

Integrating Engineering Databases: How Does the Application Domain Influence the FDBMS Architecture?*

Markus Strässler

ABB Corporate Research Ltd.
CHCRC.C2
Segelhof
CH-5405 Baden-Dättwil
Switzerland
Email: {straessl|mschoen}@ifi.unizh.ch

Martin Schönhoff

Department of Information Technology
University of Zurich
Winterthurerstr. 190
CH-8057 Zurich
Switzerland

Abstract

Federated database technology provides a basis to build integrated engineering environments. The specific requirements of this application domain influence the architecture and implementation details of a federated database management system (FDBMS) built to integrate engineering applications. This paper introduces an application scenario of an engineering environment, which is used to illustrate the requirements to be satisfied by an appropriate federated system. New functionality required at the integration level is specified.

The influence of the application domain on the following architectural aspects is discussed: schema architecture, data model, object materialisation and caching, integrity control and update propagation, version control, and the adoption of heterogeneous local systems.

1 Introduction

The collection of all database objects and document files describing an industrial product (e.g., CAD drawings, 3D models, finite element analysis, fluid dynamic simulations) and its engineering and manufacturing process are considered as *product data*. This data is produced by many different heterogeneous engineering applications. Various versions of this product data are created while the design team is looking for the optimal solution.

In order to eliminate errors resulting from tedious manual or file-based data transfers between engineering systems, to eliminate data inconsistency across different tools and to provide a global view of all project-related data, a tight integration of the different engineering systems is a necessity. *Federated database technology* can be applied to build such integrated engineering environments (e.g., [SCC⁺97, SSD97]).

*Work reported in this paper is executed within the project IDEE (Integration of Data in Engineering Environments). The project is part of the Swiss Priority Program for Information and Communication Structures (SPP ICS, 1996–1999) sponsored by the Swiss National Science Foundation under project number 5003–045354. Additional information can be found at <http://www.ifi.unizh.ch/dbtg/IDEE>.

Many different proposals of architectures of federated database systems exist. This paper will show the influence of the application domain of engineering applications on several design decisions concerning the architecture of the federated system. Architectural aspects covered include the schema architecture, the data model, object materialisation and caching, integrity control and update propagation, version control, and the adoption of heterogeneous local systems (i.e., how they can be coupled to the federation according to their capabilities).

Considering the requirements and architectural implications, the conclusion is drawn that a tightly coupled FDBMS using an object-oriented data model that supports version management is best suited for the integration of engineering applications.

The conclusions presented in this paper lead to actual design decisions taken as part of our ongoing IDEE (Integration of Data in Engineering Environments) project [SSD97]. The project's main goals are versioning of design data and system-wide consistency based on an FDBMS architecture. These goals have been defined through the engineering-specific requirements collected in cooperation with our industrial partner, the electromechanical company ABB (Asea Brown Boveri).

The remainder of this paper is structured as follows. Section 2 presents a small but typical application domain scenario. Thereby the integration requirements and the properties of different engineering applications are introduced. Section 3 illustrates the influence of these requirements on architectural design decisions of the federation. This includes remarks about actual decisions taken in IDEE. Some related work is introduced in Sect. 4. Section 5 concludes the paper.

2 FDBMS Application Scenario

The following scenario introduces concepts and terminology needed in the industrial application of an integrated engineering environment. It reflects our experience in the gas and steam turbine development departments of ABB (Asea Brown Boveri) in Baden, Switzerland.

2.1 Systems to Be Integrated

We introduce three typical, yet very different systems and present their integration-related properties. The three systems, which are shown in Fig. 1 with the flow of their product data between them, are:

1. a design calculation tool,
2. a CAD drawing and construction tool, and
3. a BOM (bill of material) tool.

At the beginning of a design process, different kinds of design calculation tools are often used. These tools take some order-specific requirements or design goals as input (e.g., minimal required efficiency, maximal allowed costs, steam temperatures and pressures). Given this input, they calculate the basic layout and characteristics of the product to be built (number of turbine blade rows, aerodynamic characteristics of steam flowing through the turbine, etc.).

Then, several CAx tools are used for the design of the product until it is ready for production. This often includes the use of a CAD tool to construct the product geometry. This tool

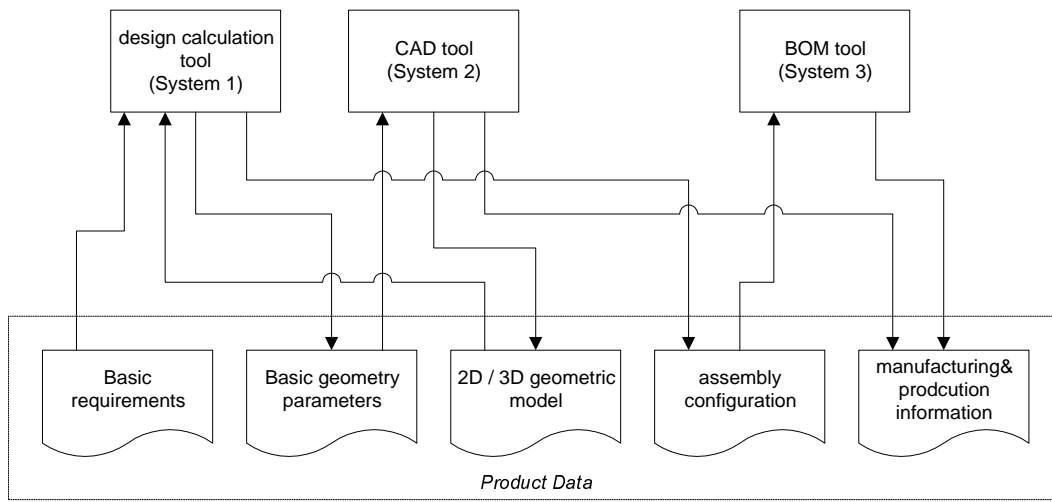


Figure 1: Three typical systems to be integrated and the data flow between them

also generates many of the required manufacturing data (including tolerances, roughness, manufacturing machine programs). The exact geometry can be used as input to the calculation tools which (re)calculate the properties of the actual design.

