AN ONTOLOGY FRAMEWORK TO ACCESS IFC MODEL DATA

SUBMITTED: May 2003
REVISED: August 2003
PUBLISHED: October 2003 at http://www.itcon.org/2003/29/
EDITORS: Robert Amor and Ricardo Jardim-Gonçalves

Peter Katranuschkov, Senior Research Assistant, Dr.-Ing
Institute of Applied Computer Science in Civil Engineering, University of Technology Dresden, Germany
email: Peter.Katranuschkov@cib.bau.tu-dresden.de

Alexander Gehre, Research Assistant
Institute of Applied Computer Science in Civil Engineering, University of Technology Dresden, Germany
email: Alexander.Gehre@cib.bau.tu-dresden.de

Raimar J. Scherer, Professor, Dr.-Ing
Institute of Applied Computer Science in Civil Engineering, University of Technology Dresden, Germany
email: Raimar.J.Scherer@cib.bau.tu-dresden.de

SUMMARY: *In the last decade several research and development projects have shown that product data technology (PDT) can successfully and beneficially replace the traditional document-centred approach to project realisation. Today, with the development of the IFC project model by the IAI, the product modelling paradigm is being rapidly introduced in commercial software as well. However, actual PDT application in the AEC domain is still limited to CAD data exchange and some basic project-centred data management facilities. There is a great need of human-centred product model services supporting the engineer with additional knowledge about the models, providing customisable user-friendly capabilities for management and modification of the data, and enabling structured access to the information on the project(s) he is working on. These issues have motivated the development of an ontology framework that can serve as advanced user gateway to product model data. The suggested approach draws upon recent ICT achievements, especially regarding IFC environments.*
In this paper we present the rationale, the principal design, and the technical structure of the ontology framework based on the XML schema specification. Further on, we explain its software realisation, provide examples of its current application and outline possibilities for its further development and envisaged broader usage. Reported is research work performed in conjunction with the EU ISTforCE project (IST-1999-11508).

KEYWORDS: *Ontology, Model-Based eWork, Engineering Data Browser, XML Schema, IFC.*

1. INTRODUCTION

Existing holistic approaches to information and knowledge management of organisations, such as the BPR (business process re-engineering) and the ERP (enterprise resource planning) models have shown the importance of the integrated treatment of the data definitions underlying the software applications used in an organisation. Today there exists a broad understanding that standardised reference of all data definitions to a common set of harmonised schemas is a vital prerequisite for the achievement of true *model-based* technical work that can bring about added value to various aspects of engineering product development (Björk, 1995, Wix, 1996, Owen, 1997). Among these aspects are the coordination and cooperation of team work, efficient integration of application tools, improved data exchange and sharing, common model repositories, better navigation in the multi-dimensional project data space etc. Such expectations, originating from the higher semantic richness of PDT-based environments compared to traditional document-centred systems, are rapidly gaining shape with the progress of the IFC project model (IAI 2000). However, the actual state of the art of PDT application in the AEC domain is still limited to CAD data exchange and, in a few cases, some basic model management facilities (Liebich & Wix, 2002). Current collaboration environments are mostly document oriented, but even when PDT is utilised the approach is project-centred, providing for some advanced teamwork features but offering little help with respect to the personal needs of the individual users. What is missing is the provision of an *efficient user gateway to the technical model data structures*. If an IT system would support standardised modelling and data management operations that can be carried out on end-user level, and with the help of intuitive user

interfaces, the acceptance of product data technology in building construction practice and the deployment of high quality PDT-based applications can greatly be improved.

However, recent practice has shown that establishing comprehensive, standardised product data models proves to be a long and complicated process. According to (Turk, 1998) the problems experienced in the development of standardised, large-scale product data models are due to (1) the difficulties in which the conceptual product models are being formalized by the experts in the field, (2) the difficulties to agree on a common representation, and (3) the incompleteness of the models, particularly in the areas of unconventional and creative design. Such problems can typically be noticed by all models that are not directly related to a specific application. They are largely valid for the IFC project model as well.

To help tackle these issues and provide a common medium for communication with a shared, reusable product data base to various distributed platforms, users, applications and network protocols, in the EU project ISTforCE (IST-1999-11508) we have developed methods and specifications for an *engineering-friendly ontology framework* that can bridge the gaps between users, data models and software applications.

In the 90s, there has been considerable research in the area of artificial intelligence and other related technical domains to explore the use of formal ontologies as a means of defining content-specific agreements for the sharing and reuse of knowledge among heterogeneous software components and the respective end-users of these components. However, the achieved results are still only of limited practical applicability.

We take a more pragmatic approach to the development of such ontologies. Particularly, for the domain of civil engineering, we are designing an engineering ontology which has the primary purpose to support end-users in their practical work with project model data in a natural way. The target of the undertaken work has not been the specification of yet another conceptual model in building construction. Rather, we build upon already established terminology and classification work in the civil engineering domain and the upcoming IFC2x project model to show and verify how an ontology can be usefully applied as a gateway for end users and applications to the shared data and knowledge in a distributed PDT-based environment. The suggested framework aims at providing a simple and light-weight basis enabling end users and (non IFC) applications to access, retrieve and reason upon product data. It focuses on the understandability and the flexible user interface for querying/retrieving/examining the product data e.g. for project management, controlling or coordination purposes, and not on the data exchange between CAD applications, neither on comprehensive support for PDT-based environments. In this respect it is remarkably different from any product data centred approaches and thus complements and does not compete with IFC, ISO STEP, IIDEAS, EPISTLE or any other large modelling effort. Furthermore, the ontology specification is fully *web-enabled*, based on a widely accepted technology (XML) and capable for use as an interactive exchange mechanism for structured and typed information in a distributed environment.

The rationale, the principal design, the technical structure and the software realisation of the developed framework, as well as examples of its application and further envisaged usage are outlined in the following sections.

2. RATIONALE

2.1 Why an Ontology

The taxonomies and other representation forms created by product model developers provide a comprehensive set of model entities needed by IT systems. However, in the modelling process inevitably a number of concessions are made with regard to the way engineers use to think and work. Hence, practitioners can only use the models *indirectly*, with the help of dedicated specialised applications, and not *directly*, by interactively working with the model data itself. This contradiction is based on the different way humans and software actors work.

Normally, engineers think in complex abstract patterns and work goal-oriented. Their thinking patterns are adapted to the actual problem context, and they use different modelling abstractions to adapt the design context to an appropriate representation for the targeted design solutions. Due to that, each real world object typically has many different representations in the engineering way of thinking and problem solving. This is a natural method to manage the information needed to solve complex engineering tasks.

In contrast, a computer program "thinks" in strictly formalised patterns. It is not capable to abstract a problem by itself. Computer programs are specialised in doing automated tasks, solving problems by pre-defined methods, and tackling large amounts of strictly and consistently specified data. Typically, they operate on a single internal model, containing all needed information in one global context.

The engineer has to build the bridge between these two paradigms. If he wants to use a product model in a direct manner, he has to possess specific computer science expertise, and even then he will be handicapped in his natural way of working.

Hence, a primary objective of an advanced IT system should be to "think like an engineer". It should be able to communicate with the engineer in his language, and do the mapping to its internal data representation by itself. Such behaviour can be established by means of an ontology-based model access framework that can provide:

- facilities for high-level building information representation enabling the transition from information (model data) to knowledge (engineering concepts using "interpreted" model data);
- capabilities to capture the engineering semantics familiar to the end users;
- adequate support for multiple views;
- specifications of high-level engineering functions offering advanced methods of manipulating the model data to end users and applications;
- mechanisms for accessing product data repositories through natural-language-like queries.

2.2 Ontologies as a Specification Mechanism

Each piece of formally represented knowledge is based on a *conceptualisation*, i.e. the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them (Genesereth & Nilsson, 1987). A conceptualisation is an abstract, simplified view of the world that we wish to represent for some purpose. Every data model, knowledge base, or software system is committed to some conceptualisation, explicitly or implicitly.

An *ontology* can be understood as an explicit specification of such a conceptualisation (Gruber, 1993). The term is borrowed from philosophy, where an ontology is seen as a systematic account of existence. In the terms of an IT system, this means that what "exists" is that which can be represented.

In the context of civil engineering, an ontology will thereby be described by defining a set of representational terms in the target domain. These definitions must adequately associate the names of entities in the universe of discourse (bearing structure, wall, column, beam, joint, support, spring …) with human-readable text describing what these names mean, and formal axioms that constrain the interpretation and the use of the introduced terms. Thus, different from a product data model, an ontology is comprised of statements of a logical theory (and not only data specifications) about the targeted abstraction of the real world.

A common ontology framework can be used to describe *ontological commitments* for a set of actors (humans, software applications, AI agents) so that they can communicate about the domain of discourse without necessarily using a globally shared theory. An actor commits to the ontology if its *observable* actions are consistent with the definitions in the ontology. This idea is based on the *Knowledge-Level* perspective introduced by Newell (1982) in the domain of distributed artificial intelligence. The knowledge level is a level of description of the knowledge of an agent that is independent of the symbol-level representation used internally by this agent.

In practical terms, this means to define a common vocabulary with which queries and assertions can be exchanged. However, whilst in the classical AI approach a decentralised distributed agent environment with intelligent software agents is generally presumed, for the ISTforCE platform (and many similar IT systems) we proposed a more pragmatic asymmetric approach aligned with the asymmetry of client/server interaction, with clients and end-users "asking" the questions and servers "telling" the answers. This approach greatly simplifies the implementation methods and imposes weaker requirements to engineering applications. In fact, in this case the ontological commitments are only *agreements* to use a shared vocabulary in a coherent and consistent manner. The actors sharing the vocabulary do not need to fully share the knowledge base; each may know things the others do not, and it is not required to answer all queries that can be formulated in the shared vocabulary.

To be more explicit, let us consider what may happen when information about a wall is requested, for example to check its code compliance for fire resistance. In a pure IFC-based environment this information would be stored in a shared project repository as an *IfcWall* object and a number of related resource objects, such as *IfcMaterial*, *IfcPropertySet*, *IfcShapeRepresentation*, *IfcLocalPlacement* etc. An appropriate query can easily be defined, but – due to the different structuring and partially different semantics of a "wall" as understood by different users

and applications –, it is not so obvious as to what should be the response, except if all model schemas are fully harmonised and all users and applications "know" and use them consistently – an unlikely situation. In contrast, the ontology approach would mean to define the concept of a wall *unequivocally*, in a manner that is independent of the specific data definitions of the applications that are using it but provides sufficient means to execute operations on wall objects and interpret the results of these operations correctly.

