SCHEMA MAPPING AND OBJECT MATCHING: A STEP-BASED APPROACH TO ENGINEERING DATA MANAGEMENT IN OPEN INTEGRATION ENVIRONMENTS

Peter Katranuschkov[(1)], **Raimar J. Scherer**[(2)]

ABSTRACT: The interoperability of different engineering data models and the product data exchange between heterogeneous design tools are among the most fundamental research problems related to the development of open integrated environments for CAE/CIC. The successful solution of these problems is of primary importance for the acceptance and the wide implementation of product data technology on the basis of ISO 10303 (STEP).
This paper presents a flexible, non proprietary, STEP-based approach to design tool integration that has been developed in the EU project COMBI. It focuses on the interoperability methods and tools for product data management realised in the COMBI system: (1) a descriptive schema mapping language which complements the EXPRESS specifications of the implemented product data models, (2) an active object-matching mechanism which serves for providing the consistency of the modelling objects across disciplines, time and design domains, and (3) a communication management module which uses enabling World Wide Web CGI technology to allow for remote and concurrent information exchange via the Internet. Along with a critical discussion of the described integration concept, important future research topics targeting the development of a concurrent engineering environment, which will be pursued in the COMBI follow-up project ToCEE, are also briefly outlined.

KEYWORDS: Integrated Engineering Environments, Product modelling, Building product models, Interoperability, WWW-based communication, STEP

1. INTRODUCTION

In the last years, with the growing trend towards organised in work groups collaborative product development in building design and construction, the needs for establishing open integrated software environments on the basis of product data technology have gained broad recognition. The emerging standard STEP (ISO 10303-1) and several international pilot projects aiming at STEP-based integration, like the European projects ATLAS (Böhms & Storer 1994), COMBINE 2 (Augenbroe 1995), COMBI (Scherer 1996) etc., have become powerful promoters of this new information technology in the AEC domain. However, gathered experience has also shown that a detailed specification of a harmonised, *universal building product model* that can serve as a common database for all information needs in an integrated system *does not seem very realistic* because of a number of specific conditions, inherent to the scattered and multidisciplinary nature of the building industry.

(1) Sen. Research Assistant, Institute of Structural Mechanics and Applied Computer Science, Dresden University of Technology, Mommsenstr. 13, D-01062 Dresden, Germany
E-Mail: peter@bbbsr1.bau.tu-dresden.de, Fax: +49 (351) 463-3975, Phone: +49 (351) 463-2251

(2) Professor, Institute of Structural Mechanics and Applied Computer Science, Dresden University of Technology
E-Mail: scherer@bbbsr1.bau.tu-dresden.de, Fax: +49 (351) 463-3975, Phone: +49 (351) 463-2966

According to (Hannus et al. 1995), these „boundary conditions“ for product modelling and product data exchange are associated with the following issues:

- data exchange is primarily between different organisations using different applications and operating within different domains of AEC, and the scope of exchanged data varies between nations, companies, projects, project stages, application systems and current view of the actor;
- a large variety of heterogeneous software tools are being used, and the scopes of sending and receiving applications are not likely to be known a priori;
- AEC has fuzzy boundaries and covers a wide range of technologies, i.e. information needs in general are unpredictable.

Therefore, various more flexible approaches towards integration, addressing the problem of *interoperability*, i.e. sharing of AEC information distributed among heterogeneous models and agents, have been proposed recently. For example, in the COMBINE-2 project (Augenbroe 1995) the idea of a common building model (IDM) that integrates all actors' views has been implemented. Information exchange and interoperability are supported by a generic data exchange system (DES) which contains a data exchange kernel, off-line and on-line data interaction managers based on an SDAI binding for C++ (ISO 10303-22), and a knowledge-based communication control module using blackboard technology. Since the IDM is semantically richer than each application model, mapping is reduced to „meshing“ and „stripping“ the application models to/from the IDM. The ATLAS project (Böhms & Storer 1994) also relies strongly on the concept of semantic integration, which is achieved by means of a rich set of hierarchically organised reference models. Information exchange between the ATLAS applications is realised on the basis of these common reference models with the help of a set of data conversion tools. Semantically different representations are handled by means of a rule-based language with late LISP binding, which allows the rules to operate on top of SDAI data storage (Poyet et al. 1994). In the SEED system interoperability is achieved with the help of a meta-level agent communication language, which implements schema mapping through a shared neutral object model and appropriate language bindings. This neutral object model only exists conceptually for the applications which partially share its schema (Killicote et al. 1995). ACL/KIF supports an advanced knowledge-based agent-to-agent communication which includes translation of different data abstractions and logic processing of messages between different knowledge-based applications (Khedro, Genesereth & Teicholz 1994). However, both SEED and ACL/KIF propose their own integration strategies, which are not compliant to STEP.

In the remainder of this paper we present the interoperability methods and tools developed in the ESPRIT project COMBI, which provides a STEP-based framework for an open integration environment promoting the idea of „product model integration by communication“. A detailed survey of other proposed approaches can be found in (Liebich et al. 1995).