Within the design process, a complete list of all different parts used to build a product is compiled. This tree-structured list (“bill of material”, usually called BOM) contains hardware parts as assemblies, screws and bearing oil, as well as non-physical parts like assembly instructions and special treatments at manufacturing time. Assembly configuration (e.g., the number of screws) can be changed by the BOM tool as well as by the CAD tool. Both tools should therefore be able to share each other’s changes.

2.1.1 System 1: Design Calculation Tool

Design calculation tools are simulating the behaviour and the environment of physical parts of a product. These simulations are based upon aerodynamic and other physical formulas. Therefore, such tools are typically written in a procedural language like FORTRAN or C. They use several different input and output files to store their data. Various calculation sets are stored in different directories, managed by the user of these tools.

System 1: Calculation Program

schema	not explicit; “objects” defined as structures in procedural code and as formatted textual representation in the input and output files
data model	not explicit; “objects” in procedural language and textual input and output files
identifiers	no local identifiers available; data items are referenced by the absolute path to the file which contains them
integrity control	no local integrity control; the tool does not know the relation between input files (requirements, control files) and output files (results, analysis, plot files)

version control	no local version control; version control not integral part of application, but often file-based version control implemented using file-version control systems like RCS or CVS
external data access	data is stored in files which can be used as shared files of the local tool and the integration layer

2.1.2 System 2: CAD Construction Tool

CAX tools like CAD (computer aided design), CAM (computer aided manufacturing) and CAA (computer aided analysis, like finite element stress analysis) are used at different steps in the design process. Such tools are usually based on a proprietary internal representation of the product geometry and the product assembly structure.

System 2: CAD Construction Tool

schema	only a geometry-related schema exists; this does not reflect the specific product at all, therefore no product schema exists; different products are separated as different CAD models stored in different files
data model	proprietary; in some newer products, STEP/EXPRESS [ISO94a, ISO94b] is used
identifiers	some data item identifiers are available; items of different products have to be separated by different filenames of the files containing the CAD models
integrity control	local integrity control exists, but rules are not externally available
version control	partially supported version control of data items; since products are separated in different files, file-based version control is also used
external data access	data is stored in files which can be used as shared files of the local tool and the integration layer; data is also available at the proprietary programming interface (API) of the local system

2.1.3 System 3: Bill of Materials (BOM) Tool

Tools managing the bill of material and the product assembly structure are typical database-centred applications. Most tools use an underlying relational DBMS to manage the list of different product part items and their relations. The “catalog” of valid product configurations is also part of the data managed by such a BOM tool. The report generation functionality of the DBMS is heavily used to generate several lists of the product structure.

System 3: Bill of Materials (BOM) Tool

schema	only a structure related schema exists; this does not reflect the specific product, therefore no product schema exists; different products are separated as different data sets identified by product specific key values
data model	very often the relational data model is used
identifiers	some local data item identifiers are available; items of different products have to be separated by different key values

integrity control	local integrity rules are available
version control	partially supported version control of data items
external data access	access via SQL queries and commands and via API

2.2 Functionality Required at the Integration Level

When engineering applications are integrated, new functionality is possible at the global integration level, including

- global view on integrated data (allowing new global applications like project management tools),
- global version control and configuration management, and
- global integrity rules.

2.2.1 Global View on Integrated Data

Various engineering applications are used to perform tasks in different fields like aerodynamical simulations, stress analysis, manufacturing simulations and manufacturing machine programming, drawing preparations and overall plant planning. Each of the application developers and expert users in those fields has only a limited view of the overall product data. To actively discover inconsistencies and contradictions, a centralised and homogenised global view of the integrated product data is needed. This standardisation and homogenisation process is sometimes painful, but helps to reach a product- and customer-centred view instead of a organisationally structured view of the data.

The combination of the global view of the integrated data, integrity management and global versioning forms the basis on which global project management functionality can be built. This includes tracking of the design state of the different project parts related to release dates. In the case of additional, integrated workflow management support, the required release processes of parts and assemblies can be defined and tracked.

2.2.2 Global Version Control

Various versions of product data are created while a design team is looking for the optimal solution. The precise tracking of the history of product data enables engineers to discover cases in which dependencies between data have been violated. This means that parts of the data are based on preconditions (e.g., based on other product data) which were later changed without adapting all depending data. It also allows to track which versions of product parts fit together and form valid product configurations.

Therefore, beside the support of integrated data, integrated global *versioning* is needed in modern engineering environments which integrate different applications. This continues the tradition of engineers who manually trace versions of their designs on drawings and calculation documents.

In comparison to an “unversioned” federation, a federated database management system (FDBMS) *with* version management would offer additional functionality on its federation layer. This includes

- the propagation of new local versions to the federation layer,

- the propagation of new global versions to the affected local systems,
- the propagation of local update and delete operations to the global federation layer and vice versa, and
- the update of version-specific global relationships.

When we examine the engineering applications introduced above, we discover that they comprise different version history concepts. One can distinguish three classes of version histories, namely non-versionable objects, linear histories and more general directed acyclic graphs (DAGs) [SS98].

- The most general form of a version history is a DAG. However, most versioning systems do not support a general DAG, but rather a DAG with a single root or just a tree (i.e., a DAG with a single root and without “merged” branches). Also, a more general version history DAG can be mapped into a tree without unacceptable loss of information. Hence at least a tree-like version history is required. We call objects with such a version history *fully versionable* objects.
- Linear version histories represent an object’s development over time, i.e., only revisions can be created. We hence call objects with a linear version history *revisionable* objects.
- For *non-versionable* objects, the “version history” is a single version object that is updated “in place”. This kind of “version history” obviously does not provide any real version management, but represents the situation in systems without version management.