A commitment to such a high-level ontology will guarantee consistency, but it cannot guarantee completeness of the discourse. It is expected that dedicated tools can and will take care of the latter. This simplifies both the specification and the implementation of the ontology, and enables users and software systems to easily use its concepts; however, without full certainty for the success of each possible query. Therefore, it is important to define the level at which the ontology is specified. This representation level is chosen on the basis of two criteria: (1) *lean content* (interactions should not be overburdened with too many concepts), and (2) *sufficiency* (the representation should be sufficient for unambiguous error-free discourse).

2.3 Principal Usage Scenarios

Use of the ontology framework within an advanced Model Access Service involves *two kinds of actors* and respectively defined use cases (Gehre & Katranuschkov, 2000):

1. *End user (engineer),* using the ontology directly to query/retrieve/exchange/modify information contained in a product data repository. These actions require an Engineering User Interface embedded into a common Web Browser.
2. *Advanced engineering applications* used by the engineer for the solution of specific design tasks. Such applications should commit to the ontology specification and be capable to query the Model Access Service and interpret its responses correctly, but they are not necessarily required to work with the full technical product model specification.

The activities of these principal actors have to be supported by the following *software actors*, as part of an ontology implementation environment:

- *Web Browser,* comprising the front-end to the ontology-based model access services;
- *Product Data Provider,* that should retrieve and store the actual product data from a product data repository by using appropriate interfaces, such as the ISO STEP physical file (SPF) and database access (SDAI) specifications, ifcXML (Liebich, 2001) etc;
- *Ontology Repository* to store the ontology specifications;
- *Ontology Interpreter* that should be capable to map end user requests and responses provided in terms of the defined ontology concepts to real product data objects;
- *Explanation Component* that should be responsible to handle the communication between the Web Browser and the Ontology Interpreter and to generate user-related presentation output.

Numerous variations of ontology usage as a gateway to product model data can be envisaged on the basis of the principal use cases outlined above. How this can work in practice is explained in section 5 further below.

3. RELATED WORK

The developed ontology framework presented in this paper is original work which does not adapt nor extend any earlier efforts. Specific product model developments in ISO STEP, IAI/IFC or elsewhere are intentionally not used as basis for the design of the framework, to keep its generality as broad as possible. However, closer relationship to the IFC project model has been particularly considered for practical purposes. High-level IFC kernel entities such as *IfcObject*, *IfcRelationship*, *IfcProduct*, *IfcBuilding*, *IfcBuildingStorey*, *IfcBuildingElement* etc. were "borrowed" as patterns for the development of core ontology concepts whereby especially useful was their ifcXML representation (Liebich, 2001). Furthermore, whilst not directly applicable, a large number of theoretical and practical efforts in the domains of conceptual modelling, ontology development and software technology have provided useful background for this research.

In the area of conceptual modelling, several prior efforts have delivered valuable ideas for the ontology design and specification (Björk, 1992, Ekholm, 1996, Angus & Dziulka, 1998, etc.). Here especially the idea of a

minimal kernel model (de Vries, 1991, Wix, 1996) that can be used as basis for an extensible set of more specific concepts is of interest.

Useful material have provided also known classification systems like *OmniClass* and *Uniclass*, as well as various studies of building structure configurations, e.g. by Schodek (1992) and Schueller (1995).

From theoretical point of view many valuable hints could be obtained from recent research in the area of ontology construction. There exist several publications describing the fundamental principles (Hobbs, 1995, Eschenbach & Heydrich, 1995) and the development process of ontologies for specific purposes (Benjamin et al., 1998). Useful studies are available for various application domains, such as physical systems (Borst et al., 1997), agent-based knowledge sharing systems (Gruber, 1993), remote database access (Ouksle & Ahmed, 1999), knowledge management (Lima et al., 2002). The latter work carried out in the frames of the EU e-COGNOS project is especially interesting for its IFC-enabled approach.

The development of the *LexiCon* system performed in conjunction with the Dutch BAS, the EU CONCUR and the Norwegian BARBi projects (Woestenenk, 1998, Woestenenk, 2000) is more pragmatic and targets a related area of building construction. Its scope lies in the categorisation of entities in the AEC/FM industry thereby providing a useful source for initial conceptual specification.

The EU *eConstruct* project utilises the LexiCon for the creation of a common ontology for building construction eBusiness processes, but with different focus. The developed communication-oriented language *bcXML* (Böhms et al., 2001) aims specifically at supporting eBusiness and eCommerce transactions with regard to project supply chains. This work can be seen as complementary to the engineering ontology framework presented herein.

Last but not least, the concept of *design patterns* from the area of object-oriented software design deserves particular mentioning. In the fundamental book of the "gang of four" (Gamma et al., 1995) design patterns are elaborated basically in view of software development but, as shown e.g. by Hay (1996), they can be applied on much broader scope, for various conceptual modelling issues. We use the approach directly in the implementation of the ontology framework but have also borrowed principal ideas from Christopher Alexander's seminal work in architecture (Alexander et al., 1977, Alexander, 1979) for certain elements of the ontology specification itself, and for the creation of *mapping patterns* (used to support the transformations of ontology data to/from a product model, such as IFC).

4. DESIGN OF THE ONTOLOGY FRAMEWORK

4.1 Basic Issues

For sufficient representational adequacy an ontology only needs to establish formal definitions on the level of terms and general concepts, whereas computer-interpreted product data models do basically provide more detailed, but also more rigid and deterministic data abstractions.

On the high level, the relations between terms, concepts and reality can be represented by the so called *Ogden triangle* (Ogden & Richards, 1994). Ekholm (1996) has extended the original idea to suggest a more precise inter-relationship between real-world objects, product data classes and ontology terms (see Fig. 1). According to that, the terms used in a technical, mathematical or computer language are symbols and *designate* concepts, e.g. the term "building" is a symbol that designates the concept "building". The *reference* of the concept "building" is the class of all things with building-like properties. A concept therefore *represents* certain properties of an object, for example the concept "building" represents certain spatial, functional and experimental properties of things belonging to the class of buildings. The *sense* of the concept "building" is given by a context of related concepts, *interpreted* in accordance with personal associations, scientific theories or economic considerations. This interpretation is external to the "building" concept and is not subject to formal ontological specification. There is also a direct relation called *denotation* between a symbol and the reference class of the corresponding concept. For example, the term "building" denotes the reference class of the concept "building". Finally, there is a relation of *proxy* between a symbol and the properties represented by the corresponding real-world object.

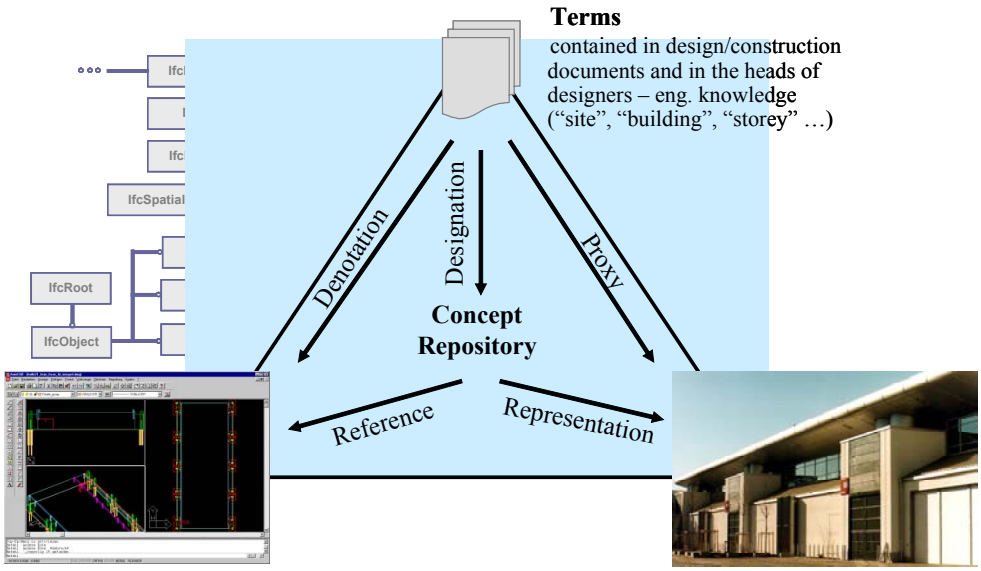


Fig. 1: Fundamental semantic concepts and their inter-relationships (after Ekholm, 1996).

By providing a common conceptual specification where basic terms and concepts are related, technical product data, application data structures, classification system terminology and even real-world objects can be brought together in a unified high-level representation that can be (1) uniformly used by the components of an IT system, and (2) readily accessible to the end user. However, an ontology can give us more than a classification, or a taxonomic hierarchy of classes, including inheritance or subsumption relations. It provides elements of both these representations plus a well-defined set of *operations, describing observable behaviour*.

4.2 Design Criteria

In view of the work of Genesereth & Nilsson (1987), the following design criteria have been adopted in the design of the ontology framework:

1. *Clarity*
   The ontology should effectively communicate the intended meaning of defined terms. Its definitions should be objective and, where possible, stated by means of appropriate logical propositions.
2. *Coherence*
   The ontology should be coherent, i.e. it should sanction inferences that are consistent with its basic definitions (this principle is in fact valid for any conceptual modelling effort as well).
3. *Extensibility*
   The ontology should anticipate the use of a shared vocabulary. It should offer a basis for a range of envisaged tasks, but it should also be possible to extend and specialise its concepts. Developers and end users should be able to define new terms for special uses based on the existing vocabulary, and this should not require revision of already existing definitions.
4. *Minimal ontological commitment*
   The ontology should make as few claims as possible about the modelled real world products, allowing the actors in the environment certain freedom for their own interpretations. In that respect, ontology design is remarkably different from the typical product modelling approaches.
5. *Web-enabled*
   Whilst not directly related to the conceptual design of the ontology framework as such, this final issue is an essential prerequisite for its efficient use in distributed Internet/Intranet environments.