2. THE INTEGRATION FRAMEWORK OF COMBI

COMBI focuses on design tool integration, with special consideration to the heterogeneity of engineering applications -traditional, widely accepted numerical analysis programs, knowledge-based design tools, expert systems, conventional CAD etc. which can range from simple stand-alone programs to complex systems based on proprietary local integration environments. The conceptual schema of the COMBI framework is described in detail in (Ammermann et al. 1994). It is comprised of a set of product models, a set of external (remote) design tools and a central control kernel, responsible for the interaction and the coordination of all other components. The principal architecture of the COMBI system is outlined on fig. 1, concentrating on the structure of the developed integration prototype - PROMINENT (Katranuschkov & Scherer 1995).

Product Models

The set of object-oriented product models forms the knowledge base of the integration framework. It is hierarchically structured in the following three levels of data abstraction and data reduction:

- *application models*, which are most detailed and contain very specific application related objects,
- *partial aspect models*, which are more general and depict the common properties of objects that have to be shared within each design domain, and
- *a neutral domain-independent kernel model*, which - contrary to most other existing approaches - does not attempt to provide an integrating superset of the separate partial and application models, but serves rather for maintaining the persistence and preserving the integrity of the product model instances in each specific design project context, and for „bridging" the different semantic representations of the modelling objects across disciplines and design domains.

This intentionally sought minimal common kernel model is a unique feature of the COMBI framework, which allows greatest autonomy in the development, implementation and future extension of the individual partial and application models, putting the emphasis of the integration strategy on the use of intelligent interoperability mechanisms, rather than on the semantic homogeneity of the representation.

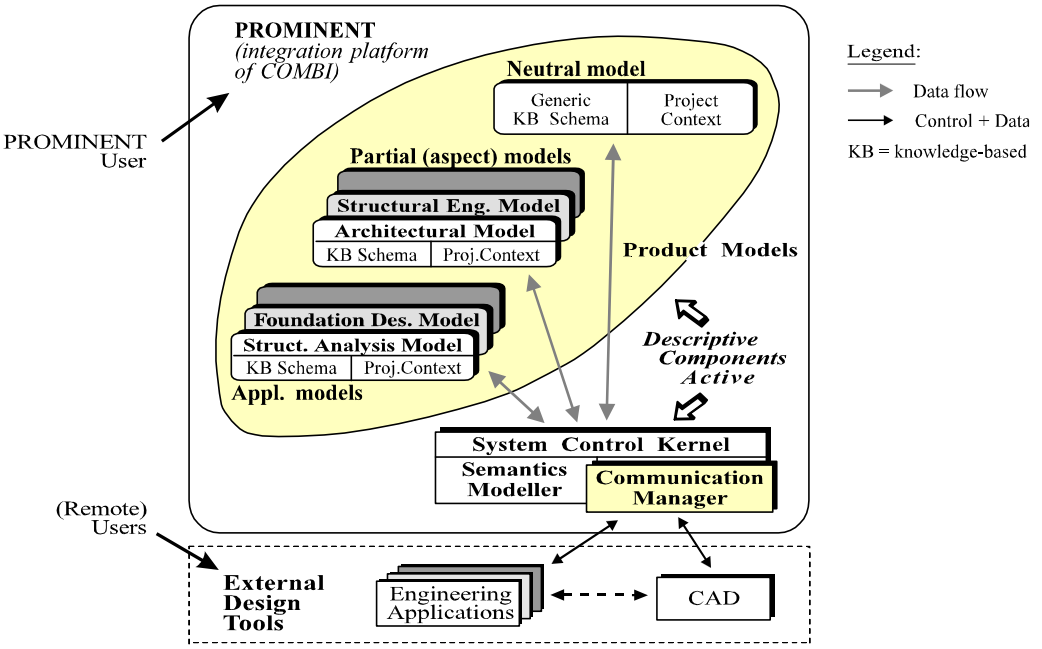


Fig. 1: System architecture

Each model in the framework is by itself comprised of 3 interrelated components: the *generic schema* of the model, which provides the typological representation of engineering knowledge for the considered modelling perspective, an *instantiated project context*, which contains the actual instances of the modelling objects and is capable of representing the dynamic changes of the data during the design process, and *active control methods*, which support the data transformations within the system kernel, the communication with the external applications, integrity checking etc.

Interoperability

To provide for a successful product data exchange between the remotely used engineering applications in the integration environment, COMBI addresses the following interoperability issues: (1) the transformations between the different object representations in the different partial and application models, (2) dynamic object evolution and object re-classification and (3) WWW-based communication management.

The separate models in the COMBI framework are specified formally in EXPRESS (ISO 10303-11). They use as far as possible the integrated resources defined in (ISO 10303-41, -42, -45). Interoperability is achieved with the help of a *schema mapping language*, which complements the EXPRESS specifications, and an active *object matching mechanism* installed in the communication management module of the system, which is responsible for the consistency of the product data. Fig. 2 shows schematically the full scope of the developed product models and the basic methods used for transforming the different semantic representations of the design information from one modelling space to another.

