Different Levels of Centralized Engineering Database Strategies for Integrated Development

Carolin Eckl
Berner & Mattner Systemtechnik GmbH

Erwin-von-Kreibig-Str. 3
D-80807 München
Carolin.Eckl@berner-mattner.com

Markus Brandstätter
Lehrstuhl für Raumfahrttechnik
Technische Universität München
Boltzmannstraße 15
D-85748 Garching
m.brandstaetter@tum.de

Abstract. The development of large and complex products is currently carried out in small, separate teams of engineers trained in diverse disciplines. Mechanical and electrical engineers are involved in the development of high-tech and embedded systems as well as software designers. This diversity of disciplines interferes with the efficiency of communication, which is usually carried out in (informal) discussions and by passing documents and models.
Within this paper, the passing of documents is explained and enhanced in several levels. Solutions, which are currently available (such as tool chains) are discussed as well as approaches that rest upon a central, document- and object-oriented database. The most elaborate database strategy presented uses similarity relations between documents to infer semantic meaning and facilitate the access to meaningful and relevant development data. This paper concludes with the description of development processes that are introduced with the use of this database solution.

Introduction

Some difficulties experienced in product development can also be detected in jigsaw puzzles – a simple version with few pieces can easily be overviewed and is assembled quickly, whereas a more complex version consisting of a multitude of pieces can hardly be overviewed in advance. This often leads to the formation of separate connected groups of pieces (like islands), which have to grow together to yield the picture.
Similarly, systems consisting of few parts can be handled by one developer trained in the craft. Due to its complexity, a high-tech product such as a satellite or car is no longer understood in depth by a single person, but requires a large group of developers. It shows that the more people are involved, the more communication and organization contribute to an increase in complexity.
On top of the already high complexity, the technical development of such high-tech systems involves a multitude of disciplines – at least industrial design for the appearance, mechanical engineering for the hardware layout, electrical engineering for wiring and electrical system behaviour and with increased regularity also computer science for more flexible behaviour. Parts developed by each of the separate development areas have to fit and work together in the final product. The collaboration of parts is often tested in a "big bang" integration into a prototype at the end of the design phases, which frequently leads to the need for an elimination of errors through rework and thus delays.
At the same time, the utilization of models during the design phases is continually increasing (Petrasch and Meimberg 2006, 11) to master complexity through a visualization of the situation

and provide more understandable means for communication between developers. Like the involved subject areas, the models used are very different from one another. This variety of models and modeling paradigms derives from the differences in the angle, from which the system is viewed (which is illustrated in Figure 1).

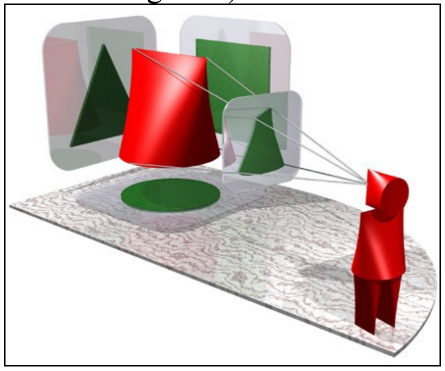

Figure 1: Different viewpoints and angles (Abadi and Cardelli 1995)

As mentioned before, the knowledge developed within the separate groups and disciplines has to be brought together in order to obtain a complete product or system. The essence of this knowledge is already present within the different models created during the design phases. Therefore, the logical next step is easing communication between developers through the management of models and ultimately an interlinking of the various separate models to anticipate the integration on a more abstract level and hence facilitate the construction of a prototype. As a consequence, providing for an earlier integration is directly related to the development costs due to the fact that correcting errors in models (i.e. in earlier development stages) is cheaper than a rework on the product itself in later phases (NASA 1995).

The support for the interlinking has to occur on the level of models, which are often created within software tools such as CATIA or Matlab. In the remaining paper, tool will refer to this type of software program that supports the development of a product or piece of software.

In general, there are two ways of interlinking the models edited and created with these tools:

- either by directly connecting two or more tools at runtime or
- by transforming the data contained in the saved documents of one tool to a format that can be read by a second tool

The following section will explain the differences between the two methods more clearly and shows already existing approaches, which represent the first steps towards linking models. After this, more elaborate steps towards truly interlinked models are pointed out, which are based upon a central data storage that increases the ease of model-based communication. The main processes required by such an elaborate approach during development are discussed thereafter and finally, this paper concludes with the vision of an even closer connected central model.

The tool-centered development method

Information (including models) is currently exchanged between the members of a development team. The most frequent way to accomplish an information exchange is the passing of documents to a colleague or customer. This may happen in an organized way, which is described within the development process such as reviews, through a software-based document management system or by informal requests.

Frequently, the contents of received documents are entered into another tool by hand. Information and semantics may be lost during this manual translation, due to the heterogeneous ways of reusing the data (Yang et al. 1997, 255). A structured and reproducible way of transforming data may be accomplished through the direct connection of development tools, which is widely supported by software vendors and software development companies that write plug-ins or filters for these tools. The resulting “tool chain” is discussed later, whereas the original document flow is illustrated in the following subsection.

The document flow

The most natural case of a document flow is passing on a document, which is read and modified using the same type of tool it has been created with. Any RTF (Rich Text File) document is a good example for such a case – it can be read and modified by anyone using a text editing tool that understands the rich text format. Text documents are also used frequently to pass information across the boundaries of groups, departments and enterprises, because text editing tools are present on almost any computer and the format promises good legibility.