4.3 Ontology Data Structures

The data structures of the ontology framework were developed using the *XML Schema* specification (W3C 2001) for coherence with ifcXML. However, without that specific practical requirement other semantically rich XML-based representations such as RDF, DAML+OIL or the more recent OWL would have been equally appropriate.

As a whole, the framework is constructed in three inter-related layers as follows:

- A *Core Engineering Ontology Specification Schema* (EOSchema) that provides the meta structures for the definition of domain-specific ontologies for different building construction aspects and purposes. This schema is formally defined as an XML Schema instance and is mandatory for the other components of the framework.
- A potentially unlimited set of *domain-specific ontology extension schemas* that imports the core schema and extends it with domain concepts. These schemas must also conform to the XML Schema specification but, provided that the EOSchema is public, they can be freely developed by a third party without any intervening activities of the EOSchema developer. As a practical example of such a schema, a Structural Engineering Ontology (SEO) has been developed in the ISTforCE project (cf. Katranuschkov et al., 2001).
- *XML-based ontology definitions* that correspond to the domain-specific schemas, and provide the details of the identified domain concepts. In contrast to the first two layers, here the specification of concepts is accomplished by using only basic, easy to understand XML syntax, following the representation patterns provided at the first layer, and eventually extended at the second layer (in many practical cases, for such extensions only appropriate natural language names would be sufficient).

This layered approach ensures the required flexibility and extensibility. It can also greatly facilitate the development of adequate ontologies by domain experts because the basic XML syntax used on the third layer is much easier to learn and understand than the semantically richer but also more complex XML Schema constructs.

Another feature that deserves mentioning is the uniform representation of concepts and data on the third layer. This allows to communicate both the definitions and the data related to a specific domain ontology in one and the same document, exchange file or message. Thus, each component of the environment that commits to the ontology framework would only need to be prepared to work with the EOSchema (which is much more concise than the IFC model and can be stored internally or pre-fetched from the Web). The XML-based ontology definitions are self-explanatory and can be interpreted by a commonly installed XML pre-processor. This can be especially useful for non IFC applications and in fact for any heterogeneous environment with weak commitment of the components to common concepts.

4.4 Core Engineering Ontology Specification Schema

The core *EOSchema* serves, simultaneously, three purposes. It includes:

- a formal *meta model* for the definition of specific ontology concepts in schema extensions;
- a formal *data model* for the detailed definition of ontology concepts in a XML document;
- *data model patterns* for the use of the detailed ontology concepts to describe particular XML-based ontology instantiations.

The first of these features is provided by importing and extending the EOSchema in more specific domain ontology schemas (.xsd). This is achieved by designing EOSchema as an extensible model using the *chameleon* namespace approach as recommended in (Costello, 2001). The second and the third features are provided through the developed formal meta ontology definitions which enable the specification of both ontology concepts (classes, operations) and their data instantiation (elements, specific properties, decompositions, relations) in a uniform manner, as separate XSD-validated XML documents, or even in one self-contained document.

Fig. 2 below provides an overview of the top level data structure of EOSchema in EXPRESS-G. Strictly speaking, it is not an exact presentation of the developed specification but it is a useful abstraction to illustrate the overall concept.

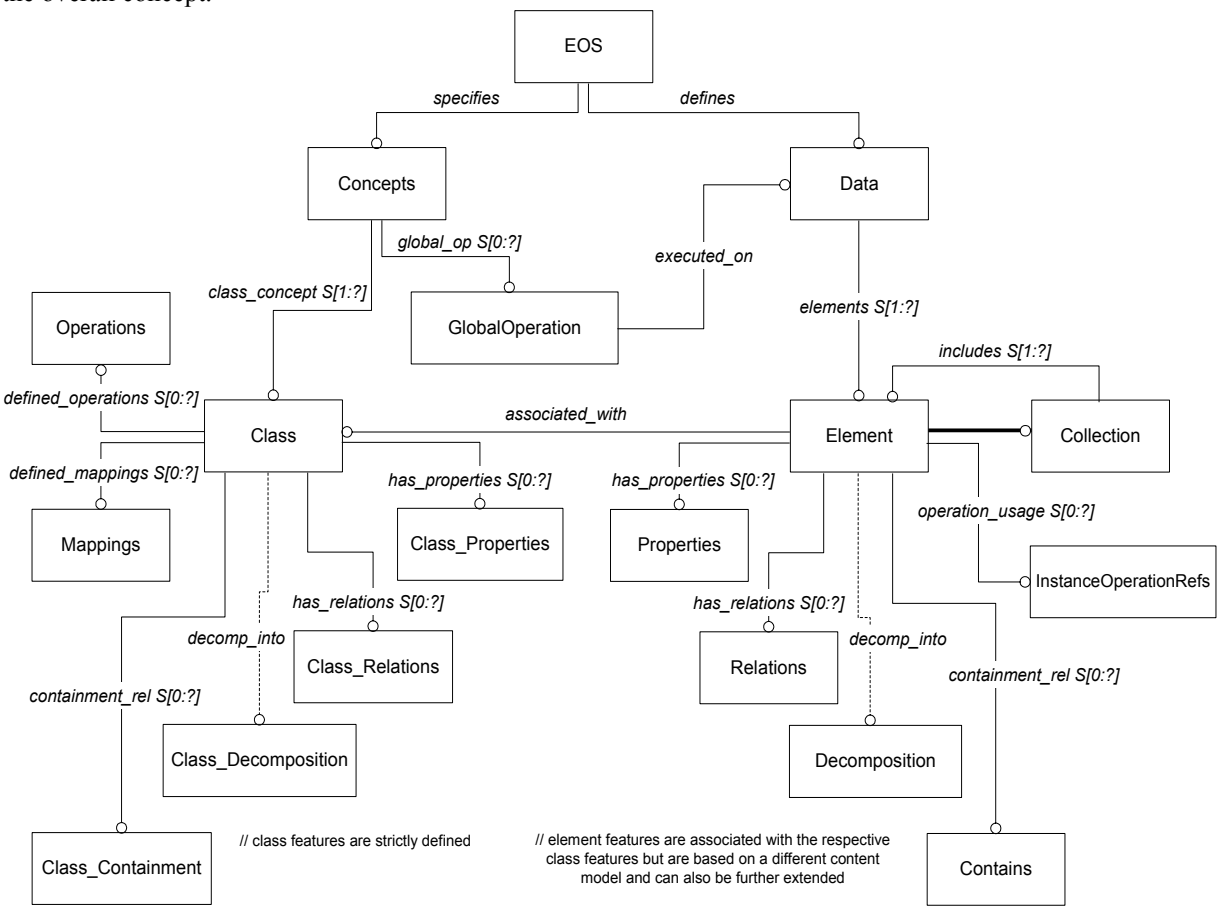


Fig. 2: Overview of the top level structure of EOSchema in EXPRESS-G.

Basically, EOSchema is comprised of two sections: (1) a *Concepts* section (defining how to describe any detailed ontology concepts in a XML document), and (2) a *Data* section (defining how to represent instances of these concepts in the same or another XML instance document).

The EOS content model itself (not shown on Fig. 2) defines also several optional attributes that can be of interest, such as *conceptsRepository* (specifying the location of the ontology concepts, if not included in the same document as the *Data* section), *dataModel* (specifying the data model schema used as basis for mapping the ontology from/to product data) etc.

The *Concepts* section includes the specified ontology concepts (provided by the *Class* type) and an optional set of global operations related to the whole data model.

The *Data* section contains *Elements* and *Collections* that are associated with (but *not* directly instantiated from) respective *Class* concepts; *Element* provides the data type for all basic (tangible or intangible) objects and *Collection* enables the creation of arbitrary groupings of elements and/or other collections at run-time.

Fig. 3, assembled from the graphical notation of the schema with XMLSpy (XMLSpy 2001), illustrates the principal procedure of refining class concepts (steps 1 and 2), data elements (steps 3 and 4), and their association (step 5).

Both Fig. 2 and Fig. 3 demonstrate the great similarity in the specification of concept classes and data elements. However, in EOSchema there is also a principal and very important difference between the *Concepts* and the *Data* section. The first provides a static model for the specification of ontology concepts as "normal" XML data, whereas the latter presents an open, extensible model for the instantiation of these concepts in more detailed

domain schemas or even at run-time. Thus, in broad sense, *Class* can be seen as a factory pattern for *Element*, and *Element* can be seen as a factory pattern for real-world concepts. Indeed, the elements and data types included in the *Concepts* section can also be subtyped, but this would require respective modifications in the ontology tools using EOSchema, such as an Ontology Browser. In contrast, the elements of the *Data* section must be subtyped by respectively extended definitions to be used with their natural names and specific properties in XML-based instance documents. With this approach, instance elements can be automatically provided (and presented to the end user) on the basis of their conceptual specifications. This can be done by a dedicated ontology tool, such as the Explanation Component of the ISTforCE Model Access Service (MAS) discussed in section 5 further below.