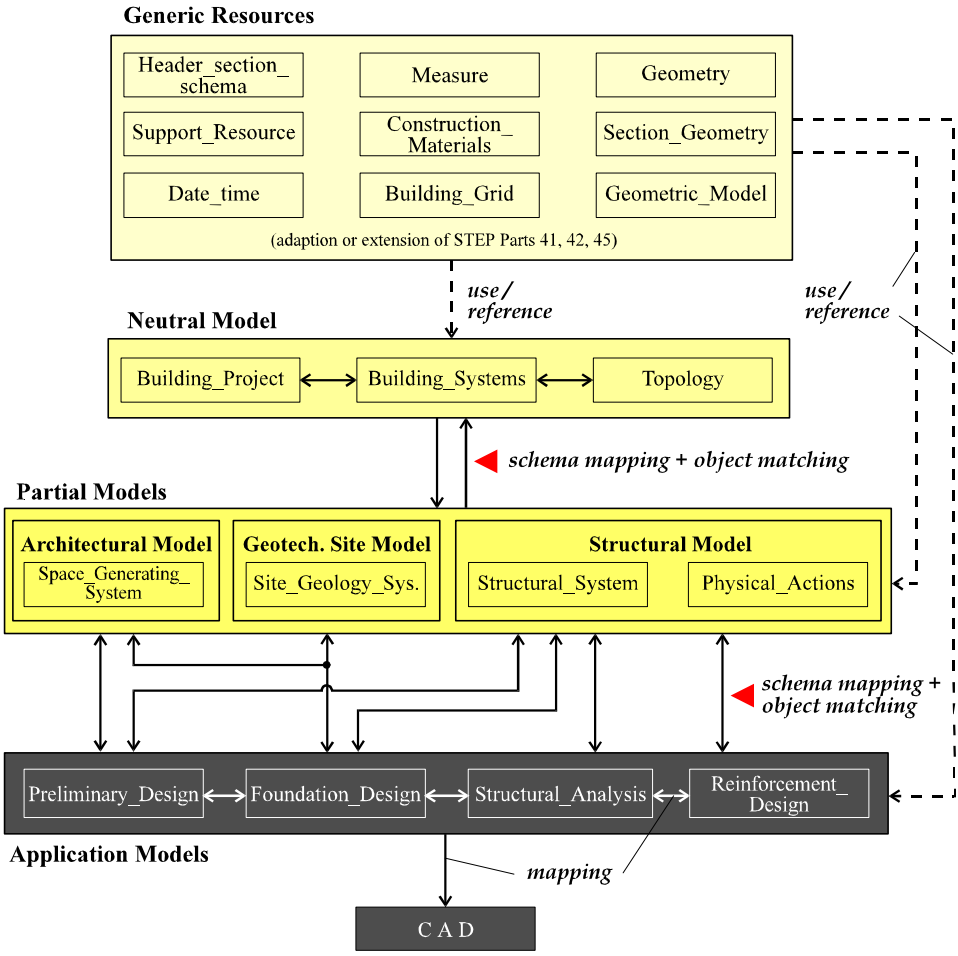


Fig. 2: Product model schemata and inter-schema relationships in the COMBI framework

The actual product data exchange with each application is realised intentionally only on the basis of STEP physical files (ISO 10303-21), since this requires no (or only minimal) changes in the internal data structures of the applications. The whole data exchange process, presented schematically on fig. 3, is identical for all integrated application tools and is controlled by the *COMBI Communication Manager* (CCM). Its basic components are described in detail in the following sections.

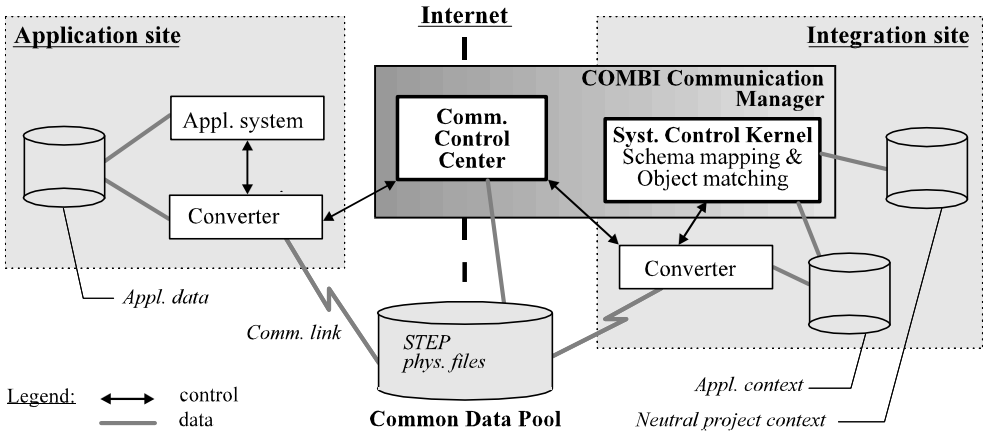


Fig. 3: Principal data exchange process with an application on the basis of STEP files

3. SCHEMA MAPPING

The correspondences between the data belonging to different model spaces of the COMBI framework are defined formally with the help of a specially developed for that purpose *schema mapping language* (CSML). Despite the nowadays existing wide range of mapping languages, for example EXPRESS-V (Hardwick 1994), EXPRESS-M (Bailey 1995), VML (Amor 1994) etc., we decided to create a new mapping language for COMBI for several reasons. First, all known languages have been developed in parallel or shortly before CSML and are still in an experimental phase, i.e. mapping continues to be a hot research topic. Second, only a few of these languages provide suitable programming language bindings for the COMBI integration platform (which is LISP-based). At last, it was found that all examined approaches target different interoperability areas, and while all they are certainly useful in a number of concrete situations, none would match sufficiently the requirements of the COMBI integration approach.