We consider documents to be more than human-readable, formatted text and so the following definition of document will be used throughout the paper:

A document is considered a container for the representation of (coherent) pieces of information. It is accessible and editable and stores its contents for a certain stretch of time and the purpose of communication.

The exchange of a document takes place whenever it shall be modified or serves as the basis for a task to be carried out within a process. Usually, the task to be carried out produces its own documents and allows for passing these on. The task-based nature of the document flow and its simplicity contribute to its wide spread; documents can be utilized without modification of the defined process. Figure 2 displays an exemplary document flow with tasks that generate, modify and combine different documents / document versions and pass them on to the next task according to the currently executed administrative process, which is drawn as arrows in-between the tasks.

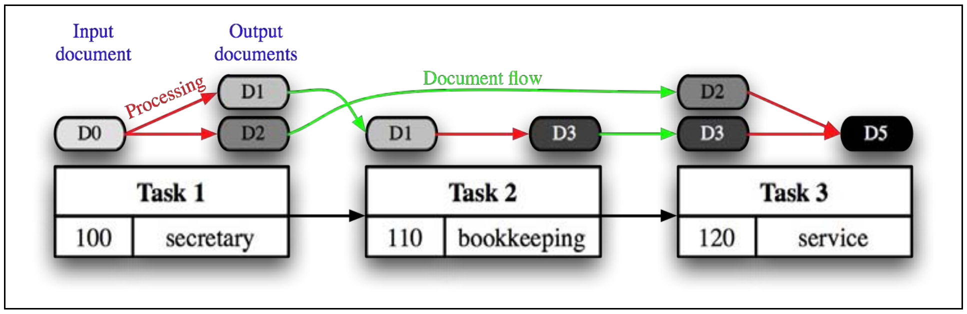

Figure 2: Task-based nature of a document flow

Note that documents are modified only during the execution of the tasks, but not within the document flow itself. This emphasizes the independence from software support and indicates that a document flow may be a purely manual work.

The probability of an unavailable input document for a task increases with the degree to which the document flow remains unmanaged. The increase in unavailability is due to the need for additional communication interactions between the sender and the receiver of a document and the fact that the sender may remain unaware of the task for which the input is needed. Therefore document flows are often supported by a central document storage and management system that may also support the enforcement of simple workflows. Both the lack in management and the confinement to a single format pose drawbacks to the simple document flow, which can be relieved through the use of the central management system outlined after the tool chain approach, which allows for using different tools.

Communication between different tools – a tool chain

On the other end of the methods to pass information, tool chains provide a less document-centric and more view- or tool-oriented approach. A tool chain can be considered a group of software programs that is executed in a certain order to obtain a certain result. Within the execution, subsequent programs rely on the output of their immediate predecessor to accomplish their assigned task.

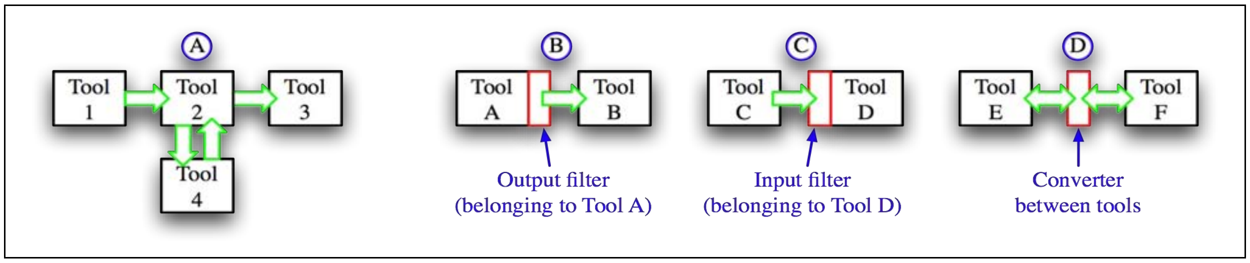

Figure 3: Tool chains

Figure 3a) is a schematic overview of a tool chain arranged in a star pattern.
Compared to the document flow, tool chains are technically more challenging to implement, since they require one tool to understand the data used within another tool. Tools provide different visualisations and interaction possibilities in order to fulfil the specific need for which they were designed. Different, related purposes are often implemented in tools of one company, because they form the company's area of expertise. The use of more than one of these tools is almost natural and hence tool chains frequently consist of tools from one vendor.
Another reason for connecting tools is the benefit of communicating with a widely used tool from a different vendor. This collaboration with an already used tool may be a central sales argument. A third reason for tool chains is the need of an engineering company, which relies upon tools from different vendors. The transfer of data from one format to the other may be extremely frequent or very sensitive and require consistent transformations and thus urge the company to start the development of a software solution.
Depending on the tools, an implementation may rely on a direct online communication of tools (e.g. implemented within ToolNet (Monteiro and Maeder 2007)) or a document flow between tool instances.
In both cases, the transfer of information between tools requires a transformation of data formats. Figure 3b) to d) provide a graphical overview of the three different types of transformation mechanisms used within tool chains.
The first is the output filter, which is attached to the tool providing the information. This is the mechanism of choice, if the communication to a wide-spread tool or format shall be supported. Examples are the OpenOffice Writer, which can write Microsoft Word format documents or Telelogic Rhapsody, which allows direct calls of Matlab methods to execute mathematical calculations. In addition to the output filter, the OpenOffice Writer has an input filter for Word documents, but Rhapsody models cannot be referenced by Matlab routines. This is symbolic for most tool chains: a document-based tool usually combines readability and writability of another format, whereas online communication generally remains uni-directional.
Third party software development companies rely on standalone converters, which allow for bi-directional transformation of inputs and outputs, as binding new functions into an existing