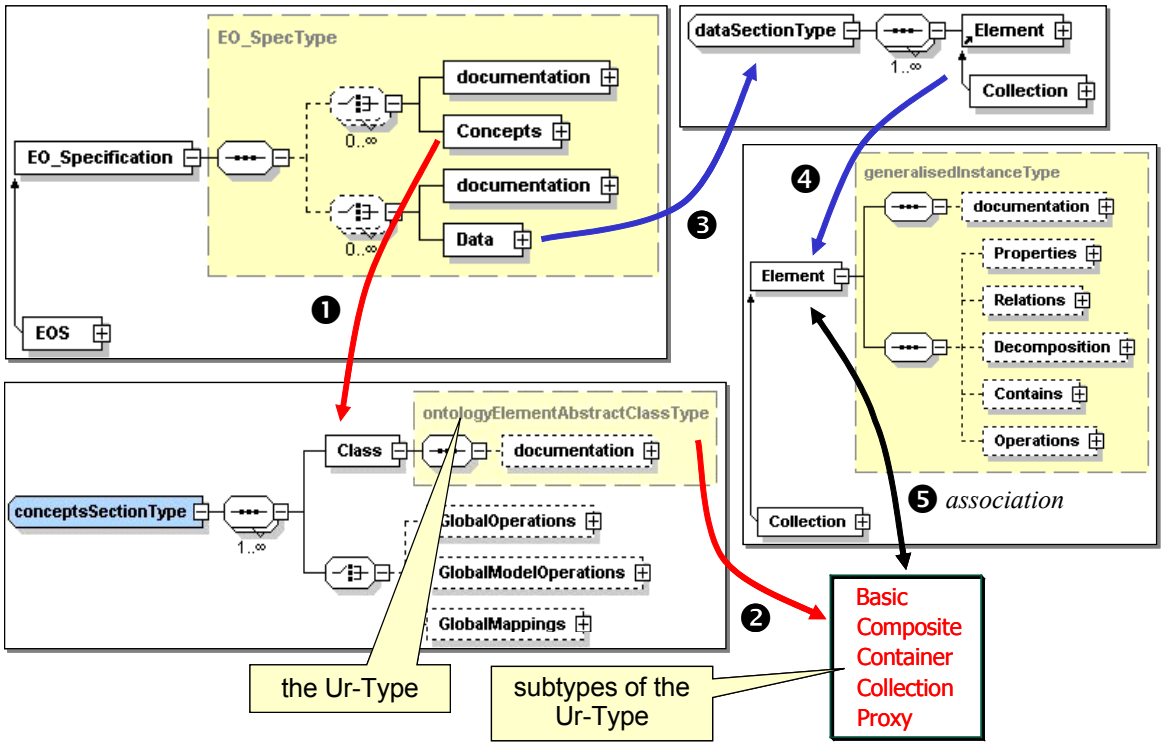


Fig. 3 Schematic presentation of the specification of concept classes and data elements and their principal inter-relationship.

Classes are described by their properties and relations to other classes, including decomposition and containment relationships. Additionally, they define operations that can be executed on the associated elements, and mappings to/from respective product data entities. Except for the notion of operations and mappings, this is very similar to the core model definitions of *IfcRoot* in IFC2x (IAI 2000) and *thing* in ECM (EPISTLE 2001). In fact, all *Class* specifications in EOSchema are derived from an abstract *ur-type* – the *ontologyElementAbstractClassType* –, that is never used for the description of specific ontology classes but is extended into five subtypes as follows:

- *Basic* type, representing atomic, i.e. not further decomposable objects with tangible properties that correspond to real-world concepts such as "beam", "wall", "window" etc., as well as to the respective "leaf nodes" in the IFC project model, directly usable in CAD systems;
- *Composite* type, representing objects assembled from other basic or composite elements by means of strong has-part relationships (aggregations);
- *Container* type, representing different kinds of pre-defined associations that correspond roughly to the respectively defined "triples" in IFC, i.e. ifcRelationshipXX – relatingObject (the Container) – related Objects (the contained elements);
- *Collection* type, providing an abstract definition for collection objects that can be generated ad hoc by an ontology application in response to user requests;

- *ProxyElement* type, providing the representation of "extension" objects for which there is no available formal definition.

Due to the specific rules introduced by XML Schema, each of these types has a separate content model for properties, relations etc. However, although clearly distinguished, these definitions comply to a *common pattern*, as shown in Fig. 4 below. Similar patterns exist on the level of properties and relations as well. For conciseness they are not further explained here, details are provided in (Katranuschkov et al., 2001).

Class subtypes	***Content model pattern***						
	Properties	**Relations**	**Decomposition**	**Contains relation**	**Operations**	**Mappings**	**Documentation**
	⇩ ***Pattern Use*** ⇩						
Basic	1 or more	0 or more	N/A	N/A	1 or more	0 or more	yes
Composite	1 or more	0 or more	exactly 1	N/A	1 or more	0 or more	yes
Container	0 or more	0 or more	N/A	at least 1	1 or more	0 or more	yes
Collection	N/A	N/A	N/A	at least 1	1 or more	0 or more	yes
Proxy	any (optional)	any (optional)	N/A	N/A	optional	N/A	yes

Fig. 4: Pattern for class types and its specific use by the separate subcategories of the CLASS concept.

4.5 Domain-Specific Ontology Extensions

A domain-specific ontology extension is needed to establish common semantics for the engineering services in a targeted application domain. This is achieved by creating:

1. an *ontology extension schema* (.xsd), based on EOSchema and providing the vocabulary of all domain-specific concepts, and
2. an *ontology definition file* (.xsi), based on XMLSchema-instance (see W3C 2001) and providing the actual description of these concepts thereby enabling their proper run-time instantiation.

Basically, the extension schema must import the EOSchema data types and ensure appropriate specialisation of its abstract root element (EOS). It can restrict, extend or override the attributes of EOS in order to associate a correct product data model with the ontology, specify the locations of the ontology repository and the available services, and so on. However, the extension schema does not only make a simple import of the core EOSchema but can provide also a *vocabulary* of domain concepts, declared within a substitution group of the EOSchema *Element*. This is not mandatory but it makes it easier to write and understand ontology instance files using natural language terminology.

To illustrate the idea, let us examine a short example. Without element substitution in the extension schema, a XML instance file would contain instances like:

```
<element name="Beam" … > … </element>
```

Using the extension and substitution mechanisms, the same content information can be encoded as:

```
<Beam> … </Beam>
```

Extrapolating this to a large model with many concepts, properties and relations makes the advantages of the second engineering-like representation obvious (with an add-on for multi-lingual internationalisation).

The ontology definition file contains class concept definitions and global operation declarations. The class concepts serve as templates for the instantiation of actual ontology elements with specific properties, relations, decompositions and element-specific operations. Every class in this definition has to be included in the substitution section of the extension schema to provide the natural language feature. The global operation declarations

define functions concerning the whole available ontology and the associated product model(s). They are not assigned to a specific element but are applied to all elements in any browsing/navigation context.

The following XML code snippet illustrates how this is done for the above "Beam" example.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    elementFormDefault="unqualified" version="1.0">

   <xsd:include schemaLocation="EOSchema.xsd"/>
   <xsd:element name="StructuralEngineeringOntology" type="SEO_SpecType"
                    substitutionGroup="EO_Specification"/>
   ...
   <xsd:element name="Beam" type="generalisedInstanceType"
                    substitutionGroup="Element"/>
   ...

<EO_Specification xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:SchemaLocation="http://cib.bu.tu-dresden.de/istforce EOSchema.xsd"/>

   <Concepts>
   ...
      <Class name="Beam" id="BeamClass" extends="StructuralMemberClass"
                    xsi:type="BasicElementType">
         <Properties>
            <pDef name="crossSectionType" xsi:type="svs" valueType="string"/>
            <pDef name="crossSectionParameters" xsi:type="mvs"
                    valueType="LengthMeasure" componentType="list"
                    componentName="parameter" minCardinality="2"/>
            <pDef name="length" xsi:type="svs" valueType="PositiveLengthMeasure"/>
         </Properties>
      </Class>
   ...
```

All these specifications provide *static* concept definitions (i.e. ontology element prototypes). Their particular instantiation can then be generated at run-time using a dedicated *Ontology Interpreter*. This is explained in section 5.

The principal procedure is illustrated on Fig. 5 below.

EOSchema	OntoExampleSchema (domain extension schema of EOSchema)	Definition of a *Beam* in the <Concepts> section of a XML doc. based on OntoExampleSchema	*Beam* instances in the <Data> section of a XML document based on OntoExampleSchema
Contains all data definition types as described in sect. 4.4 above, including ***"Element"***	<xsd:schema ...> ... <xsd:include schemaLocation="EOSchema.xsd"/> ... <xsd:element name="**Beam**" type="generalisedInstanceType" substitutionGroup="**Element**"/> ... </xsd:schema>	<Concepts> <Class name="**Beam**" … > ... </Class> ... </Concepts>	<Data> <**Beam** id="B1"> <Properties> <Material> C25 </Material> </Properties> </Beam> <**Beam** id="B2"> ... </Data>

Fig. 5: Example for the layered representation of concepts and their inter-relationship enabling dynamic generation of ontology instances (the last column presents the input to a Web Browser via XSLT)

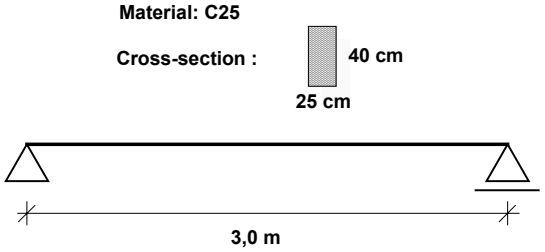
Fig. 6 provides a simple example showing the inter-relationships between the ontology specification schema and the respectively generated definition file. The whole currently developed ontology specification for the structural

design domain is much more complex, spanning over more than 25 pages as reported in (Katranuschkov et al, 2001). This report contains also the full EOSchema specification and several more comprehensive examples.

Example ontology specification (example.xml)

```
<EO_Specification>
  <Concepts>
    <Class name="Beam" id="BeamClass" xsi:type="BasicElementType">
      <Properties>
        <pDef name="Material" valueType="string"/>
        <pDef name="SectionArea" valueType="real"/>
        <pDef name="JointList" ref="JointClass"
              componentName="Joint" componentType="ulist"
              minCardinality="2" maxCardinality="2"/>
      </Properties>
    </Class>
    <Class name="Joint" id="JointClass" xsi:type="BasicElementType">
      <Properties>
        <pDef name="Coordinates" valueType="real"
                componentName="x y z" componentType="list"
                minCardinality="2" maxCardinality="3" units="{m}"/>
        <pDef name="Support" valueType="string"/>
      </Properties>
    </Class>
  </Concepts>
</EO_Specification>
```

and its respective instantiation, representing the simple beam shown below:




```
<EO_Specification conceptRepository="example.xml">
  <Data>
    <Beam id="Beam-1">
      <Properties>
        <Material> C25 </Material>
        <SectionArea units="{cm}^2"> 1000.0 </SectionArea>
        <JointList>
          <Joint id="Joint-1">
            <Properties>
              <Coordinates> <x> 0.0 </x> <y> 0.0 </y> </Coordinates>
              <Support> FXY </Support>
            </Properties>
          </Joint>
          <Joint id="Joint-2">
            <Properties>
              <Coordinates> <x> 3.0 </x> <y> 0.0 </y> </Coordinates>
              <Support> FY </Support>
            </Properties>
          </Joint>
        </JointList>
      </Properties>
    </Beam>
  </Data>
</EO_Specification>
```

Fig. 6: Example ontology definition for a simple beam.

4.6 Operations

Operations provide a formal specification of commonly agreed behaviour. They are a means to enhance the functionality of ontology-based tools with services that can be performed both *internally*, i.e. in the local context of the tool, and *externally*, by invoking a Web service.

Currently we use a light-weight, but sufficiently flexible approach based on an adapted version of the Web Interface Definition Language (WIDL) for that purpose. However, by the same principle more comprehensive specifications such as SOAP and WSDL can be applied in future versions of the framework.

WIDL (W3C 1997) defines a meta language that implements a service-based architecture over the document-based resources of the Web. It can be used to (1) automate interactions with Web services, (2) provide both on demand and scheduled extraction of data, (3) aggregate data from a number of Web sources, and (4) chain services across multiple Web sites.

The application of WIDL in the ontology framework is provided by first translating the appropriate WIDL concepts from the original XML DTD to XML Schema and then integrating these concepts into the ontology data structures. This is achieved with only a small set of additional data types:

- an *operationsType*, providing an envelope for the class-level specification of WIDL-based services, and a component model for all predefined types of operations, i.e. global operations, global model operations, local object-level operations and mappings,
- an *instanceOperationsType*, providing a respective model for the use of operations within element instances,
- a *serviceType* defining WIDL-style operations,
- a *bindingType* defining input and output parameters for services,
- a *conditionType* defining a condition within a binding,
- a *variableType* enabling the definition of variables within a binding, and
- an *operationRefType* enabling the referencing of operations defined in *Class* concepts from the respective element instances in the *Data* section of a XML document.

The following code snippet illustrates the definition of operations. It presents the supertype of the *Beam* element from the example given in the previous section 4.5. Note that by inheritance these operations are automatically applicable for all subtypes as well, i.e. *Beam*, *Column*, *Slab*, *Wall*, etc.