The CSML Language

In our understanding, mapping is a one-directional process, for example mapping data of application A_1 to application A_2 without being aware of the context (i.e. the locally stored data) of application A_2. Due to the different semantic richness of the separate partial models in the COMBI framework, an inverse mapping may not always be possible or would require a different specification (in fact, such situations may arise in any other integration environment).

A mapping operation always creates new data. These can be: a new model, new object classes in an existing model, complete or partial transformations of object instances belonging to one model into object instances of another model. In other words, given a source model with schema S' and instantiated context C' with instances $\{c_i'\}$, and a respective target model with schema S'' and context C'', a mapping operation can be defined as:

$$\textit{map-op:}\quad \underset{\textit{class mapping}}{S' \Rightarrow S''} \;\mid\; \underset{\textit{full instance mapping}}{C'(S') \Rightarrow C''(S'')} \;\mid\; \underset{\textit{partial instance mapping}}{\{c'_i\}_{i=1..m} \Rightarrow \{c''_k\}_{k=1..n}} \text{ with } \{c'_i\} \subseteq C',\ \{c''_k\} \subseteq C''$$

CSML meets all these requirements. It is based on a declarative modelling paradigm, which means that the development of a particular mapping specification can be concentrated on describing what transformations have to be done, rather than on prescribing how (in what order) to execute the necessary mapping operations. In that respect CSML is, together with VML (Amor 1994), remarkably different from all other examined approaches.

The implementation of CSML is based on LISP and thus uses a LISP-like notation style. It contains constructs that provide means for partial and complete mappings of the entities defined in an EXPRESS schema, covering the cardinality cases 1:1, 1:0, 0:1, 1:N and (with some restrictions) M:N, and allowing the following transformations of EXPRESS data types:

- simple data type ⇒ simple data type | null
- constr. data type ⇒ constr. data type | set of attributes | entity instance | null
- list of attributes ⇒ constr. data type | set of attributes | entity instance | null
- entity instance ⇒ entity instance | set of entity instances | attribute | set of attributes | null
- entity reference ⇒ entity reference | attribute | set of attributes | null.

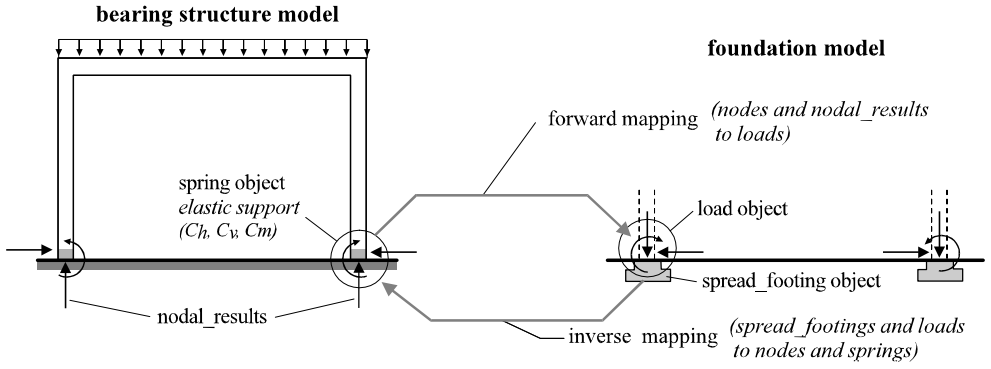
A mapping specification in CSML consists of a header and a body that can contain the sections CLASSES, DEPENDENT-CLASSES, GLOBALS, FUNCTIONS. The most important and in fact the only obligatory of these sections is CLASSES. It is used to describe the correspondences between the data in the source and the target model, while all other sections have complementary functions and are used only in certain specific situations. CLASSES can itself contain any number of the following mapping definitions: CREATE-CLASS - which describes new classes (and resp. instances) to be created in the target, COPY-CLASS - which represents exact equality between source and target instances, MAP-CLASS - which expresses any kinds of 1:1 or 1:N transformations, and MAP-GROUP - which is a compound specification that may contain any number of the other 3 types and is used to describe M:N mappings. The main and most frequently used mapping construct in CSML is MAP-CLASS. Its syntax (in BNF) is:

map_class ::= (MAP-CLASS *class* FROM *class-list*
[[{ ATTRIBUTES { *attr-mapping-op*}$^+$ }$^+$ |
{ CONDITIONS { *cond-op* }$^+$ }* |
{ CHECK *check-specs* }*]])

The individual clauses of MAP-CLASS have the following meaning. The FROM clause declares the objects in the source schema that have to be mapped. The CONDITIONS clause describes *when* (i.e. under which conditions) the MAP-CLASS definition can be applied. It may contain one or more predicate functions which must all be true to allow the mapping to take place. The ATTRIBUTES clause defines *what* are the equivalences between the attributes of the source object(s) and those of the target. These equivalence specifications can range from direct equality and standard data type transformations to complex relations like chains of references, built-in, in-line and user-defined functions. At last, the CHECK clause describes how to treat the newly created target objects after the mapping operation (in fact, this clause contains instructions for the object matching procedure; it does not influence directly the mapping and is included in MAP-CLASS for compactness of the definition).