An FDBMS should provide the integration of objects of any of these three classes of versionable objects.

Engineering design systems also support *configuration management* using *relationships* between objects, e.g., the “is-part-of” relationship between a product part and an assembly. Such a relationship between versioned objects can be *static*, which means that it always points to the same specific version, even if a new revision of the referenced version is created. Or it can be *dynamic*, i.e., it always points to a distinguished *current* version, e.g., always the latest version.

An FDBMS integrating engineering applications should provide the basis for configuration management functionality (e.g., support of versioning and relationships between parts), thus allowing to build a complex global configuration management system as a global application.

Such version control functionality at the integration level helps us to implement the following tasks during the integration of our three scenario applications:

- global version control added to applications which did not yet include version control (calculation program, BOM tool),
- tracking of the current version of the different parts of the product data, relieving the engineers from doing this task manually,
- introducing new functionality to the engineers, such as the possibility to create new variants of product data without influence on the current version used by other engineers, and
- basic configuration management functionality helping to integrate full configuration management provided by the BOM program.

2.2.3 Global Integrity Rules and Update Propagation

The new integrated view of product data provided by a federated system also has to allow the representation of domain specific knowledge about data interdependencies. This knowledge about data redundancies and relations will be collected during the design of the federation in parallel to the schema design and will be maintained while the federation exists. The following example illustrates the problem to be solved by using data dependency knowledge.

A stress calculation performed by a design calculation tool is obviously no longer valid as soon as the geometric shapes which form the basis of the calculation have been changed (e.g., by a related CAD tool). Without integration of the two systems, the designer of the basic shape has to notify the engineer who calculates the mechanical integrity about design changes and send him the new data. Likewise, the stress analysis engineer must notify the aerodynamic designers of the results and proposed changes.

Global integrity rules define dependency relations between global data items which are representations of data items of different local systems. In order to resolve detected integrity rule violations automatically, update propagation from one local system to a related one via the integration layer can be used.

Figure 2 shows an example of redundant and dependent data in the local systems. It also shows the rules needed to ensure data integrity.

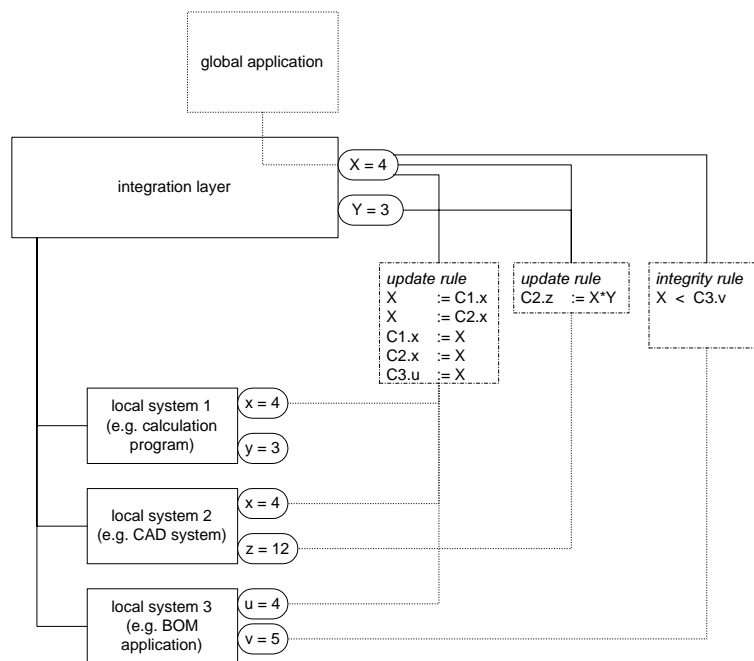


Figure 2: Update and integrity rules

Integrity rules define domain specific integrity constraints. They consist of a sometimes complex boolean function, checking the relationship of all affected data items. Whenever an update takes place, all affected integrity rules have to be checked and an appropriate reaction should be performed if the checking of one of the rules fails. In the worst case, this would enforce the abortion of the complete update transaction. In other cases, an evasive action like the creation of a new version can be possible.

Update rules define the relationship of dependent data and the appropriate strategy to propagate changes. Update rules are a subclass of integrity rules as they always ensure the dependency between the related items. One of the simplest example of an update rule is the

equality-update rule, which defines that two related data items should always share the same value. Whenever one of the values is changed, the second value should be updated to the same value. If this second update fails, the primary update should also be rejected.

Complex update rules define the relationship between two or more data items. These rules may also consider the actual operation or event triggering them, additional conditions and arbitrary actions needed to perform the update. Again, evasive actions could be defined to avoid the abortion of the current update.

3 Influence of the Application Domain on the FDBMS Architecture

In this section, the influence of the application domain of integrating engineering design applications on several architectural aspects will be illustrated. To each topic the according approach taken in the IDEE project will be introduced. The different architectural aspects to be covered are

- the schema architecture,
- the data model,
- object materialisation and caching,
- integrity control and update propagation,
- version control, and
- adoption of heterogeneous local systems (wrappers, homogenisation, local version control).

3.1 Schema Architecture

The first step to be taken is to use or adapt one of the two well known reference schema architectures for FDBMS, the one for tightly coupled FDBMS proposed by Sheth and Larson [SL90] or the other one for loose coupling [HM85]. Although the use of the latter one looks promising to support flexible exchange and enhancement of the integrated applications, tight coupling is preferred because of the following advantages:

- it supports a centralised global view on all integrated data,
- it enables the implementation of integrated versioning support at the global layer,
- it enables global updates, and
- it enables global consistency control and enforcement of global integrity constraints (global constraints ensure the consistency of engineering designs across several local systems).