```
<Class name="StructuralMember" id="StructuralMemberClass"
       xsi:type="BasicElementType" abstract="true">
...
   <Operations>
      <Service name="getSameClassInstances"
         URL="getSameClassInstances"
         methodType="local" output="structuralMemberInstanceList"/>
      <Service name="getSameClassInstancesWithinStorey"
               URL="getSameClassInstancesWithinStorey"
               methodType="local" output="structuralMemberInstanceList"/>
      <Binding name="structuralMemberInstanceList" type="output">
         <variable name="instanceListOutput" type="Collection"/>
      </Binding>
   </Operations>
</Class>
```

4.7 Mappings

The last, very important yet difficult part of the realisation of the ontology is the mapping of the ontology concepts to/from a product data model. Typically, the structure and the semantics of ontology concepts and product data classes will differ, at certain places considerably. This gives rise to a variety of structural and semantic model transformation problems that are commonly known as *mapping*.

In principle, model mapping involves a procedure that is very similar to mutlidatabase integration known from the area of database research (Spaccapietra et al., 1992, Reddy et al., 1994). It consists of the following steps:

1. Detection of schema overlaps,
2. Detection of inter-schema conflicts,
3. Definition of the inter-schema correspondences with the help of formal mapping specifications,
4. Use of appropriate mapping methods for the actual transformations on entity instance level at run-time.

Unfortunately, the mapping problem cannot be solved in the general case of arbitrary complex object-oriented data models as it is hardly possible to cover or even to identify all potential mapping cases. Nevertheless, for certain practical use cases there are a number of mapping languages and respectively developed tools that have provided most promising results. Among these are VML (Amor, 1997), CSML (Katranuschkov, 2001), and especially EXPRESS-X (ISO TC184/ SC4/WG11/N088 1999), strongly promoted in conjunction with ISO STEP. However, all these efforts are confined to mapping languages that describe inter-related transformations of the entities of whole data models; interactive mapping of individual concepts, as needed in the case of the suggested ontology framework, are not available.

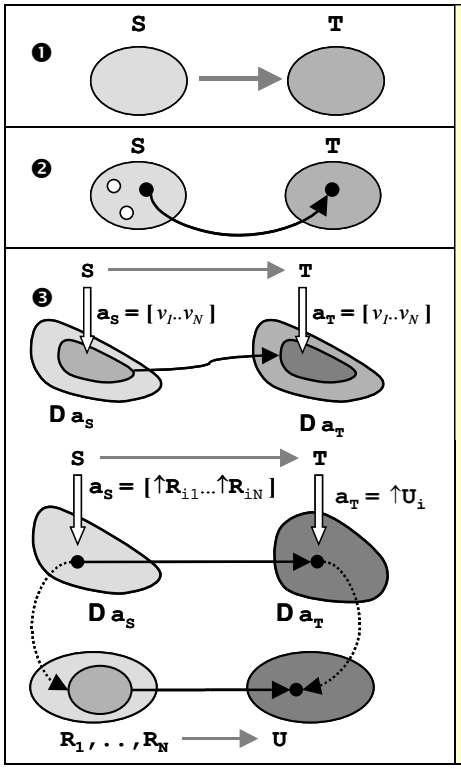
Consequently, we do not use a mapping language to deal with that problem. Rather, we have defined a number of *mapping patterns* from which individual mapping operations can be constructed. These operations are then specified in the ontology schema in the same way as all other concept/element level operations, i.e. without going into the details of their implementation. The approach is in fact quite similar to the manner mapping problems are dealt with in EDM (Eastman et al., 1995) but it provides a more structured method to identify mapping cases on the high level.

The identification of typical mapping patterns allows to understand better the mapping task and to formalise what and how has to be mapped in each particular case. Specifically, by examining the theoretical background of object-oriented modelling as well as a number of practical cases, the following set of mapping patterns can be identified:

- *Unconditional class level mapping patterns*: These patterns depict the most general high-level mappings that affect all instances of the involved source and target classes.
- *Conditional instance level mapping patterns*: In contrast to the general definition of mappings on class level, instance level mappings may include certain conditions to select the set of instances to be mapped from the full set of available instances in the source model, or – in the case of multiple cardinality – from the cross product of all referenced sets of instances.
- *Attribute level mapping patterns*: These patterns depict how an attribute with a given data type should be mapped. However, in object-oriented models the data type of an attribute can be quite complex. Consequently, there are many different patterns that have to be considered here, in several cases they can even govern the actual resulting cardinality of the mapping on entity level.

Attribute mapping patterns are further subdivided into: (1) basic attribute mapping patterns, (2) complex attribute mapping patterns, and (3) generative attribute mapping patterns. For each of these categories, several sub-cases have been identified, such as *"simple equivalence"*, *"set equivalence"*, *"functional equivalence"* for the first category; *"grouping"*, *"ungrouping"*, *"homomorphic mapping"*, *"transitive mapping"*, *"inverse transitive mapping"* for the second category; *"simple generative"*, *"functional generative"* for the third category, and so on. Fig. 7 shows the formalism used to present these types of patterns graphically, along with respective typical examples.

As a whole 29 different mapping patterns have been defined in (Katranuschkov, 2001). Some of these are relatively simple to implement but there are also several patterns that require quite sophisticated implementation methods. However, for the specific mapping cases regarding the IFC model not all of these patterns are really needed. IFC defines several pragmatic requirements restricting the use of EXPRESS modelling concepts which simplifies considerably the mapping task. In fact, for the purposes of the currently developed ontology specifications only partial mapping of certain IFC entities is of interest. Basically, this is a subset of the so called "leaf nodes" in the IFC specification, such as *IfcBuilding*, *IfcBuildingStorey*, *IfcColumn*, etc., along with their related resource objects and property sets. The involved mapping patterns for these cases are mostly not very difficult to implement because the source and target entity sets can be directly derived from 1:1 or 1:C class level correspondences.



❶ ***Class level mappings***
S and T (eventually with a subscript) are used to denote the source and the target class respectively. The ovals show the domains of the classes, the grey arrow shows the direction of the mapping.
The given example depicts a 1:1 mapping S → T.

❷ ***Conditional entity instance level mappings***
In addition to the above symbols, here the black dots represent the instances being mapped, and the white dots represent the instances that are discarded from the mapping as a result of an applied *condition*. The particular mapping equivalences are shown with black arrows. The given example refines the above 1:1 class mapping.

❸ ***Attribute level mapping patterns***
For these patterns, a few more graphical symbols are introduced. To save space, the class mapping (**S** → **T**) is indicated merely with an arrow. The domains of the attributes (**D**a) are shown as free-form areas, and the attribute values are shown either with dots (for single values), or with a darker area contained in the domain (for lists or sets). As above, the addressed mapping equivalences are indicated with black arrows. The thick white arrows denote the attribute's designation to a given class (**S**, **T**); pointer references to other objects within a model are shown with dotted arrows.
The first given example depicts a 1:1 set equivalence. The second example shows a homomorphic mapping, This is a complex equivalence, involving two pairs of mappings due to the references from $\mathbf{a}_S$ and $\mathbf{a}_T$ to the object instances [$\mathbf{R}_1$,...,$\mathbf{R}_N$] and **U** respectively.

Fig. 7: Graphical symbols used for the presentation of mapping patterns with illustrative pattern examples.

However, even in simpler cases, IFC relationships between objects are generally very deep. Most typical in that respect is the deep nesting of geometric attributes. In contrast, in the developed engineering ontology objects like *Column* are represented in highly compact form. Thus, for the mapping of ontology concepts to IFC the *transitive* and the *inverse transitive* mapping patterns will be most frequently applied. For reference, they are presented on Fig. 8 below.

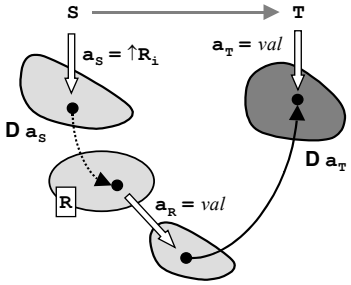
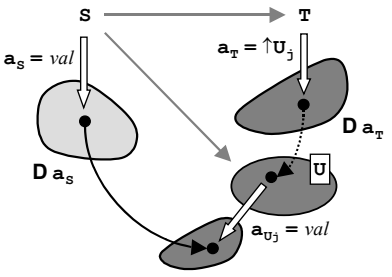
Attribute Mapping Pattern	Schematic presentation of the mapping transformations	Description
Transitive (telescope)		This pattern is well-known from the field of database integration. It represents the situation where the value of a source attribute $\mathbf{a}_R$ referenced through a pointer in $\mathbf{a}_S$ is stored directly into the target attribute $\mathbf{a}_T$. The schema on the left presents the most common case of a single reference used to access $\mathbf{a}_R$, but in general such references can also be chained.
Inverse transitive		This pattern is the inverse of the previous one. Its realisation is more difficult than by the forward mapping because, in order to create the value $\mathbf{a}_T = \uparrow\mathbf{U}_j$, all referenced instances have to be constructed in the target *before* the value of $\mathbf{a}_T$ can be properly set.

Fig. 8: Commonly applicable mapping patterns for IFC product data.

Fig 9 illustrates the said deep level of nesting for IFC "leaf nodes" on the example of *IfcColumn*. Fig. 10 presents schematically the overall approach for the implementation of mappings using as an example an instance of *IfcColumn* taken from a sample IFC exchange file.

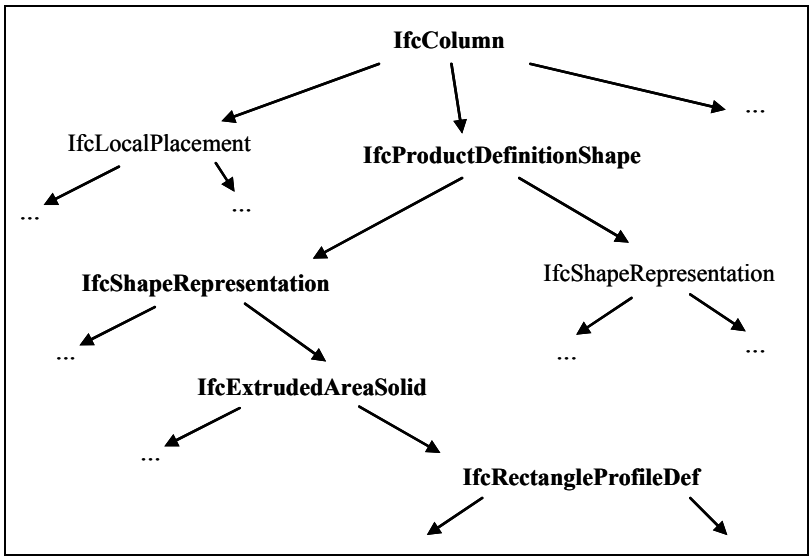


Fig. 9: Schematic presentation of a part of the IfcColumn relationship hierarchy

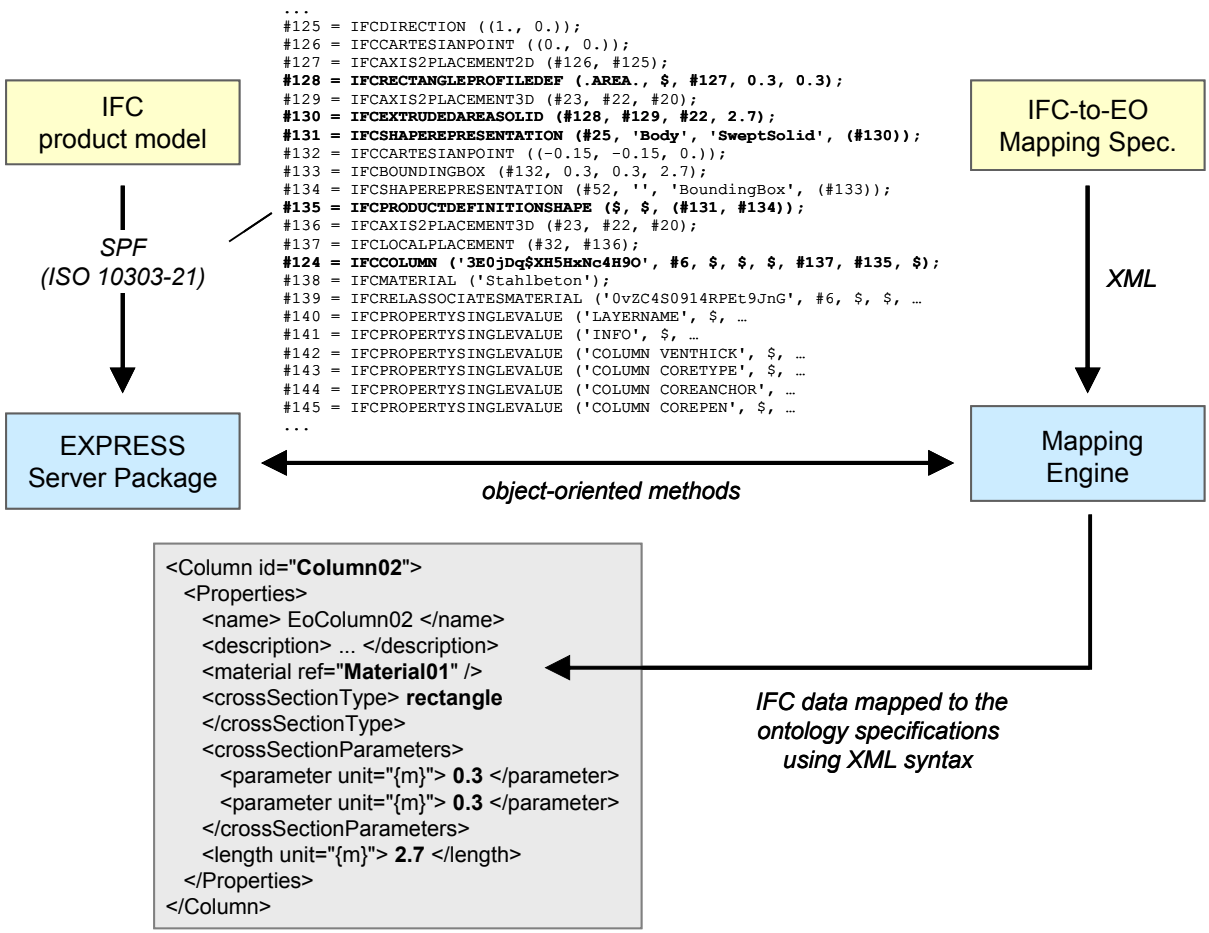


Fig. 10: Principal schema of the developed mapping approach.

5. APPLICATION

The suggested ontology framework can be used in various ways for a range of practical tasks derived from the generic use cases described in section 2.3 above. A prototype implementation of the framework carried out in the frames of the EU ISTforCE project provides proof of concept and hints to other possible options. In this section we present the realisation of the framework within the ISTforCE Model Access Service (MAS), together with small browsing/navigation examples. We show further how the ontology can be used with other conceptual models, and outline other envisaged options. More details about the implementation of the ontology framework in ISTforCE are provided in (Katranuschkov & Gehre, 2002).

5.1 Use of the Ontology Framework within the ISTforCE Model Access Service (MAS)

The goal of MAS in ISTforCE was to enable both simple and advanced human-centred product model functionality. Beside whole model access (via SPF) and remote procedure calls for product data access on object level (using Java RMI or CORBA), it provides also two additional advanced services: (1) a *Reasoning Agent*, supporting the execution of complex automated structural design tasks, and (2) an *Explanation Component,* facilitating human-centred product data retrieval for engineers not familiar with the technical IFC structure and specifications. The latter utilises the developed Engineering Ontology Framework and is therefore of particular interest here.