A simple practical example derived from the prototype implementation of COMBI can serve to illustrate the use of CSML (see fig.4). It depicts a portion of the mapping specifications needed to transform typical structural objects like nodes and nodal_results into objects for foundation design, which is necessary to allow the design cycle *preliminary structural design* ⇔ *foundation design* to be performed properly, i.e. to transform reactions into loads and, inversely, computed foundation stiffnesses into spring coefficients for frames and shear walls. This example gives a fairly good

notion of the often very different representations of the building objects an integrated system has to deal with. Other more complex examples can be found in (Katranuschkov & Scherer 1995).




```
;;; FORWARD MAPPING: preliminary structural design -> foundation design
(MAP-CLASS load FROM node nodal_results                                          ; note (1)
   CONDITIONS (RULE !check_foundation_pt node.SELF $elevation)                   ; note (2)
   ATTRIBUTES
     (MAKE id FROM node.SELF)                                                    ; note (3)
     (MAKE loadcase_ref FROM nodal_results.load_id)                              ; note (4)
     (MAKE load_magnitude FROM (MAPLIST (LAMBDA (X) (* X -1.0)) nodal_results.reactions) )   ; note (5)
     (MAKE appl_pt FROM (LIST (REF node.coord FOR x) (REF node.coord FOR y) (REF node.coord FOR z) ) )   ; note (6)
)

;;; INVERSE MAPPING: foundation design -> preliminary structural design
(MAP-GROUP spring node WITH                                                      ; note (7)
   (MAP-CLASS node FROM spread_footing load ... )                                ; note (8)
   (MAP-CLASS spring FROM load                                                   ; note (9)
       ATTRIBUTES
         (MAKE id FROM SELF)                                                     ; note (10)
         (MAKE node_list FROM (LIST (NEWOBJECT node.SELF) ) )                    ; note (11)
         (MAKE hinge_list FROM spring_coeffs) ... )                              ; note (12)
)
```

Notes: (1) - Starts the description of the inter-schema relationship b/n the classes *load* and {*node*, *nodal_results*};
(2) - Applies a user-defined predicate function to the *node* entity and the global variable *$elevation* (defined elsewhere) to determine if the mapping of a particular *node* instance can proceed. In this case, requires that a structural node is a support and lies on a flat foundation with a preset elevation defined in the global coord. system of the building;
(3) - Assigns the implicit entity identifier of the source (*node*) to the explicit attribute *id* of the target (*load*);
(4) - Performs a simple attribute assignment;
(5) - Uses a built-in and an in-line function to transform a list of attribute values in the source instances to a respective list of values in the corresponding target instances;
(6) - Constructs a list from three chained references: *node ® coord ® x (y , z)*
(7) - Starts the mapping specification for a group of classes and establishes a local scope for all their instances in the subsequent mapping declarations within the MAP-GROUP block;
(8) - Describes the mapping for the class *node* (the concrete equivalences for the attributes are omitted for brevity);
(9) - Starts the description of the mapping for the class *spring*;
(10) - Same as for note (3), but because there is only one source class (*load*), it must not be stated explicitly as in (3);
(11) - Defines *explicitly* a list type attribute from the associated *node* object(s) to each newly created instance of *spring*;
(12) - Performs a simple attribute assignment, where both source and target are lists (no type conversion necessary).

Fig. 4: Mapping specification example

Comparison to Database Operations

CSML manifests an equivalent expressive power to the basic operands required in a relational database system (Ullman 1982) and thus can be easily used to perform all typical data transformations that can be applied to database schemata. Fig. 5 shows how CSML emulates the operands projection, selection, union, set difference, cartesian product and intersection. It can also be used for comparison to the VML language, since it deliberately follows the examples given in (Amor 1994).

Projection *Given:* a source entity with attributes $S_1..S_m$ to produce a target entity with attributes $T_1..T_n$ *Mapping:* (MAP-CLASS T from S ATTRIBUTES ... (MAKE T_j FROM S_i) ...)	**Selection** *Given:* a source entity S with attributes $S_1..S_m$ and a set of Boolean formulae F which work on S to produce T *Mapping:* (MAP-CLASS T FROM S CONDITIONS (RULE !F ... S_i ...) ... ATTRIBUTES (MAKE T_1 FROM S_1) ...)
Union *Given:* R ($R_1..R_m$) and S ($S_1..S_m$) $\Rightarrow$ T *Mapping:* (MAP-CLASS T FROM R ATTRIBUTES (MAKE T_1 FROM R_1) ...) (MAP-CLASS T FROM S ATTRIBUTES (MAKE T_1 FROM S_1) ...)	**Cartesian Product** *Given:* R ($R_1..R_m$) and S ($S_1..S_m$) $\Rightarrow$ T *Mapping:* (MAP-CLASS T FROM R S ATTRIBUTES (MAKE T_1 FROM R_1) ... (MAKE T_m FROM R_m) (MAKE T_{m+1} FROM S_1) ... (MAKE T_{m+n} FROM S_n))
Set Difference *Given:* R and S, which have key attributes $R_1..R_k$ and $S_1..S_k$ and attributes $R_1..R_m$, to produce a target entity T *Mapping:* (MAP-CLASS T FROM R S CONDITIONS (RULE lambda (R_1 ... R_k S_1 ... S_k) (and (not (equal R_1 S_1)) ... (not (equal R_k S_k)))) ATTRIBUTES (MAKE T_1 FROM R.R_1) ...)	**Intersection** *Given:* R and S, which have key attributes $R_1..R_k$ and $S_1..S_k$ and attributes $R_1..R_m$, to produce a target entity T *Mapping:* (MAP-CLASS T FROM R S CONDITIONS (RULE lambda (R_1 .. R_k S_1 .. S_k) (and (equal R_1 S_1) ... (equal R_k S_k))) ATTRIBUTES (MAKE T_1 FROM R.R_1) ...)