In the engineering design domain, we often have to deal with applications which are used for experimental designs or applications which are even implemented by the design engineers themselves. Their local schemas therefore change almost as often as their data. It is neither manageable nor necessary to propagate each of these local schema changes to the component

schema of the federation (especially if there does not even exist an explicit local schema, e.g., in the case of a FORTRAN program). Therefore, we propose to use a component schema which only represents, in terms of the global data model, the parts of the local schema the component is willing to share in the federation. Thus, the component schema is a combination of the originally proposed component and export schemas of [SL90]. The resulting four level architecture (see Fig. 3) is used in IDEE [SSD97].

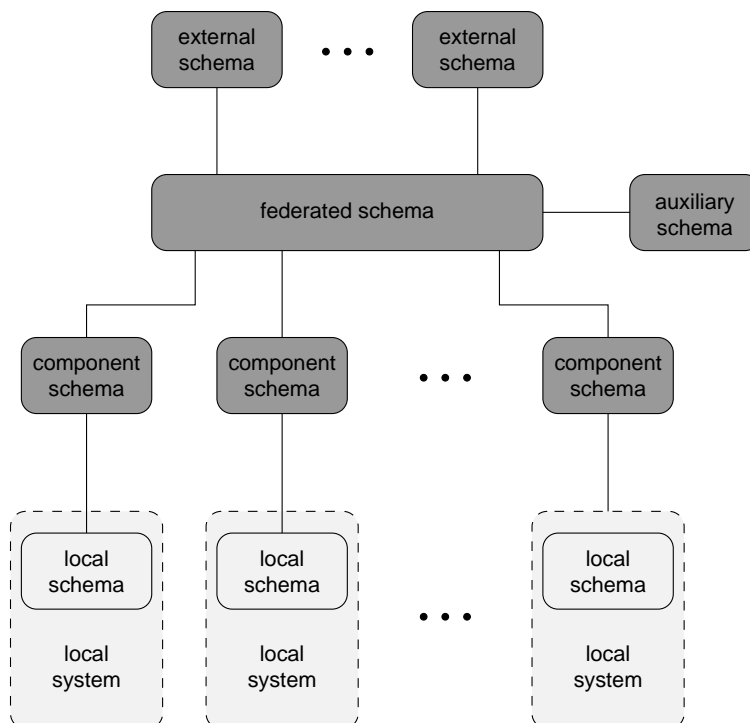


Figure 3: Four level schema architecture for IDEE [SSD97]

We introduce a single federated schema to logically centralise the integrated data. The *auxiliary schema* comprises the global constraints, data not available in any of the local systems, and federation management information.

3.2 Data Model

One also has to choose a data model to express the different component schemata as well as the federated schema and the external schemata. The local systems to be integrated provide heterogeneous local data models (e.g., the relational model or even no explicit model like system 1 in the example scenario). The global data model therefore has to be able to provide rich expressiveness (the degree to which an external concept can be directly represented) in order to represent all concepts found in the local data models. *Object-oriented data models* are regarded to be most suitable, though not perfect, for this task [SCG91].

In the practical implementation, a *standardised data model* is preferred, although standards do not include the latest research results and generally represent only the least common denominator of a standardisation panel. However, standards increase portability of both data and applications, because they are supported by many different vendors. It is therefore reasonable for the industry to enforce standardisation.

For these reasons, the IDEE project follows standards, too. Since our project applies federated database technology to engineering data and applications, both, database and engineering

standards have been considered. The choice was reduced to the most prominent representatives of either side: ODMG 2.0 [Cat95, CB97], the database standard of the Object Data Management Group, and the ISO STandard for the Exchange of Product data, STEP [ISO94a]. For IDEE, the advantages of ODMG 2.0 vs. STEP turned out to be overwhelming.

Advantages of ODMG 2.0 over STEP exist especially on the modelling side. STEP can only represent static structures, and therefore provides only *structural expressiveness* [SCG91]. ODMG 2.0 additionally allows for the definition of methods and therefore behaviour - it is *behaviourally object-oriented* [Dit86] and has *behavioral expressiveness* [SCG91] (which, of course, is advantageous for integration). ODMG 2.0 has been equipped with enough functionality to meet the needs of database application developers.

Nevertheless, ODMG 2.0 as the global data model does not help to solve all integration problems [CEH⁺97]. In fact, ODMG 2.0 needs to be extended in order to support federation specific requirements like the definition of different schemas [Rad96]¹. Furthermore, version management extensions must be provided. Other problems with the ODMG 2.0 include its query language OQL which may be suited for database experts, but rather not for engineers. Additionally, ODMG 2.0-compliant database management systems do not yet exist.

The strengths of STEP can be found in the specification of constraints and especially in the availability of standardised schemas (called “application protocols”) for many different application domains, e.g., technical CAD drawings [ISO94a, Part 201, explicit draughting]. If all local systems in a federation used (parts of) a standardised STEP schema as their component schema, the same schema could be used as the federated schema. Thus, neither schema nor semantic heterogeneity needed to be resolved during schema integration [BLN86]. But our area of gas or steam turbine development is not yet covered by a STEP protocol, so we would depend on our own non-standardised schema.

STEP as the global data model would obviously simplify the FDBMS architecture, because a single coupling module would be sufficient for all local systems. However, applications with a STEP interface are still rare in industry. Individual implementations of the remaining interfaces (which are “at least theoretically possible” [Dig92]) would be required. But with ODMG 2.0 as the global data model, a similar simplification is possible, too. Local systems with a STEP interface can be integrated through a single coupling module which translates STEP schemas into their ODMG 2.0 equivalents [Lüh96]¹.

