systems
- Based upon proposed or developmental configuration.

As one may conclude from this list, it is often necessary to perform lineage analysis on *historical* and *proposed configurations* of versions of the meta-data and mappings (including stitched mappings). In this way, lineage analysis, in a similar fashion to impact analysis, requires a sophisticated version and configuration management environment.

5.4 UNDERSTANDING AND ANALYZING TRACEABILITY THROUGH THE LIFE-CYCLE

Here the word lifecycle refers to the meta-data management lifecycle, analogous to the software engineering lifecycle. In this way, not only is it important to be able to identify operational configurations of versions of the meta-data, it is also necessary to track the *version history*, i.e., what versions lead to other versions. A good version nomenclature (numbering) system is critical. Valuable also is the ability to notate version history relationships among the versions.

Not so analogous to the software engineering CM principles is the need to provide traceability and *automated generation* of versions of mappings, whether stitching, transforms or standardization relationships. A very common reason for the need for a new version is the change to a data type or data schema. This change will "cascade" down the data stream, impacting many mappings among meta-data elements. Each of these changes creates another version of the mapping in the meta-data management environment. In order to be manageable, these mappings must be created in as automated a fashion as possible. A wizard like, or make-file like, mapping regenerator or *migrator* could provide such functionality, creating a new version of each of the mappings based upon the information contained within the previous mapping, a set meta-data comparison rules, and the make-file or wizard based specifications.

5.5 MANAGING CONCURRENT ENGINEERING ACTIVITIES

The analogy with software engineering CM is very strong for currency control of meta-data design and engineering. The meta-data CM environment will interact with the design tools:

- CASE for RDBMS's, messages and software interfaces
- The ETL designer and/or repository
- The DW designer or CASE tools
- The BI designer and/or repository.

In the software engineering analogy, the source code is completely maintained within the CM tool. This is not always the case for meta-data, as the meta-data that may be extracted from a particular tool (ETL, BI, DW builder, etc.) could easily be less than what is necessary to re-create the functionality of the tool. Thus, concurrency issues must also be managed, and in fact should be managed, in the ETL/BI/DW/CASE design tool wherever possible. Multiple versions in the ETL repository may exist before a new version is placed in the enterprise meta-data management environment. Again, this consideration also impacts the meta-data CM process, as the example back in figure 3 demonstrates, ensuring that the appropriate milestones are defined for new versions to be placed in the enterprise meta-data management environment.

5.6 ENSURE PROPER DISTRIBUTION, ASSEMBLY AND DEPLOYMENT/REUSE OF ENGINEERING ASSETS

In the software engineering analogy, make files and other specifications allow for the direct and automated generation of an *operational* configuration of the software. I.e., the source code is compiled into object code and is then assembled with other libraries and components to create a functioning system. Unfortunately, while CWM and the efforts of ETL/BI/DW/CASE/Repository vendors have made meta-data movement and reuse more effective, this is not equivalent to having source code generate object code. Instead, these tools refer to *forward-engineering*.

It is important to note this very strong distinction between the version and configuration management of software and that of meta-data. Meta-data, is very likely to be forward-engineered into, that reused by, many different data management tools. However, this type of reuse is often the result of meta-data translation or migration, oftentimes across different methodologies (e.g., UML and Relational). A common example would be:

1. Develop the data warehouse schema in ERwin
2. Forward engineer that into an RDBMS
3. Migrate the model to an ETL design tool, say Informatica Designer, as a target schema
4. Forward engineer that same model into the ETL engine
5. Migrate the updated ETL target model to the BI design tool, say BO Designer [2]
6. Forward engineer into a reporting tool, such as Crystal Reports [2].

In other words, meta-data reuse is a critical way in which meta-data is leveraged. It is also an example of why meta-data management is so critical to the enterprise, as its quality may impact many different systems.

This view of meta-data as a reusable asset also means that impact analysis is not simply an answer. I.e., it is not simply a report that is used as a reference to make changes from. Instead it is an integral part of the CM process to actually migrate the changes to the appropriate tool. Again, using the data warehouse example, changes to the design of the operational data store in a CASE tool are moved to the ETL source schema and forward engineered. These changes are also migrated to the standard reference model(s) and this same information can be migrated to the target side of the ETL tool as well as the BI designer., i.e., wherever the data flow takes one. In order to manage this meta-data properly, these steps all must be captured as versions of meta-data for the given tools, including the related development, testing, deployment and operational configurations of those versions. In practice, this level of CM makes extensive use of the automated meta-data comparison, mapping comparison, mapping generation and mapping update functions, which comprise a complete meta-data CM environment.

6. NOTE ON REPOSITORIES

Even still, the above example is not sufficient in order to truly answer our MD CM questions. In order to address all of the above requirements, one must also be able to know what the deployed configuration *in fact was* at some point in time, or what it *will be* or *might be* (given a hypothetical or proposed version). A simple cut might be, "what are the sources for a particular report summary on a given date, for a given date range, or will be after deployment of new versions?"

To get at these kinds of answers, version and configuration management must be a fundamental part of the meta-data management environment, oftentimes considered the repository. Unfortunately, most repositories are designed around the concept of capturing and managing meta-data for what the IT infrastructure (sometimes even expanded to the Enterprise Architecture) *should* look like right now, also oftentimes referred to as "the single version of the truth."