Fig. 5: Specification of the basic operands of relational databases with CSML

However, though powerful enough to support complete and partial model transformations - both horizontally, i.e. directly between applications, and vertically - to/from the persistent neutral model, CSML alone is not sufficient to capture all dependencies and ensure the consistency of the product data, because the static mapping specifications cannot take into account all dynamic changes to the modelling objects and their evolution during the design process. Therefore, the implementation of CSML requires an active data management component that can track correctly the modifications of the data structures resulting from the mapping operations.

4. OBJECT MATCHING

We understand *object matching* as a process of associating the entities of a structured data set - typically in the form of object instances and their relationships - to multiple classification hierarchies reflecting different modelling aspects for these data. In other words, object matching can be interpreted as a *mechanism for dynamic object classification*

The necessity for „matching“ object instances against the class specifications in the generic schemata of a product model arises when the mapping transformations are applied for *updating already existing product information*. To focus the discussion, let us consider again the example given in fig. 4. At the first iteration step of the presented design cycle, i.e. after the preliminary structural design tool has completed the initial configuration of the structural system of a building and the relevant data are passed to the foundation design tool to determine the appropriate foundation system, no consistency problems will arise, since the foundation design objects will be created for the first time and added as new items to the product data base of the project. However,

when returning to analyse the bearing structure - this time taking into account its interaction with the foundation - the situation will be different because the newly created foundation objects can affect the structural system in various ways. First, the actually computed stiffnesses of the designed foundations will as a rule cause changes to the presumed support conditions of the pre-designed frames and shear walls and hence will lead also to changes in their overall stiffness. This may, however, lead further to partial or full re-design of the structural system if its components do no longer satisfy the defined mechanical, serviceability and constructability requirements. Thus, it may become necessary to re-dimension beams and columns, which would result, in turn, in new loads to the foundations and can also influence architectural or HVAC design. It is also possible that some structural assemblies must be entirely reconfigured, in which case they might no longer match the determined foundation layout. At last, because of changes in their properties, certain elements might even need to be re-classified, e.g. spread footings may „migrate" to strip footings, or the whole shallow foundation system may need to be substituted by pile foundations.

This simple interaction example shows that even small, at first glance „harmless" changes in one partial model can cause cascadingly propagating changes in one or more other models leading to serious consistency problems. In the integration environment of COMBI such problems are handled by the specially developed object matching mechanism residing in the system control kernel. It is activated automatically whenever an application tries to update its corresponding information context, i.e. create a new version of its instantiated model, or when the results of a mapping operation have to be verified against the constraints defined in all other relevant models.

The developed method follows the *frame-based modelling paradigm* and is strongly related to the hierarchical architecture of the COMBI framework. Its key idea is to use the persistent neutral model of the framework as a reference structure for all other modelling representations by means of associating each object instance in a given context to instances of *primitive object classes*[(1)] defined in the neutral model. These primitive object classes cover only basic, domain-independent modelling aspects like the topological representation of the building elements, their implicitly maintained unique identification, generic object-to-object relationships like *is-part-of*, *has-part*, *depends-on*, *connected-to* etc., while domain-specific aspects, including among others the explicit shape representation of all tangible building objects, are dealt with in the respective partial and/or application models. Thus, it is possible to establish a „bridge" across the separate modelling spaces which can still be managed locally and independent of each other, as far as their inter-relations are not concerned.

Though labelled „primitive", the generic classes defined in the neutral model are not simple. Besides attributes and methods determining their behaviour, they do contain also specific *rules* which, when triggered, allow to recognise if an object instance can be associated to a particular class. In order to let these „matching" rules work on arbitrary objects whose structure is not known a priori, at the same time avoiding the classical multiple inheritance technique which would lead to an enormous complexity of the data structures, we use dynamically established „when-needed" *delegation links*[(2)], as proposed by (Zucker & Demaid 1992). In this way, a unified approach to all partial models is applied, and the matching rules are concentrated only in the neutral model which greatly simplifies the complicated matching process.

As a concrete example of the technique outlined above, let us examine again the design interaction case presented in fig. 4, now focusing on the process of updating the existing structural system after

(1) We use here the term „primitive" to emphasize that the neutral model is not a superset of all other integrated aspect models, but contains only minimal, domain-independent information which has the primary task to „glue" the different modelling spaces together.