3.3 Object Materialisation and Caching on Global Layer

The example scenario illustrates that typically most of the applications to be integrated into an engineering design systems are not database systems. They often consist of a specialised application which is started and then loads a project specific data set (e.g., a CAD model). Without caching, any read access to a local object of such a local system would eventually result in restarting the local application and reloading the according data set. This may be partially avoided, if read access requests to local objects are directly served by the integration layer. This layer should therefore support an object cache. Such a cache would also improve performance if the different local applications are executed on different systems, connected to the integration layer using (slow) network connections. This is often the case in industrial application domains.

Caching can be realized as part of the coupling module of each local application. Cached objects will therefore be described in the local data model of the application. This solution

¹[Rad96] and [Lüh96] were written with the old standard ODMG-93 in mind. However, federation-specific issues have not been addressed in ODMG 2.0 either.

results in several independent and application-specific caches. Such caches have the advantage of optimal integration with the local data storage. However, they violate local autonomy.

Objects can also be cached on the applications' component schema (solution A) or integrated schema (solution B) level, both times modelled in the global data model. Similar caches result if objects are buffered at either of these two levels. Both solutions only need a single cache as part of the integration level if the coupling module is running on the same machine as the integration layer.

The advantage of solution B is the minimal access time required, solution A is only slightly slower if the coupling modules are running as part of the integration layer as proposed above. The advantage of solution A is the fact that complete objects are usually loaded from local systems and, if necessary, whole objects are invalidated in the cache. This is not the case for solution B, since both, a component level object can be split up onto several integration layer objects and an integration layer object may consist of several parts of different component level objects. Furthermore, solution A makes it much more difficult to apply different caching strategies to objects which originate from different local systems. With solution B, on the other hand, the database integrator can, for example, enforce a write-through strategy for one local system and defer local write operations until the end of a global transaction for another system, if this is more appropriate.

We call an object *materialised* at a specific schema level if its object state (i.e., its set of attributes) is physically allocated at this level. Otherwise only access methods to get the object state from a lower level exist. To support caching at a specific level, the objects also have to be materialised at this level.

Caching is important in the engineering application integration domain, because the different local components are not optimised towards fast external data access. In the example scenario, for instance, access to complex structured files of calculation programs or to the internal databases of CAD systems may introduce long latency times.

The IDEE approach exploits the advantages of caching and materialisation at the component schema (solution B) shown above. Every domain object is identified by a related OID (object identity). These OIDs are also implemented as objects in IDEE (as shown in Fig. 4). As long as an object is only referenced by other materialised objects, but its attribute values have not yet been accessed, it is only represented by its OID object. The object itself is not yet materialised, which saves memory in the coupling modules and prevents the materialisation of any indirectly referenced object.

Access to an object's attributes is only possible using access methods (e.g., `get_A()` and `set_A(newValue)` to access an attribute A). This simplifies the implementation of the access to attributes at the integration level, since objects are only materialised at the component level. At the integration layer, no objects are materialised, but they exist only "virtually". The writing of a new attribute value to the local system only takes place if a new value, different from the one in the cache, has to be written.

It depends on the component system whether all or some of the cached objects have to be invalidated if changes are written to the local system or local operations take place in the local system, eventually changing other objects there. Due to this fact, we also classify methods into *destruction-free methods* which do not change local objects and *write methods* which update one or several objects of the local system.

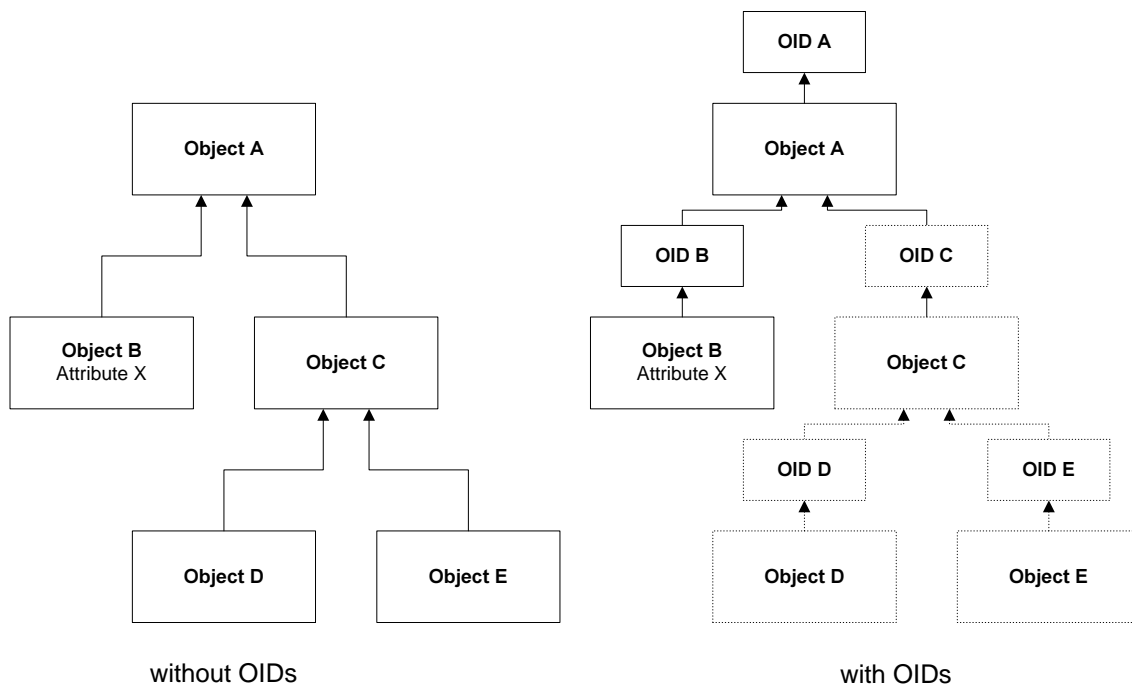


Figure 4: OIDs and object materialisation in IDEE

3.4 Integrity Control and Management, Update Propagation

In Sect. 2.2.3, we introduced the requirements on integrity control and update propagation. Local rules have to be integrated into global rules and additional global integrity rules are defined and have to be enforced. Typical examples of such rules are

- proportion rules (e.g., the diameter of a screw should always be smaller than the corresponding hole, but the clearance should not be more than 10 percent of the diameter),
- aggregation rules (e.g., the sum of all costs of the parts of a product should not be more than the price at which the product is offered), and
- transitional rules (e.g., the status of a part can only be changed to “in work” if the part is not yet in status “released”).