proprietary tool requires excellent knowledge of the internals of the tool. The technically most primitive converter that can be used for any tool communication is the skilled employee. This solution may be ideal in cases, where only few communication events between two tools occur, but – as mentioned before – may lead to inconsistent transformations.
The direct connection of tools has severe drawbacks regarding the exchange of information between different people. Online connections require the involved applications to be running. In addition, most software is still desktop-oriented rather than executable on a central server, so involved developers need to run development tools in parallel and have them connect to each other via the network. Both the availability of the tool's services over the network and the parallel execution are difficult points to master.
The other paradigm – basing the exchange of tool data on a document flow – exhibits the same problems as the simple document flow explained before: documents must be actively passed on. One solution is to store documents centrally so that the document flow exhibits more "pull"- than "push"-characteristics. This approach is explained within the following section.

Centralized data storage

A truly centralized document storage encompasses all development documentation as well as business information within a company that is used by more than one person. The transfer of information across the boundaries of a team or department is always a delicate matter as it is connected to responsibilities, customer agreements and territorial claims. This delicacy may reduce the effects of a central document management in inter-departmental projects. Therefore, an effective document management system has to differentiate between project-specific and company-wide data and implement access rights.
In accordance with their access rights, users of the document management system may navigate through the documentation. Target-oriented navigation and access policies are simplified through the hierarchy introduced by document folders. Meaningful links between documents further contribute to a successful search. Both the links and the hierarchy generally have to be introduced by human users. Guidance for establishing links between documents is provided by associated metadata information.
In the remainder of this paper, we will assume that the document metadata as well as a copy of each version of a document is saved within a database and abstract the other services of the document management system into a database services layer, which provides organized access to the data contained within the database. The database services constitute the central layer of the system architecture as they provide access and search functionality for data contained in the database (shown in Figure 4).

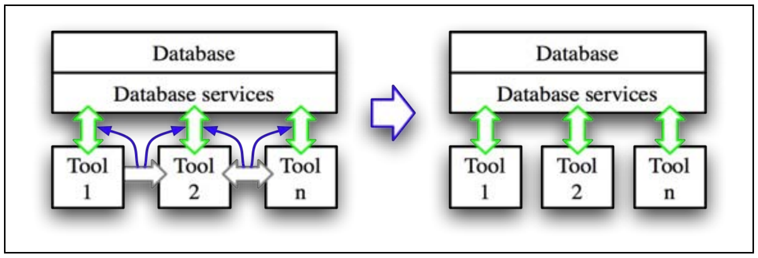

Figure 4: Database approach

The diagram also displays the reduction of inter-tool communication. Naturally, tool chains relying on direct, online communications between tools cannot be amended so that the tools communicate via the database, whereas tool chains based on a document flow are easily transformed. The main difference is the location, from which the documents are retrieved. A second modification is the handling of different types of documents by the database services.

Document types

Basically, there are three different types of documents: two types of documents discerned by the format of their content and the documents storing metadata (as depicted in Figure 5). An input document for a task may be created for a human reader and contain only plain and unstructured content such as a text file. Structured documents in contrast contain data in a pre-defined format, that can easily be interpreted (e.g. an XML document with associated XML schema). The difference between the two is the degree to which the contents can be read, “understood” and automatically processed by a software system.

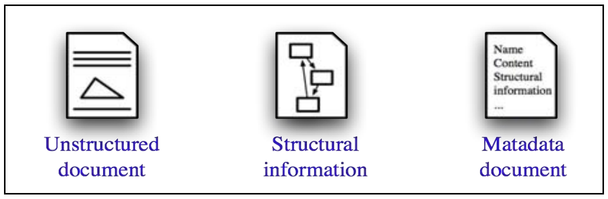

Figure 5: Document types

Metadata documents are structured documents themselves, which contain a list of attribute-value pairs belonging to the associated document.
The architecture of the system may provide a different handling of the document types to better support the development processes. Therefore, the next section will introduce three architectural levels of database solutions and explain their particular benefits.

A hierarchy of database solutions

Three conceivably different database architectures can be found, which are characterized by their ability to provide support during the development process and assistance in finding relevant documents. Higher levels also aim at providing a better insight into the product being developed and interdisciplinary implications. In Figure 6, the three architectures are shown. The least complex is described as Level 0 architecture and lays foundations for the enriched system architecture of Level 1 and therefore also for Level 2.

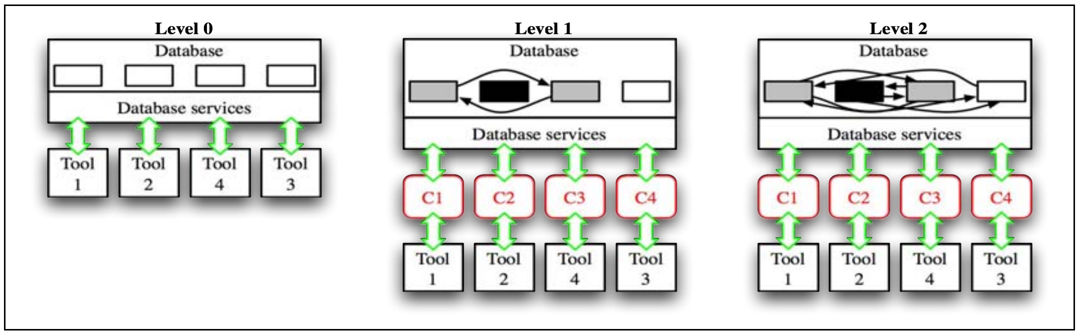

Figure 6: Database levels

Level 0. The architecture used within Level 0 is very similar to the tool chain approach explained before. Its main difference is the central document storage, to which tools can send documents and from which a tool may retrieve documents in a format it can read/write.