(2) Delegation is a regime of object-to-object communication which allows one object to have temporary enhanced capabilities by acting as an extension of another object.

mapping the objects contained in the foundation model onto the structural model. In a first step, the mapping operation creates temporary objects (with temporary, implicitly assigned identifiers), which represent only that portion of the structural system which ensures the contact to the foundation. The task of the matching procedure is then to verify the correctness of these objects in respect to the overall building model and to merge them into the instantiated structural model context, thus updating its state. In order to do this, all temporary objects are first associated to respective objects in the neutral model on the basis of their topological properties, as well as by using predefined in the CHECK clauses of the mapping specifications „monitoring values", which can be object attributes or calculated at run-time Boolean expressions. Thus, load objects, spread footings, structural nodes and springs become associated to vertex points, strip footings are associated to line edges etc. After that, the corresponding objects in the existing data structures of the product model can be easily identified and checked, and their state can be correspondingly updated - eventually creating new instances or re-classifying existing ones when appropriate. For example, if a spring element has not been defined previously because the support nodes of the columns have been assumed „fixed", it will be created and added to the project data base. On the opposite, if the support conditions become very stiff, e.g. by changing the design from shallow to deep foundations, it may be necessary to remove existing spring elements and at the same time re-classify foundation elements from spread footings to piles. However, it should be noted that since the developed matching mechanism does not yet support conflict management, all such changes are at first merely recorded and proposed to the user(s), but are executed only after explicit permission.

5. COMMUNICATION CONTROL

In order to provide support for real cooperative design work, an integration environment has to include also - besides efficient interoperability and data exchange mechanisms - a communication management layer which is capable of *controlling the purpose of actor-to-actor communication*, i.e. „supervise who can do what and for what purpose at each stage of the design process". Without such control, information exchange can easily become chaotic unless forcedly squeezed into a very restrictive schema.

In COMBI, this task is performed by the developed *Communication Control Center* (CCC), which monitors all transactions to/from a common data pool via the Internet, using enabling World Wide Web CGI technology on the basis of the capabilities of the standard *httpd* demon (Luotonen & Berners-Lee 1994). While the complete details of the communication control mechanism realised in COMBI will go beyond the scope of this paper, we shall provide below a broad outline of its most important features to complete the discussion of the presented integration approach.

CCC is a relatively simple tool, planned to be considerably enhanced in the COMBI follow-up project ToCEE (ToCEE 1995). It is implemented entirely on the WWW which allows concurrent asynchronous access to the common data pool of a project by all project participants. CCC supports different process related operations like: storing / retrieving / viewing STEP physical files (by using one of three alternative protocols - *http*, *ftp* or *mail*), posting messages to other participants in pre-specified formats, requesting summary reports of the state of the project and the possible further design steps at each stage of the design process. In order to provide control over the transactions between the distributed on the global network design agents, CCC uses an additional information layer containing the following components: (1) *a project description schema*, which identifies the involved participants, their network addresses, roles, access rights, used tools etc., (2) *a process description schema* implemented in the form of a Petri-Net, which allows to determine the meaningful transactions and possible further design tasks by means of explicitly stored control items for each implemented model (extracted from its current instantiated context), and (3) *a facts base* realised in the form of a blackboard, which stores all communication messages accompanying the product data exchange and helps reasoning about possible design actions.

6. CONCLUSION. CURRENT STATE AND FUTURE WORK

The described interoperability methods developed in the COMBI project are a promising beginning on the way towards STEP-based concurrent engineering systems. As proof of concept, they have been tested in a prototype environment including a knowledge-based system for preliminary structural design, a proprietary integrated system for structural analysis, dimensioning and CAD, a decision support system for soil characterisation and foundation design and a conventional, general-purpose CAD system. The obtained results have verified the potential power of the suggested approach.

To give an impression of the current state of the system, on fig. 6 below is presented a snapshot of the developed prototype integration platform PROMINENT, showing a run-time reflection of the example sketched on fig. 4. It illustrates the WWW-based communication with the foundation designer, the graphical user interface implemented as support environment for CSML (exposed on the main desktop of the integration platform), and the on-line interaction with the used general-purpose CAD system (presenting, in particular, the overlapping of the structural and the foundation model for visual evaluation of the obtained solution - not possible in this form in any of the two involved design tools).