5.1.1 Ontology Processing in MAS

Along with the access to product model data via engineering applications capable of connecting to MAS directly, we set out to develop an ontology-based user-friendly interface that would enable viewing and retrieving of the model data via a standard Web Browser. In accordance with that, a specific task of MAS was to provide a translation of the strictly formalised IFC data structures, which are not readily understandable to the end user, to the engineering vocabulary he is used to work with. This task was accomplished with the help of the developed Engineering Ontology framework.

Ontology processing in MAS is activated by a client-side *Ontology Navigator* (Web Browser) which provides the front-end user interface for browsing/inspecting/manipulating the product data by means of model requests utilising familiar engineering semantics. Thus, the engineer can retrieve and modify product model information fast and without deeper knowledge of the underlying IFC structure, and at the same time he can be aware of the engineering meaning and structuring of the data.

As a simple example, consider the ontology concept *frame* which is a part of the specified structural engineering vocabulary but is not contained within the IFC product model schemas. By utilising the developed Engineering Ontology framework, engineers can use this concept to navigate through model parts defined as "frames", that are "known" only as beams and columns in the IFC model. Another example is the "join" operation that brings together information scattered around the model to one *object*, e.g. the ontology representation of a compound element contained in several IFC objects providing its material details, connectivity, geometry, location etc.

Within MAS, the ontology specifications are stored in an *Ontology Repository* which provides functions to retrieve, update, add and validate them (see Fig. 11).

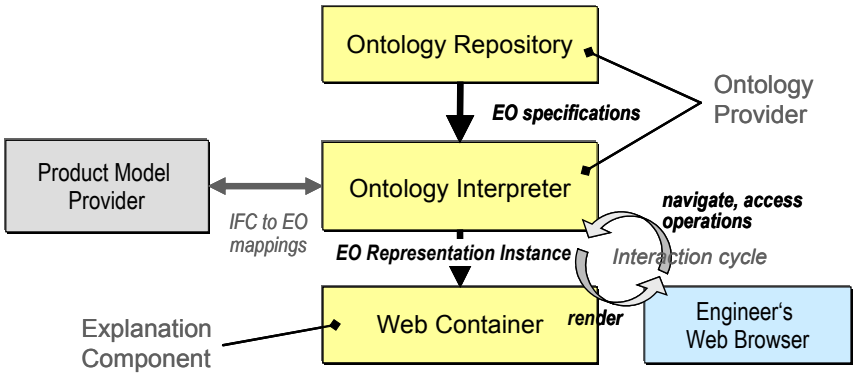


Fig. 11: Information flows in ontology processing.

The central processing unit for the ontology framework is the *Ontology Interpreter*. It is initialised with an exclusive reference to an instance of the MAS internal product model service, so that it can act on the product model directly.

The Ontology Interpreter uses the specifications provided by the Ontology Repository for real-time *mapping* between the IFC model data and the implemented ontology, as suggested in (Katranuschkov, 2001). However, as this may often be a complicated task by itself, and because not all information is present in both models, a complete mapping cannot always be achieved. Therefore, the mapping process is supported by the additional information provided in the extended specification schemas and ontology definition files, and in some cases even involves user interaction. As a result of the process, the Ontology Interpreter generates *Engineering Ontology Representation Instances* as "pure" XML files. They can be seen as a special ontology based "view" on the underlying IFC model.

The Representation Instances are then delivered to the *Web Container* of the MAS, responsible to provide the pure XML files (if the client is an advanced ontology enabled application), or to generate browser-compatible human readable HTML content, respectively. The translation is done by using XSLT and JSP technology to create one or more HTML files that are readily parsed and presented by a standard Web Browser. Moreover, the HTML files do not only provide the "raw" ontology view of the model but are also enhanced by navigation and access operations defined as hyperlinks in the HTML document that can again access the Web Container. Through this recursive navigation process, schematically illustrated on Fig. 11 above, comprehensive browsing of the IFC product model becomes possible.

5.1.2 Browsing / Navigation

Navigating through the ontology based model is performed via a text-centred interface supported by an externally linked graphical viewer. Fig. 12 illustrates the principal navigation capabilities and the textual content presentation.

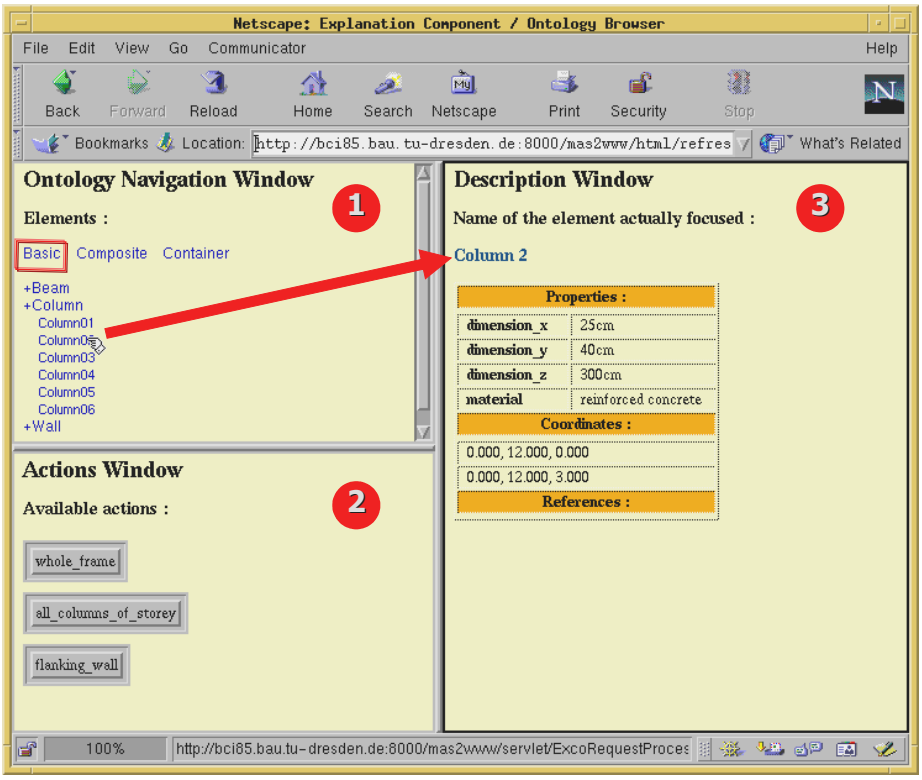


Fig. 12: The MAS Explanation Component User Interface.

The division of the navigator window into three specific frames facilitates ontology and content processing on the server side and provides better orientation for the user. The first window frame in the upper left corner allows navigation on the basis of the defined fundamental ontology concepts. The presented navigation elements are

structured like the ontology itself, categorised in the three main element types *Basic*, *Composite* and *Container*. The two other principal types – *Collection* and *ProxyElement* –, are currently not included in the *Navigation* frame as they are mostly related to server internal processing and the handling of responses to ad hoc natural language queries that are still under development.

The *Actions* frame below the *Navigation* frame provides functionality that is directly derived from the specified operations in the ontology schema (see section 4.6). Hence, following the ontology element that is currently focused, the user can find here context-sensitive operations that are of interest for the particular context. For example, if a "column" is under the current focus, the *Actions* frame provides operations like "whole frame", "all columns of storey" and "flanking walls".

The *Description* frame on the right side of the *Navigation* frame displays the information related to the focused ontology element. Here the user can view the available properties, as well as references to other elements. In the context of this window frame such references are only considered for composite elements that are assembled from other elements. For example, if a structural frame is focused, references to all columns and beams from this frame will be presented.

In addition to these features, MAS supports also geometry-based viewing/navigation for IFC 1.5.1 and IFC2x models via a plugged-in external IFC viewer. Fig. 13 below gives an impression of this add-on feature.

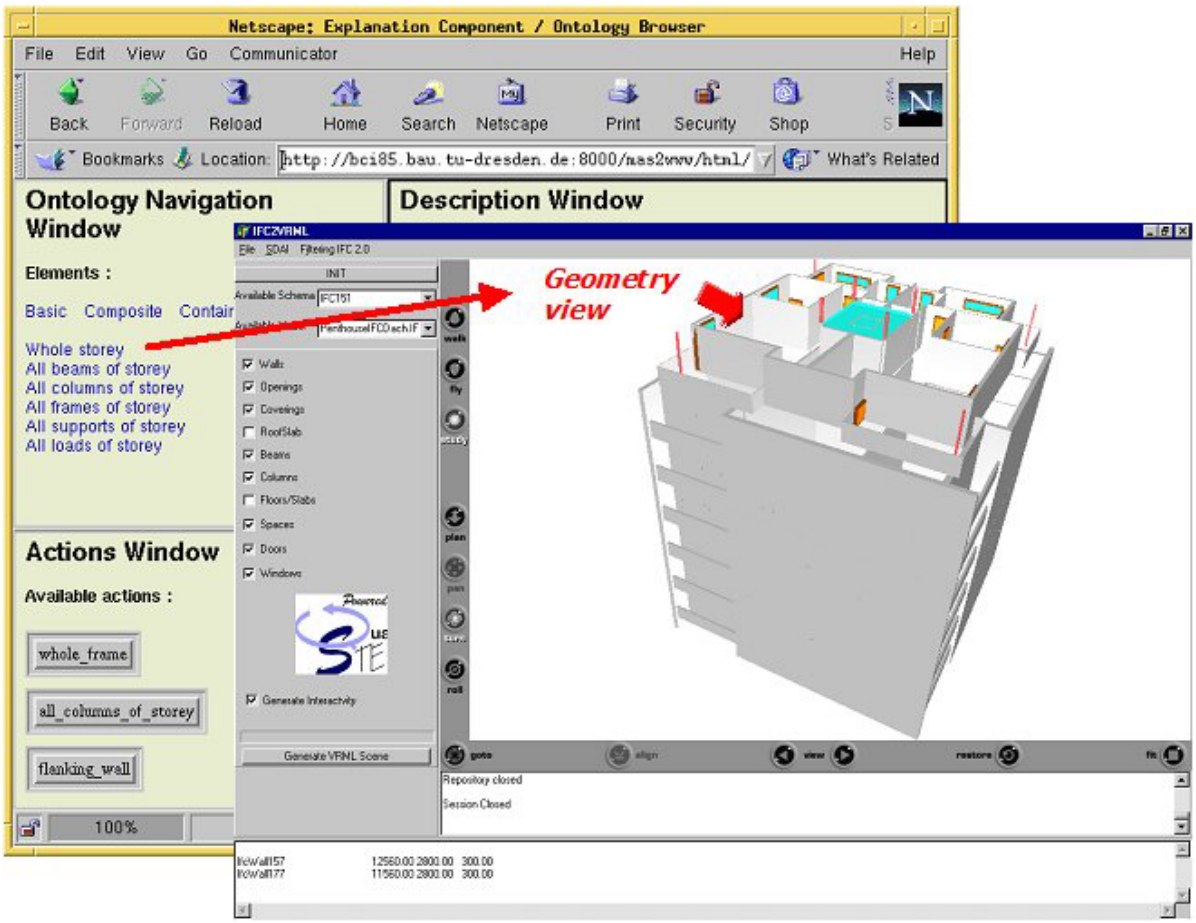


Fig. 13: Typical screenshot of client-side ontology navigation (the geometry view is supported by a plugged-in IFC Viewer developed in the German iCSS project by FIDES, Munich).

As a whole, the realisation of the ontology in MAS follows the *MVC pattern* (model-view-controller). The model layer is constituted by the object-oriented product data provided by a product data server and the Ontology Repository of the MAS. The Ontology Interpreter acts as the controller for the user actions and as mapper between the ontology definitions and the IFC data objects. The view layer is provided by the Ontology

Navigator and the underlying Web Container that generates the actual presentation documents and/or input forms.

5.2 Using the Ontology Framework with LexiCon

The implementation of the suggested ontology framework in the ISTforCE project specifically addresses structural engineering applications and IFC model data. However, the framework is not limited to that. As already mentioned, one of its major features is its extensibility. Based on a small set of concepts provided in the core EOSchema various specific ontologies can be created in a straight-forward manner and with reasonable effort. Below we give an example from an undertaken exercise intended to show that such extensions basically follow the same development pattern.

For demonstration purposes the classification provided by the *LexiCon* system (Woestenenk, 2000) has been selected. LexiCon is a popular development used partially or fully in several research projects (eConstruct, PROCURE, BARBi). To keep the presentation concise we describe here only the representation of a single LexiCon "built object" in the ontology.

Fig. 14 presents a screenshot of the available LexiCon tool at the time of the performed study. The mouse focus is on the chosen *"beam"* object.

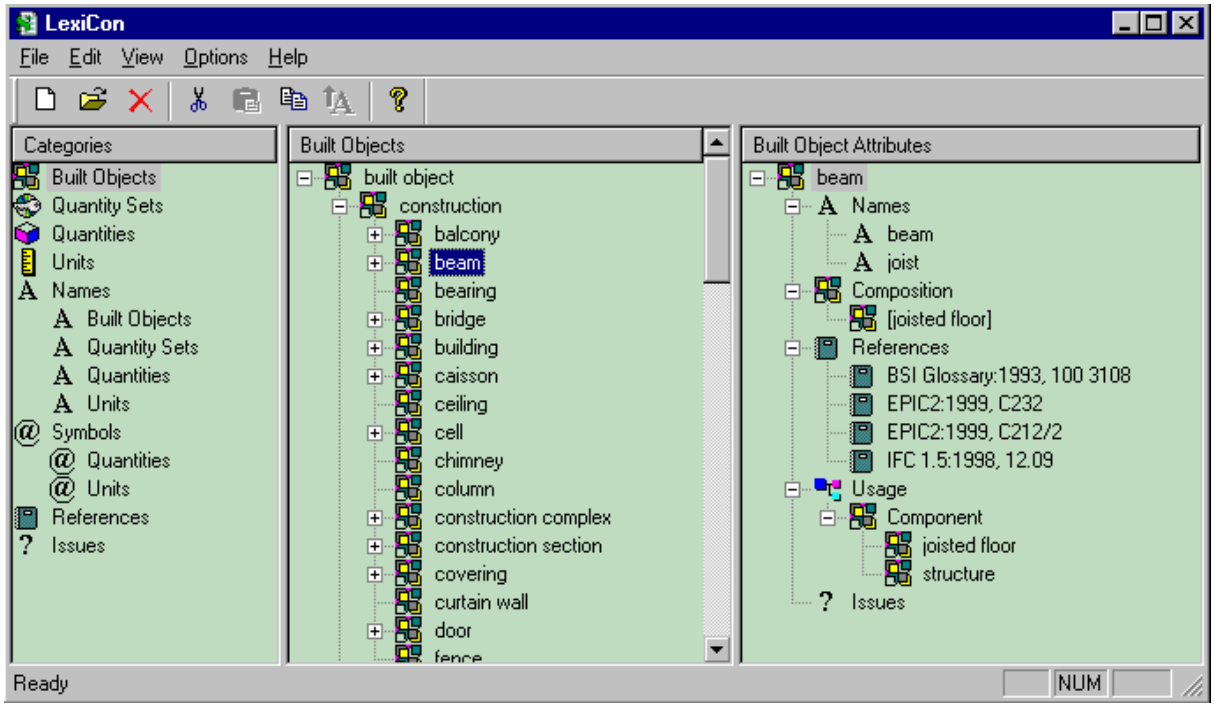


Fig. 14: The element "beam" focused within LexiCon (Woestenenk, 2000).

The LexiCon tool provides a GUI that is quite similar to the Ontology Browser shown in the previous section. It divides the user window in 3 frames corresponding to the developed classification approach, i.e. (1) *Categories*, (2) *Built Objects*, and (3) *Built Object Attributes*. Navigation is possible via any of these frames whereby each provides a respective functional context.

For the chosen example the ontology class definitions and element data are provided in the same XML file and can therefore be sent together to the ontology interpreter. A LexiCon extension schema specifies additionally the natural language vocabulary of terms corresponding to the defined class patterns.

Fig. 15 presents the formal representation of the LexiCon "beam" in the ontology. By comparing that with the earlier shown examples, the uniform approach with respect to schema extensions becomes readily visible.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    elementFormDefault="unqualified" version="1.0">
  <xsd:include schemaLocation="EOSchema.xsd"/>
  <xsd:element name="LexiConOntology" type="LexiConSpecType"
                    substitutionGroup="EO_Specification"/>
  <xsd:element name="beam" type="generalisedInstanceType"
                    substitutionGroup="Element"/>
  ...
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<EO_Specification xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:SchemaLocation="http://cib.bu.tu-dresden.de/istforce EOSchema.xsd" />
<Concepts>
  ...
  <Class name="be  " extends="ConstructionElement" xsi:type="BasicElementType">
    <Properties>
      <pDef name="Names" xsi:type="mvs" valueType="string"
componentName="Name"/>
      <pDef name="Issues" xsi:type="svs" valueType="text"/>
    </Properties>
    <Relations>
      <relDef name="Composition" ref="ConstructionElement"
          minCardinality="0" maxCardinality="unbounded"/>
      <relDef name="References" ref="ReferenceElement" minCardinality="0"
          maxCardinality="unbounded"/>
      <relDef name="Usage" ref="##any" minCardinality="0"
          maxCardinality="unbounded" componentName="Component"/>
    </Relations>
  </Class>
   ...
</Concepts>
<Data>
  ...
  <beam>
    <Properties>
       <Names> <Name> beam </Name>  <Name> joist </Name> </Names>
    </Properties>
    <Relations>
       <Composition ref="joisted floor" />
       <References>
         <rel ref="BSI Glossary, 1993, 100 3198" />
         <rel ref="EIPC2:1999, C232" />
         ...
       </References>
       <Usage>
         <Component ref="joisted floor"/>
         <Component ref="structure"/>
       </Usage>
     </Relations>
   </beam>
   ...
</Data>
</EO_Specification>
```