The benefit compared to the pure tool chain approach is the availability and versioning of documents as well as the annotation with metadata.

In this basic database solution, metadata consists of file information including:

- name and
- owner/creator of the document
- date of creation and when the document was last modified
- file size and type
- version information and
- access restrictions

Information about the editor of a document can also be used for notifications about changes. Notifications may be issued in form of a message such as an email or in a dashboard-like summary upon login at the database services of the document management.
In current implementations, the document themselves are checked in to the database system without any transformation of the format. In order to assure that the system realizes changes in documents, a checksum calculated from the contents of the document may also be placed within the metadata. Finding relevant documents may be based upon this metadata as well as (semantic) searches through the contents of the documents (e.g. as in (Castells et al. 2007)).

Level 1. This system setup is built upon the previous level and its idea of a central database that is connected to different modelling and documentation tools. One modification is the way documents are added to the database (refer to Figure 7 for an overview). In Level 1, additional metadata is extracted from the contents of the document. The additional metadata is content-based rather than file-based as in the previous level. It is used for summarizing information about the content, which allow for broader searches, more accurate indications about the similarity of documents and resemble bibtex information. Bibtex is a widely used storage format for document information and is commonly used for bibliographic collections. The bibtex format allows for a classification of the document (e.g. website, book or journal article) and defines meta-information specific for each class of document (e.g. a URL for a document of type website) (Fenn 2007). Information stored within the bibtex attributes has to be entered by the user to a large extent.

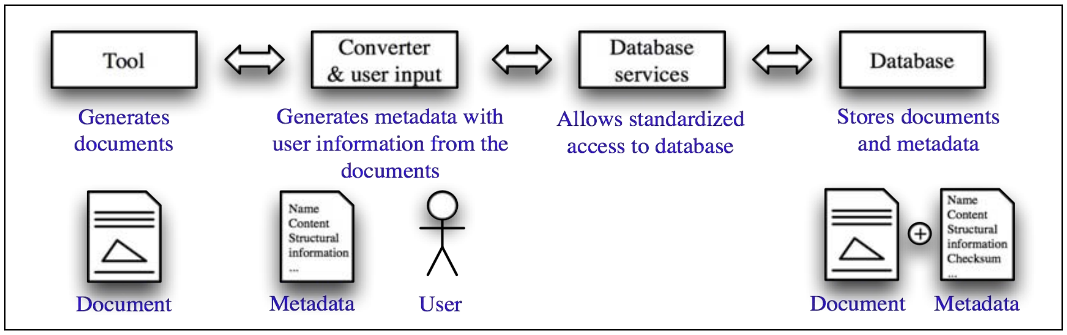

Figure 7: Converter

Other parts are extracted from the contents of the document by the converter. For the extraction of such data, the converter has to discern between structured and unstructured documents. Unstructured documents are directly copied into the database and only simple information is extracted from the content (e.g. headlines). At the same time, the converter may also fill predefined attributes of the bibtex-style metadata.

In contrast, structured documents are cut into pieces according to their internal structure and these are saved together with a copy of the original document. This way, structural information is preserved within the database and may lead to faster location of information and a more accurate description of the data contained within the document.
The different transformations of the converter as well as an overview of the check-in process for documents are explained within the following subsection.

Handling structured and unstructured documents by converter

The processes of checking in a structured or an unstructured document do not differ greatly as shown in Figure 8.

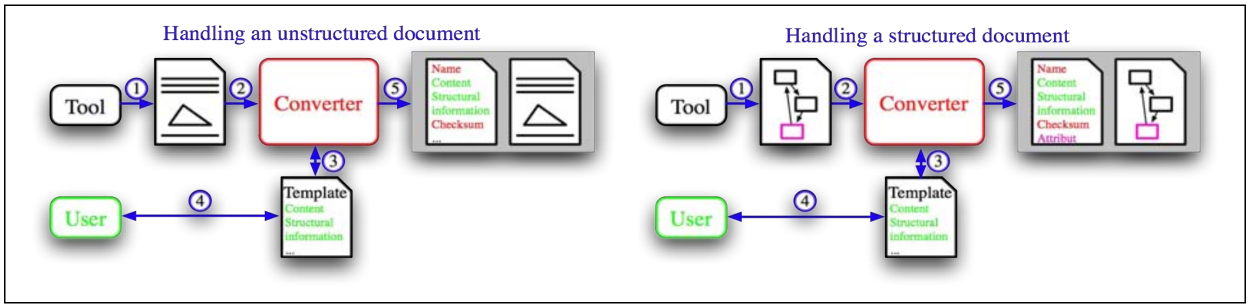

Figure 8: Handling structured and unstructured documents by the converter

In a first step, the tool creates or modifies a document, which is saved locally in the user's workspace. The user then notifies the converter of his check-in request for the new document version and is asked to fill a template containing bibtex-style attributes. Upon completion of the user's input, the converter adds its own specific attributes about the document, which may be a checksum or content-derived information and subsequently places the checked-in document as well as the metadata document within the attached database.
The difference between handling an unstructured and a structured document is the additional extraction of the internal structure, which can be made available by the owner's choice. In Figure 8, the bottom-most rectangle of the structure of the document may be chosen to be exposed in a special attribute, which is added to the metadata. The attributes of this rectangle may later be used to understand design decisions or calculate a similarity rating for search results.