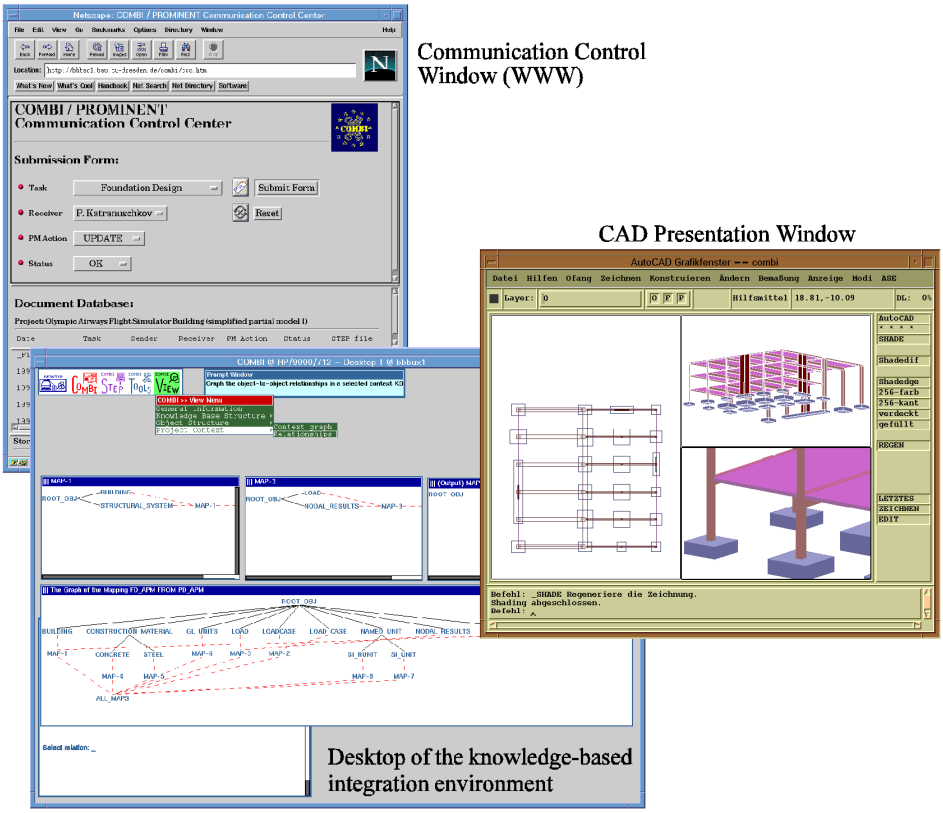


Fig. 6: Snapshot of the COMBI integration platform

The presented integration approach is a promising contribution on the way towards computer-aided collaborative product development. However, it has to be enhanced considerably in order to meet the requirements of a real concurrent engineering environment. Future development efforts have to be directed towards extending communication control to a full-scale information logistics service, enhancing the interoperability of the integrated system to include methods for conflict resolution,

estimation of the impact of early design solutions on the whole life-cycle of a building, document management, auditing, interface to computer-interpretable building codes and regulations etc. These and other related research issues will be investigated in the follow-up ToCEE project (ToCEE 1995), launched in January, 1996. The funding of both the COMBI and the ToCEE projects by the European Commission, which enabled this research work, is herewith gratefully acknowledged.

REFERENCES

Ammermann E., Junge R., Katranuschkov P., Liebich Th. & Scherer R. J. (1994): *Concept of an Object-Oriented Product Model for Building Design*, Rep. 1-2/94, Inst. of Structural Mechanics and Applied Computer Science in Civil Eng., Dresden Univ. of Technology, Germany.

Amor R. (1994): *A Mapping Language for Views*, Dept. Rep., Computer Science Dept., Univ. of Auckland, New Zealand.

Augenbroe G. (1995): *COMBINE 2 Final Report*, EU / CEC Joule Programme, Project JOU2-CT92-0196, TU Delft.

Bailey I. (1995): *EXPRESS-M Reference Manual*, ISO TC184/SC4/WG5 N243.

Böhms M. & Storer G. (1994): *ATLAS - Architecture, methodology and Tools for computer-integrated LArge Scale engineering*, Proc. JSPE-IFIP WG 5.3 Workshop, DIISM'93, Tokyo.

Hannus M., Karstila K. & Tarandi V. (1995): *Requirements on Standardised Building Product Data Models*, in: Scherer R. J. (ed.), Product and Process Modelling in the Building Industry, Proc. First ECPPM'94, Dresden, Balkema Publ., Rotterdam.

Hardwick M. (1994): *Towards Integrated Product Databases Using Views*, Rep. 94003, Design and Manufacturing Inst., Rensselaer Polytechnical Institute.

ISO 10303-1, -11, -21, -41 to -45 (1992-94): *STEP Parts 1, 11, 21, 22, 41-45*, ISO TC184/SC4.

Katranuschkov P. & Scherer R.J. (1995): *User Manual of the Product Modelling Integration Environment for Building Design - PROMINENT v.1.1*, Rep. 2/95, Inst. of Structural Mechanics and Applied Computer Science in Civil Eng., Dresden Univ. of Technology, Germany.

Khedro T., Genesereth M.R., Teicholz P.M. (1994): *Concurrent Engineering Through Interoperable Software Agents*, Proc. First Conference on Concurrent Engineering: Research and Applications, Pittsburgh, PA.

Killicote H., Garrett J.H., Choi B. & Reed K.A. (1995): *A Distributed Architecture for Standards Processing*, in: Pahl P.J. & Werner H. (eds.), Computing in Civil and Building Engineering, Balkema Publ., Rotterdam.

Liebich Th., Amor R. & Verhoef M. (1995): *A Survey of Mapping Methods Available Within the Product Modelling Arena*, working paper, EXPRESS User Group, ISO TC184/SC4.

Luotonen A., Berners-Lee T. (1994): *CERN httpd Guide*, CERN Rep.

Poyet P., Grivart E., Brisson E., Besse G., Irvine M., Greening R. & Böhms M. (1994): *ATLAS: Implementation of Knowledge Base Extensions*, ATLAS Deliverable D301a, EU / CEC ESPRIT III Project No. 7280, Brussels.

Scherer R. J. (1996): *COMBI Final Report*, EU / CEC ESPRIT III Project No. 6609, TU Dresden.

ToCEE (1995): *Project Programme*, EU /CEC ESPRIT IV Project No. 20587, Brussels.

Ullman J. D. (1982): *Principles of Database Systems*, Second ed., Computer Science Press.

Zucker J. & Demaid A. (1992): *Modelling Heterogeneous Engineering Knowledge as Transactions Between Delegating Objects*, in: Gero J. (ed.), Artificial Intelligence in Design '92, Kluwer Academic Publ., Rotterdam.