The following problems have to be solved while building the federated system:

- inconsistencies between local integrity rules have to be resolved
- additional new global integrity rules have to be identified
- inconsistencies between local and global integrity rules have to be detected and resolved
- local integrity rules may be transformed into global rules

Several research groups have already concentrated on the integration of local integrity rules during schema integration [SCC⁺97, Con97, RR97]. Several procedures to tackle the problem have been proposed.

The functionality of global integrity maintenance and enforcement at run time of the federation has to be implemented as part of the global integration layer. It has to be carefully designed

not to violate the autonomy of local systems and use existing local mechanisms for integrity control as far as possible.

To specify the task, we should differentiate between three categories of integrity rules [EN94]:

- *data model inherent integrity rules*: Such rules (e.g., inheritance rule in object oriented data models: each subclass inherits the features of its superclass) are characteristics of the data model and can therefore not be changed or defined explicitly. Those rules are automatically maintained by the global and local DBMS. Mismatches between the local and global data model need to be handled by implicit or explicit rules.
- *implicit integrity rules*: These rules are defined during the schema design of a database. Some of the rules, like key integrity in the relational model, are enforced automatically by the DBMS.
- *explicit integrity rules*: The rules are explicitly defined and have to be enforced by a dedicated module of the FDBMS. This module checks the adherence to the rules and starts appropriate actions, if the rules are violated.

Data models generally support inherent integrity rules and provide possibilities to define implicit integrity rules. But usually, these rules are not sufficient to describe all the integrity properties of the actual domain specific real-world extract. Additional explicit integrity rules have to be defined in either a *procedural* or a *declarative* way.

The procedural implementation of integrity rules results in the explicit formulation of the rules and the handling of rule violations in the application language of the user applications or an additional integrity control application. Using an object-oriented data model, the procedural integrity rules may be directly included into the implementation of attribute access methods.

The declarative implementation results in a rule base which contains the integrity rules in a declarative form, e.g., as ECA (Event/Condition/Action) rules. A generic checking mechanism located in the communication path between the local systems and the federation layer checks the rules and enforces them. Such rules can be completely defined by the following properties:

- the set of objects which are affected by the rule,
- the event that triggers the checking of the rule,
- the condition which must always be met, and
- the (re)action that is executed if the rule is violated.

A typical concept for the implementation of such an integrity control mechanism can be found in [TC96]. It builds upon active database concepts [Gat94, DG96] and therefore uses ECA rules. Figure 5 shows the integration of active mechanisms into the FDBMS architecture. Global integrity rules are checked and enforced by the global integration layer. However, this layer a priori has only information about global operations. Hence, local operations have to be monitored and the necessary events have to be sent to the global layer, notifying it about local changes.

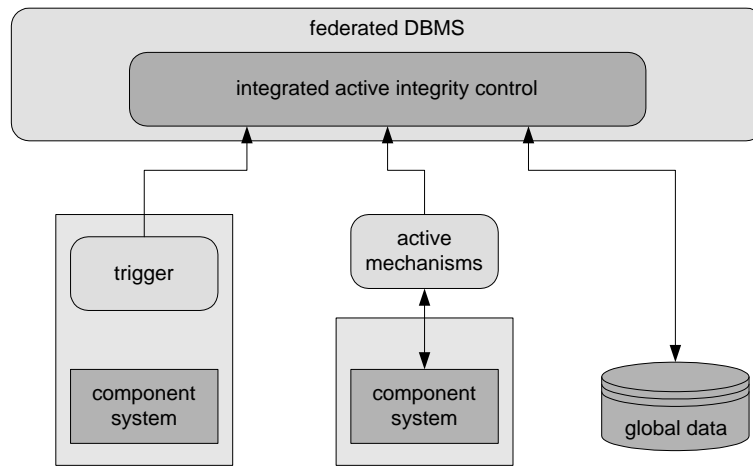


Figure 5: Architecture of an active integrity control mechanism

3.5 Version Control

As introduced Sect. 2.2.2, version management is essential in design applications and integrated design systems. Numerous data models with integrated version management have been proposed and implemented, many of them especially developed for the needs of engineering databases (e.g., [Kat90, KBG89, DL88]).

In IDEE, version management at the federation layer will be supported. Since the ODMG 2.0 standard does not include versioning, we will model a versioning mechanism by means of ODMG 2.0 constructs. Our mechanism will take care of different kinds of engineering objects, i.e., not only data objects will be versionable, but also constraints and calculation routines (both will also be modelled using ODMG 2.0).

Rules defining which version of a data object could be used with which version of a local system (e.g., a calculation tool) will be introduced. Designers should be able to access or reference the latest version and revision of an object without knowing the exact version number.

It is helpful and important for a designer to understand the differences between varying versions. The designer should be able to look up and visualise the changes that have been introduced between different versions. Because many different version and revision numbering schemes are in use in the industrial practice, IDEE should be easily adaptable to a new version numbering scheme.