Level 2. Similarity is a substantial concept, as it is fast to implement, easy to understand and supports the discovery of relevant documents. Up to Level 1, a search is the only way to exploit similarity information. Level 2 adds an explicit way of linking metadata to indicate similarity or the lack of it. This linking manifests itself in links between metadata documents (which themselves are associated with a versioned content document) or between attributes of metadata. Linking attributes adds a semantic indication of why the link suggests similarity, whereas links to metadata documents provide no meaning other than that the associated documents are similar to some degree.
As it is based upon the architecture of the level 1 solution, level 2 can extend the converter concept to automatically add similarity links between documents using a set of similarity rules. Emerging links may not always be correct, as the automation may lead to errors. Considering the hypothesis that two documents are similar exactly when a link between the two exists, there is a Type I error of “false positives” - documents evaluated as being similar, which are not – and a Type II error (“false negatives”), which subsumes similar documents that are not linked by the converter. One solution to minimize the errors is a careful adaptation of the underlying rule set.

Not even a carefully adjusted ruleset can guarantee perfect linking and therefore domain experts are needed to evaluate the meaning and correctness of all links. The expert may put a relation of one of the four types: equivalence, similarity, dissimilarity or indifference between two documents.

Relationship types between documents

The basic relationship concepts are indifference and similarity. An indifferent relation is a non-existing or not specified relationship between two documents or attributes. Similarity corresponds to the automatically deduced relationship that – as explained before – may be interspersed with errors. In order to clear out these incorrect relationships, a user can explicitly remove a similarity relationship and thus transform it to an indifference between the two documents or attributes. These two fundamental relationships are exemplified on two documents containing geometrical shapes within Figure 9.

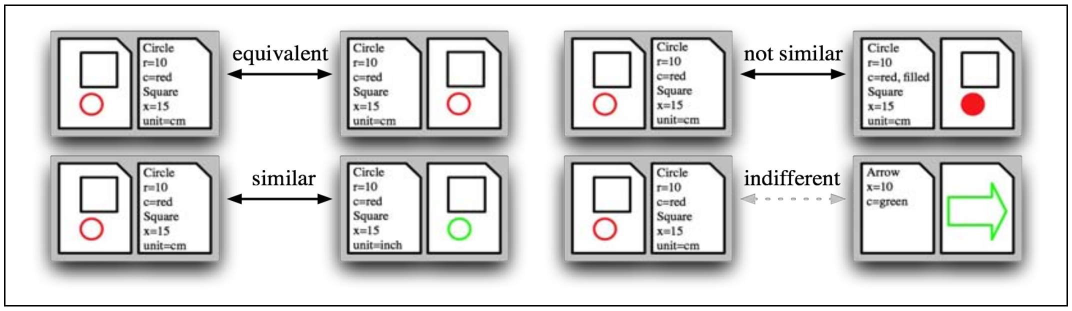

Figure 9: Example – relation types

Relationships, which at first glance may look appealing, are not always correct. Dissimilarity is introduced to explicitly account for these cases and to document incorrect similarities for other developers. Figure 9 illustrates this with the filling of graphical elements as strong indicator for similarity.

A fourth relationship class that can be used between documents or metadata attributes is equivalence. This concept is very important for interlinking models contained within structured documents and exposed through special attributes. Two structural elements marked as equal allow relating the two models to which they belong. In addition to the simple equality, which relates two structural elements that match in all of their aspects, the equality may also be parameterized. The equivalence parameter may be the reference to an attribute (in case of relating documents) or a formula to calculate one attribute value from the other.

An example is the size of a square measured in inches within one document and an equal square within another document, whose size is given in centimeters. The equivalence relation between the two size attributes would have to carry the formula for translating the inch-value into the centimeter-value (or vice versa). In place of this mathematical formula, logical expressions such as OCL (Object Constraint Language (Object Management Group)) may also be used to relate documents or attributes.

These four different relationship types can be seen as different degrees of similarity and may therefore be implemented as one type of relationship having an attribute that indicates this degree and an explicit mapping of degree to relationship concept (e.g. negative degrees indicate non-similarity, zero is equivalent to an indifference, positive numbers belong to similarities and a positive degree above a certain threshold is equality).

As an illustration of the data and the relations held within the database, Figure 10 relates attributes of a CATIA model to those of an Eagle-CAD model. The contents of the input documents can be related although they seem to be displaying completely different circumstances (i.e. a geometrical model and an electrical drawing).