Such an approach is appropriate for some very important requirements, specifically those growing out of the concept of the common information model [6], Enterprise Architecture [7], and logical and conceptual data/information/process management. These are valuable exercises, and it is certainly valid to say that the first step in MD CM is to reduce the number of configurations as much as is possible, i.e., standardize and rationalize. However, when working with the actual systems, CASE tools, ETL, warehouse, marts, BI tools, etc., one version of the truth at a conceptual level "relates to" but does not detail what is necessary for accurate answers to our questions about lineage and impact analysis across versions and configurations of versions.

Additionally, the mechanisms employed by these repositories in order to "integrate" meta-data from different tools (CASE, ETL and BI, for example) are fundamentally different from version and configuration management concepts. Such repository environments tend to *merge* the meta-data items from different tools together, generally at the time of extraction from a tool, when they are thought to represent the same data source (e.g., the CASE tool design of the RDBMS with the ETL source schema of the same), as opposed to importing a version of what that tool has as meta-data and then managing the versions, configurations of versions, and the stitching across tools (see figure 4). As a result, reconstructing the version history, let alone performing change impact analysis when only one aspect of what has been merged together has changed (new version of the ETL, for example), is a very awkward or nearly impossible task.

In some way, this distinction in purpose is why the word "repository" or "meta-data repository" has not been used extensively in this article. Instead, a more utilitarian term "meta-data management environment" has been chosen, in order to concentrate on the CM aspects of meta-data management. Terminology, especially when weighted with marketing, is always a difficult sea in which to navigate. A "repository" can (and this article would support the argument that it *should*) be constructed with these MD CM capabilities as a fundamental design concept. A repository can and should be more than what might be referred to as a "meta-data warehouse."

7. CONCLUSION

Meta-data version and configuration management is quickly becoming a critical concern as more meta-data is extracted and reused by CASE, BI, ETL and warehouse environments. Many of the basic concepts of software CM may be successfully applied to meta-data CM. Due to the relationships intrinsic to meta-data elements, e.g. the stitching of meta-data across tools and meta-data reuse, there are also a number of meta-data specific CM concerns that must be addressed. With the growth in meta-data exchange, the options should only improve in terms of environments that are capable of the sophisticated type of CM necessary.

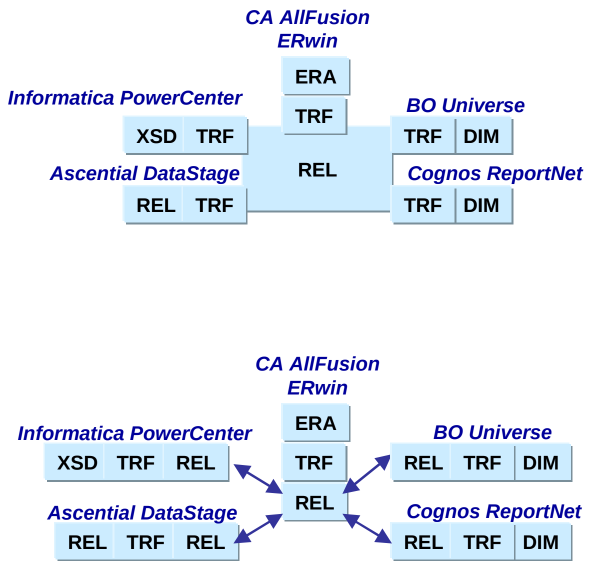


Figure 4. Meta-Data Merging versus Meta-Data Stitching

8. REFERENCES

[1] Appleton, Brad. The ACME Project: Assembling Configuration Management Environments (for Software Development). website at http://www. cmcrossroads.com/bradapp/acme/scm-defs.html

[2] Business Objects, http://www.businessobjects.com/

[3] Carnegie Mellon Software Engineering Institute, home page at http://www.sei.cmu.edu/.

[4] Computer Associates, http://www3.ca.com/.

[5] Do, Hong Hai, Rahm, Erhard. On Metadata Interoperability in Data Warehouses, *Report Nr. 01(2000)*, UNIVERSITÄT LEIPZIG, 2000.

[6] Distributed Management Task Force, Inc. (DMTF), Common Information Model (CIM) Standards at http://www.dmtf.org/standards/cim/.

[7] Federal Enterprise Architecture at http://www.feapmo.gov/fea.asp,
the DoD Architecture Framework at http://www.dod.mil/comptroller/bmmp/pages/arch_arch_home.html, and the Institute for Enterprise Architecture Developments at http://www.enterprise-architecture.info/.

[8] Hass, Anne. *Configuration Management Principles and Practice*, Addison Wesley Professional, 2003.

[9] Informatica, http://www.informatica.com/.

[10] Inmon, W. H., Claudia Imhoff, Ryan Sousa. *The Corporate Information Factory, 2nd Edition.* Wiley. 2000.

[11] Marco, David. Meta Data & Knowledge Management: Managed Meta Data Environment: A Complete Walk-Through, Parts 1-8, *DM Review*, 2004