3.6 Adopting Heterogeneous Local Systems: Wrappers and Homogenisation

Since we want to integrate existing engineering applications as component systems without changing them, the coupling mechanisms to connect them to the integration layer have to be based on the existing interfaces of these applications. Several different types (according to their interfaces) of local systems have been introduced in the scenario (Sect. 2). In total, five different groups of local systems have been identified, based on how the coupling modules may access local data (see Fig. 6) [SSD97].

- *Systems using a DBMS*

Coupling a CAx tool that uses a commercially available DBMS is done by (operational) mappings from the component schema to the local DBMS schema.

- *Applications with an external programming interface*

Such an interface offers commands which are equivalent to the user interface controls, as well as operations which access internal structures of user data.

- *Applications with an internal storage module interface*

A storage module interface provides the opportunity to access low-level application data (e.g., lines or 3D objects of a CAD program). Application objects and operations of higher abstraction (construction of assemblies, calculation of the centre of gravity of a complex part) are not accessible.

- *Shared data files*

If a local system does not offer any means to access data via the application itself, the data files of the application might be used. This raises the problem of synchronising access to the files.

- *File exchange*

In the case that the application's data file format is unknown or subject to frequent changes, the remaining possibility is to exchange files of some known (preferably standardised) exchange format. Since the synchronisation problem is even worse in this case, data from exchange files will be read-only at the global layer.

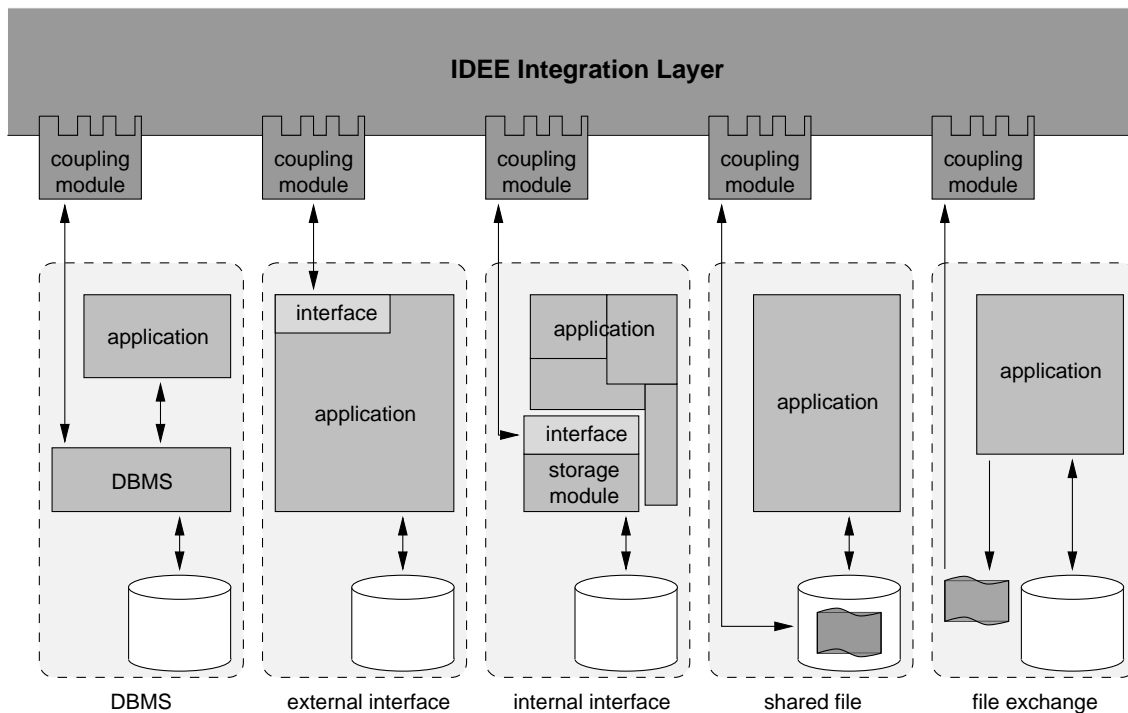


Figure 6: Interfaces to couple component systems [SSD97]

In order to support global consistency rules and update propagation, we have to provide basic functionality to monitor the updates that are autonomously carried out within local applications. Therefore, monitoring facilities have to be implemented as an extension of the local application (e.g., a CAD program) or – more likely – as an independent module monitoring changes of local data. The latter approach can further be subdivided depending on whether

accesses of local applications to their local data are directly caught, or whether alerters are used which are attached to the data (e.g., triggers in relational DBMS).

These monitors report changes to the global layer which cares for the proper propagation of updates to sibling systems. Possible monitor implementation concepts are:

- triggers
- notification services, event generation (explicit or implicit)
- tracking of trail files or log files
- tracking of file checksums to detect changes
- force the local system to create new versions upon updates and use the version management coupling to monitor changes.

4 Related Work

FDBMS have been a research topic for more than a decade now. Nevertheless, to the best of our knowledge, there are only few publications about the use of FDBMS for the integration of engineering applications. [Con97] provides a recent overview of the state of research in FDBMS. The book also includes references to projects that contribute especially to the needs of engineering environments.

In industry, product data management (PDM) systems are a practical approach to integrate several product data producing engineering-applications. However, despite their name, these systems control documents, i.e., containers, rather than the data themselves. They therefore cover the requirements on object versioning, integrity control and update propagation only partly. PDM systems also fall short of an FDBMS in many other aspects, most notably, transparency.

A five-level schema architecture [SL90] and an object-oriented global data model [SCG91] are generally accepted for most FDBMS [CEH⁺97]. [BNPS88] proposes operational integration to support a high degree of heterogeneity of the component systems. We adopted these approaches in the IDEE architecture, mainly because most engineering applications provide only poor support for their integration [SSD97].

5 Summary and Conclusions

We have presented the domain specific requirements and additional global functionality needed of an integration of several engineering applications.