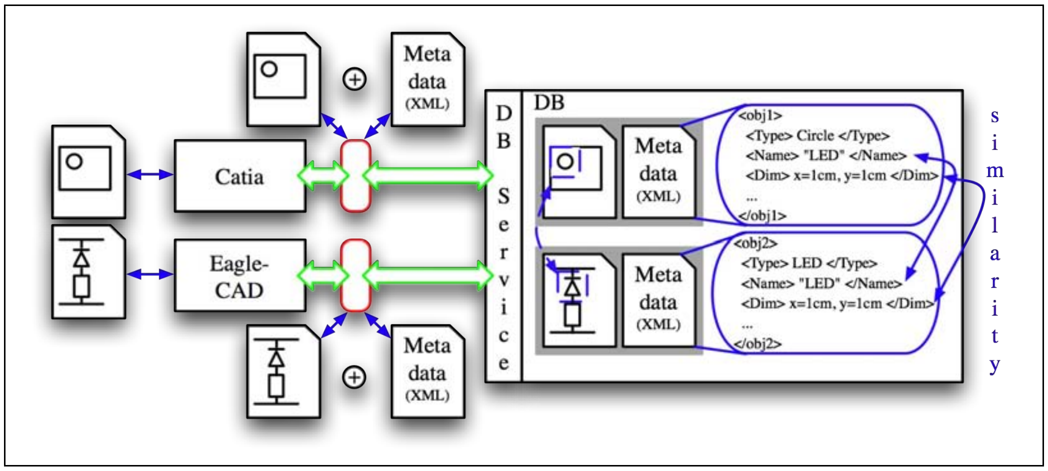

Figure 10: Concrete use of the database architecture

Within the previous chapter, an architecture for a centralized database storing and relating development documents has been laid out as well as its intermediary forms, which were developed from existing solutions. Taking the Level 2 approach, the next chapter provide a short overview of the development phases and subprocesses that are supported, newly introduced or modified by its database architecture.

Processes for the database approach

Currently, the (model-based) design of technical systems is carried out in parallel creation of models using a multitude of software tools. The models of these tools are in general separate and the knowledge put into them is exchanged informally and offline between the development team members (which is indicated within Figure 11).

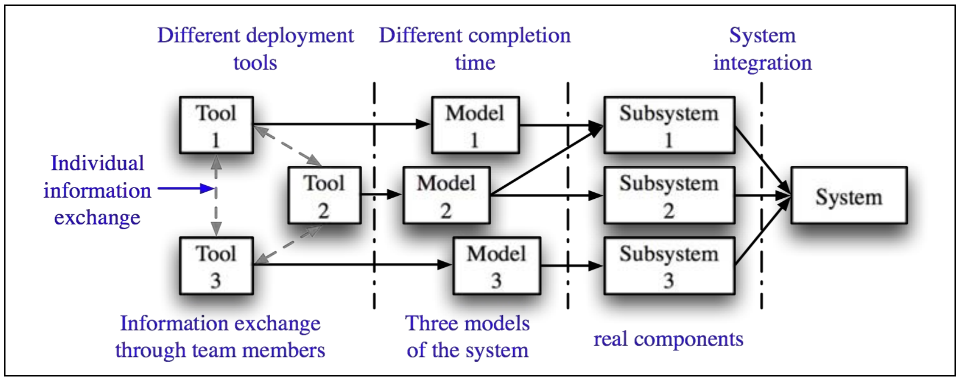
Different deployment tools
Different completion time
System integration
Individual information exchange
Tool 1
Tool 2
Tool 3
Model 1
Model 2
Model 3
Subsystem 1
Subsystem 2
Subsystem 3
System
Information exchange through team members
Three models of the system
real components

Figure 11: Classical development process

During the design process, the models created within the tools are refined and extended to reflect design decisions and compromises made. At some point, a model is complete enough to allow for its implementation within a real (sub-) system. Models created during the design process are not ready for implementation at the same time due to dependencies between models, delays during their individual development and the necessary correction of integration defects. It is obvious that the uncoordinated evolution / offline update of models leads to longer lead times and therefore to a larger difference in completion times. As the last model that provides input to

a subsystem determines the start of its realization and therefore indirectly the begin of the implementation of the system or prototype, better coordination may bring about a shortened development time.
The approach presented within this paper aims at reducing these lead times and providing support throughout the development process by interlinking documents (and thus models) within a central database. Concurrent access to the data stored within this database as well as version management and links to related documents yielding the current editor in charge are the basic strategies pursued by this solution. Explicit support for certain phases of product development supplements the approach.

Extending the phased development process

Hardware as well as software can be developed in various ways, which all follow to their own paradigm. A wide-spread process model is phased development, which is standardized for instance in the ECSS-M-30A standard for the development of spacecraft for and by ESA.
This standard provides the foundation for the amendments and subprocesses discussed within the remainder of this paper, but can easily substituted by any other phased development process (e.g. the V-Model (INCOSE 2007, 3.3) and especially fits spiral development according to Boehm (Boehm 1988, 61)).
The phases of the development process occupy longer stretches of time, and therefore require additional shorter organisation processes that are executed several times during a phase. These organisation processes are commonly termed „micro processes“, whereas the lifetime process is referred to as „macro process“.