With respect to the architecture of an FDBMS used as basis of such an integrated environment, we concluded that tightly coupled FDBMS using a four-level schema-architecture and an object-oriented data model like ODMG 2.0 or STEP are best suited to our application domain. Tight coupling is preferred against loose coupling although an engineering environment can not be seen as static (i.e., applications will change, join or leave the federation from time to time). The global view on product data provided by a tightly coupled FDBMS will allow new global applications like global project and product management.

Global versioning, integrity control and update propagation are additional essential global functions. Other integration specific parts of an implementation of an integrated engineering

environment like an object cache, object materialisation and component-wrappers can also be efficiently implemented as part of such an integration layer.

Acknowledgements

Many thanks go to Dr. Dirk Jonscher and Prof. Klaus R. Dittrich for their advice in the course of the IDEE project and on the preparation of this paper.

References

- [BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database system intergration. *ACM Computing Surveys*, 18(4):323–364, December 1986.
- [BNPS88] E. Bertino, M. Negri, G. Pelagatti, and L. Sbatella. The COMANDOS integration system: an object-oriented approach to the interconnection of heterogeneous applications. In *Proceedings of the Second International Workshop on Object-Oriented Database Systems (OODBS), Bad Münster am Stein-Ebernburg, Germany*, number 334 in Lecture Notes in Computer Science, pages 213–218. Springer-Verlag, September 1988.
- [Cat95] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93, Release 1.2*. Morgan Kaufmann Publishers, San Francisco, California, USA, 1995.
- [CB97] R. G. G. Cattell and D. K. Barry, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, San Francisco, California, USA, 1997.
- [CEH⁺97] S. Conrad, B. Eaglestone, W. Hasselbring, M. Roantree, F. Saltor, M. Schönhoff, M. Strässler, and M. Vermeer. Research issues in federated database systems. *SIG-MOD Record*, 26(4):54–56, December 1997.
- [Con97] S. Conrad. *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer-Verlag, September 1997.
- [DG96] K. R. Dittrich and S. Gatzju. *Aktive Datenbanksysteme: Konzepte und Mechanismen*. International Thomson Publishing, Bonn, Germany, 1996.
- [Dig92] Digital Equipment Corporation. Product data sharing using STEP technologies, February 1992.
- [Dit86] K. R. Dittrich. Object-oriented database systems: The notions and the issues. In *Proceedings of the International Workshop on Object-Oriented Database Systems, Asilomar, California, USA*, pages 2–4, 1986.
- [DL88] K. R. Dittrich and R. A. Lorie. Version support for engineering database systems. *IEEE Transactions on Software Engineering*, 14(4):429–437, April 1988.
- [EN94] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, Redwood City, Cal., USA, 2nd edition, 1994.

- [Gat94] S. Gatzju. *Events in an Active Object-Oriented Database System*. PhD thesis, University of Zurich, 1994.
- [HM85] A. Heimbigner and D. McLeod. A federated architecture for information management. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.
- [ISO94a] ISO IS 10303–1. STEP part 1: Overview and fundamental principles, Geneva, 1994.
- [ISO94b] ISO IS 10303–11. Express language reference manual, Geneva, 1994.
- [Kat90] R. H. Katz. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, 22(4):375–408, December 1990.
- [KBG89] W. Kim, E. Bertino, and J. F. Garza. Composite object revisited. *SIGMOD Record*, 18(2):337–347, June 1989.
- [Lüh96] H. Lührsen. *Die Entwicklung von Datenbanken für das Produktmodell der ISO-Norm STEP*. PhD thesis, University of Erlangen–Nuremberg, 1996.
- [Rad96] E. Radeke. Extending ODMG for federated databases. In *Proceedings of the Seventh International Workshop on Database and Expert System Applications, Zurich, Switzerland*, pages 304–312, September 1996.
- [RR97] V. Ramesh and S. Ram. Integrity constraint integration in heterogeneous databases: An enhanced methodology for schema integration. *Information Systems*, 22(8):423–446, December 1997.
- [SCC⁺97] G. Saake, A. Christiansen, S. Conrad, M. Höding, I. Schmitt, and C. Türker. Förderierung heterogener Datenbanksysteme und lokaler Datenhaltungskomponenten zur systemübergreifenden Integritätssicherung – Kurzvorstellung des Projektes SIGMA_{fab}. In *Proceedings of Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'97), Ulm, Germany*, pages 322–331, March 1997.
- [SCG91] F. Saltor, M. G. Castellanos, and M. García–Solaco. Suitability of data models as canonical models for federated databases. *SIGMOD Record*, 20(4):44–48, December 1991.
- [SL90] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.
- [SS98] M. Schönhoff and M. Strässler. Issues of global version management in federated database systems for engineering environments. In *Proceedings of the 10th Workshop “Grundlagen von Datenbanken”, Konstanz, Germany*, number 63 in Konstanzer Schriften in Mathematik und Informatik, pages 119–123, May 1998. Extended Abstract.
- [SSD97] M. Schönhoff, M. Strässler, and K. R. Dittrich. Data integration in engineering environments. In *Engineering Federated Database Systems (EFDBS'97) – Proceedings of the International CAiSE'97 Workshop, Barcelona, Spain*, Computer Science Preprint 6/1997, pages 45–56, University of Magdeburg, Germany, June 1997.

- [TC96] C. Türker and S. Conrad. Using Active Mechanisms for Global Integrity Maintenance in Federated Database Systems. In S. Conrad, M. Höding, S. Janssen, and G. Saake, editors, *Kurzfassungen des Workshops “Föderierte Datenbanken”*, Magdeburg, 22.04–23.04.96, pages 51–65. Bericht 96–01, Institut für Technische Informationssysteme, Universität Magdeburg, April 1996.