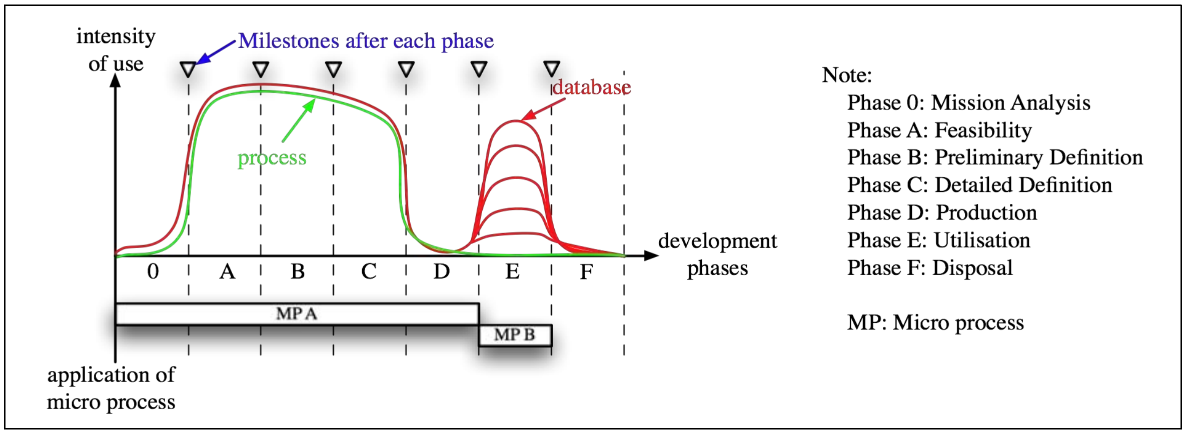

Figure 12: Example for tool and micro process usage during ECSS-M-30A phases

An implementation of the database approach presented within this paper does not support all phases of the macro process equally, but has more and less intense phases of use. The red curve within Figure 12 gives a qualitative overview of the intensity of use of the database: it starts out slowly with abstract models created during phase 0 (the mission analysis) and rapidly increases with the feasibility analysis, continues through preliminary and detailed definition and drops to almost zero accesses within the production phase D. Phase E may exhibit different intensities of database use according to the type of product that is built. A satellite, which can hardly be serviced within its orbit, or a throw-away product will need less database support than a wide-spread support-intensive type of product (see the section about Micro Process B for details). The disposal phase F requires few accesses to the database.
Figure 12 also features the number of executions of the micro processes involved in this

approach (the green curve). The number of executions follows the database usage roughly, but decreases constantly from the production phase onwards. This curve displays the summarized executions of the new and modified micro processes qualitatively. Micro processes applicable within each phase are indicated below the graph:

- Micro processes A (MP A) are executed during the design phases A to C and depict the typical check-in and discovery procedures for a development document
- During the support and maintenance phase, Micro process B uses the documents in the central database to provide helpful advice

Additional micro processes

Two different micro processes are employed in addition to or modification of other micro processes of ECSS-M-30A in combination with the level 2 system architecture. The execution of these micro processes is limited to certain phases of the macro process, as indicated by Figure 12.

Micro processes A (MP A). The additional micro processes subsumed under A are frequently executed during the feasibility, preliminary and detailed design phases. The first process within this group is concerned with the check-in of documents. A second process aims at providing consistent access and modification of documents within the database. This process should also be carried out during the production phase, but benefits this phase by a consistent access to documents only.

The process of checking in a document is detailed through these four steps:

1) *Creation of a new document*

Through the use of a modelling or development tool, a model is created or modified. Saving this model creates a file containing the text or structured information that was entered.
Note that any component taken from a supplier requires its documentation – or at least its specification – to be checked in as well as specifications of all alternatives that were considered. This facilitates the use of the database in later phases of the product development macro process.

2) *Attribute filling with the converter*

Upon logging in at the central database service, the user calls the upload routine for the newly created document and a specific directory (or abstract place) that can be referenced through the database services. The converter for this document format is called and automatically extracts predefined metadata attributes and structural information from the document (e.g. the editor, a checksum and maybe the size of the largest physical object). Then, the user is asked to fill in values into a list of attributes to retrieve bibtex-style information (e.g. the type of document, other editors or the accessibility scope of the document).

3) *Saving the document and its metadata*

The converter passes both the document and its metadata document via the database services to the database. A new document will be saved directly, a modified document as new version.

4) *Update and creation of similarity relationships*

Upon saving the document, the database service looks for similarities within the attributes of documents in the database index (e.g. the name of the document). The database services report all results found to the user, who checked in the document. The user may review the similarities found and change them to a different relationship type (please refer to page). Additional similarities, which have not been reported by the database service can be added manually by the user. A search function is implemented in the database services for this purpose. Altered similarity information is finally transferred and laid down in the database.

The second micro process within this process group is concerned with modifying documents or metadata within the database. Its steps are:

1) *Retrieving and modifying the document*

Using the built-in search mechanism for document attributes or traversing links, a reference to a document is obtained from the database services. The retrieved document is saved locally and may be modified or deleted by the user.

2) *Locking the document for modification*

Before the modification or deletion of the document or one of its relations is carried out, it is locked and the owner of the document is informed about the operation. The owner has to accept the changes in order to write them to the database and unlock the document. As long as the acceptance is pending, a viewer of the document information will get the remarking message that there are inconsistencies within the document or its metadata and that it is locked and cannot be modified further.

Micro process B (MP B). Phase E has its own micro process tailored to the use of the database. The aim of this process is the simplification and support of spare parts management and re-development.
The steps within this non-linear process are ordered in the flow chart of Figure 13 and follow the decision tree detailed in the following.

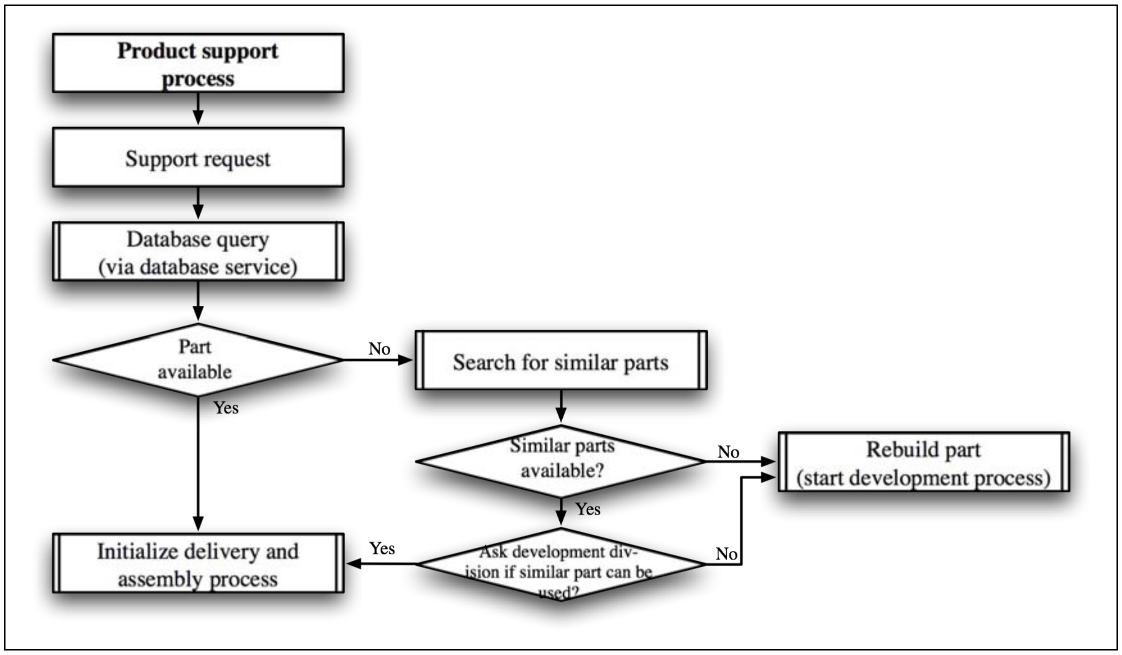
Product support process
Support request
Database query (via database service)
Part available
No
Yes
Search for similar parts
Similar parts available?
No
Yes
Rebuild part (start development process)
Ask development division if similar part can be used?
No
Yes
Initialize delivery and assembly process

Figure 13: Product support process

Upon a support request for a broken product, the person in charge issues a request to the database services (if information about spare parts is saved within the database) on whether a replacement component is available in the company's storage place. If the correct version of the spare part can be picked up, it is built into the broken system. If replacement components need to be acquired, the support tries to find a spare part of the same type and version as the broken part. The unavailability of the same version or type of spare part leads to the search for similar components. The check-in of supplier alternatives during the design phases pays at this time, as it allows for a deduction of possibly fitting parts from the database. In the worst case – if there is no alternative left – the part may have to be reproduced or even redeveloped (such as parts of the space shuttle).

The micro processes presented within this chapter complete the ideas of the architecture labeled with level 2. This level of architecture does not pose the ultimate assistance for multi-disciplinary development, but a start into supporting it.

Outlook and future prospects

Further enhancements will include the interpretation of the model data being gathered. This will require a strong semantic foundation of the models so that their interrelations may be deduced from a common model. Syntactic languages for supporting such a semantic layer are currently under way (e.g. RDF, OWL), but their formalization is not yet sufficient enough to be used for denoting relationships between complex models from different domains and tools.
Meanwhile, the database approach explained within this paper awaits being used in Concurrent Engineering applications, which aim at bringing developers and stakeholders together early on during the development process to synchronize and test individual design propositions. This application of the presented methods in a Concurrent Engineering environment will show and verify its utility. In addition, an adaptation to more agile process models (e.g. spiral development), which may especially benefit from the good accessibility of development data, is a befitting extension.

References

Abadi, Martín, and Luca Cardelli. 1998. *A theory of objects*. Monographs in Computer Science, Springer.

Boehm, Barry W. 1988. A Spiral Model of Software Development and Enhancement. *Computer* 21 (5): 61-72.

Castells, Pablo, Miriam Fernandez and David Vallet. 2007. An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval. *IEEE Transactions on Knowledge and Data Engineering* 19 (2): 261-272.

Fenn, Jürgen. 2007. Managing Citations and Your Bibliography with BIBTEX. The PracTEX Journal, no. 4 (May 25), `http://www.tug.org/pracjourn/2006-4/fenn/fenn.pdf` (accessed November 23, 2008)

INCOSE. 2007. *Systems Engineering Handbook Version 3.1*. International Council On Systems Engineering (INCOSE).

Monteiro, Manuel Reis, and Patrick Maeder. 2007. ToolNet – Domain and Tool Connector. Paper presented at the International Symposium on Grand Challenges in Traceability (GCT'07), March 22-23, in Lexington, Kentucky, USA.

NASA. 1995. *Systems Engineering Handbook SP-610S*. National Aeronautics and Space

Administration.http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-892JFall-2004/9722DD5E-CDBB-4B0F-8F5E-791CFF5FD359/0/nasasysenghbook.pdf

Object Management Group. Object Constraint Language - OMG available specification. Version 2.0. Object Management Group. http://www.omg.org/spec/OCL/2.0/

Petrasch, Roland and Oliver Meimberg. 2006. *Model Driven Architecture*. Heidelberg: dpunkt.verlag.

Yang, Q. Z., Dragan Domazet and Yizhi Zhao. 1997. Development of a Step-based Information Server for Concurrent Engineering Applications. In *Advances in Concurrent Engineering*, ed. Subra Ganesan and Biren Prasad, 255-262. International Society for Productivity Enhancement.

Biography

Carolin Eckl is a computer scientist, who studied at the Technische Universität München until 2007. Based on her minor aerospacial engineering, she is currently pursuing her phd at the chair for Astronautics at the Technische Universität München and working as a systems engineer with Berner & Mattner in projects for large German and international companies.

Markus Brandstätter studied computer science at the Technische Universität München from 1996 to 2002 and works since 2004 as a research scientist at the Institute of Mechanical Engineering TU München at the chair for Astronautics of Prof. Dr. rer. nat. Walter Systems Engineering Group. More information can be found at: http://www.lrt.mw.tu-muenchen.de.