Fig. 15: Formal definition of the LexiCon "beam" in the ontology.

5.3 Further Possibilities

The implementation of the ontology framework in ISTforCE is prototype work that can be extended in several aspects. The developed approach is not limited to the objectives of a user-friendly model access service. Other possibilities that can be envisaged include:

- *Conceptual development of domain ontologies*
  We showed in the preceding two section that the suggested framework can easily be applied to different underlying models by the same methodology. An open question is if schemas using more complex modelling paradigms can always be adequately "translated" to the ontology. Another question is the scalability of the approach by huge real-world models.
- *Establishing an environment ontology for distributed Web-based systems*
  Such an ontology can provide a formal basis enabling efficient management of various communication and information exchange issues in distributed ICT systems. Especially the light-weight XML definitions at end-user and application level can greatly facilitate the integration of a wide range of services. Here the same approach can be applied as for domain ontologies but – as there do not yet exist formal models for such purposes –, conceptual development must be done almost from scratch.
- *Support to context awareness*
  The achievement of context aware information gathering, retrieval and sharing in mobile environments (e.g. on construction sites, for monitoring of infrastructure facilities etc.) involves several demanding aspects such as (1) contextualisation and personalisation of available information from various sources, (2) explication of the created context in easy-to-understand manner to the end user, (3) environment-wide interoperability of a variety of hardware devices and software services. The second of these issues is related to the current use of the suggested ontology framework, and the third can be seen as an extended version of the previous listed item.

6. CONCLUSIONS

Appropriate support of PDT environments by domain-specific ontologies can greatly improve their value-adding capabilities in construction project work. Ontologies can hide most of the complexity of a product model from the end users and can pave the way for more user friendly interfaces. Hence, they can help to achieve faster deployment and broader use of product data technology in daily engineering practice.

From the presented research, the following specific conclusions can be drawn:

1. Using XML technology to design the Engineering Ontology provided excellent capabilities with regard to the flexibility and maintenance of the software system.
2. The development of the ontology framework as part of an extensible and open architecture greatly facilitated the realisation of the particularly targeted ontology for the structural design domain and at the same time provides a methodology for the implementation of ontologies for other AEC/FM domains in a similar fashion.
3. Even though only a prototype version of the described Explanation Component had been achieved by the end of the ISTforCE project, it proved to be very practical in the first tests performed with end users that were not very familiar with the IFC model specifications.

However, during the implementation there appeared also some caveats that need to be considered in future work.

The mapping process between the IFC model and the ontology is complicated, and often too slow. Additional work is needed to refine mapping methods so that they can be fully usable in practice. Moreover, the structural design ontology definitions should be extended both in horizontal and in vertical direction (sets of elements and their particularisation). A medium-term goal for future efforts is also the harmonisation and subsequent integration of the developed approach with the bcXML specification – with expected mutual benefits with respect to functionality, scope and life cycle processes covered.

It would be early to draw conclusions about the practical benefits of the presented approach only on the basis of the prototype ISTforCE implementation. A further exercise that was conducted recently to test the applicability of the ontology for IFC "leaf node" entities (Grosser, 2003) showed very promising results but it also does not allow to analyse objectively the scalability of the developed framework for real use cases. Nevertheless, the authors are convinced that the presented ideas can give useful hints for a number of options that are conceptually possible. Verification and refinement of the approach as well as further specific objectives are expected to take shape in the future progress of work.

7. ACKNOWLEDGEMENTS

The presented research was enabled through the support of the EU FP5 ISTforCE project (IST 1999-11508) and partially through a grant of the German Research Foundation (DFG) in Priority Program 1103. The contribution of the funding agencies is herewith gratefully acknowledged.

8. REFERENCES

Alexander C. 1979. *The Timeless Way of Building*. Second ed., Oxford University Press, ISBN 0195024028, 568 p.

Alexander C., Ishikawa S. & Silverstein M. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, ISBN 0195019199, 1216 p.

Amor R. 1997. *A Generalised Framework for the Design and Construction of Integrated Design Systems*. Ph. D. Thesis, Univ. of Auckland, New Zealand, 350 p.

Angus C. & Dziulka P. 1998. *EPISTLE Framework, V.2.0*. Issue 1.22, EPISTLE Report, Doc. reference 98-078. Available from: http://www.stepcom.ucl.ac.uk

Benjamin J., Borst, W. N., Akkermans, J. M. & Wielinga, B. J. 1996. Ontology Construction for Technical Domains. In: Shadbolt N., Schreiber G., O'Hara K. (eds.) *Advances in Knowledge Acquisition: Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW '96)*, Nottingham, UK, May 14-17, Springer-Verlag, New York, ISBN 3540612734, pp. 98-114.

Björk B.-C. 1992. A Unified Approach for Modelling Construction Information. *Building and Environment,* 27(2), Pergamon Press, Elsevier, ISSN 0360-1323, pp. 173-194.

Björk B.-C. 1995. *Requirements and Information Structures for Building Product Data Models*. VTT Publ. N 245, Espoo, Finland, 87 p.

Böhms M., Bonsma P., Tolman F., van Rees R., Fleuren J & Tøn A., 2001. *Final Edition of the bcXML Specification*. Deliverable D103, EU Project IST-1999-10303 eConstruct.

Borst P., Akkermanns H. & Top J. 1997. Engineering Ontologies. *International Journal of Human-Computer Studies*, 46(2-3), Academic Press Limited, Elsevier Computer Science, ISSN 1071-5819, pp. 365-406.

Costello R. L. 2001. *XML Schemas: Best Practice.* http://www.xfront.com/BestPracticesHomepage.html

Eastman C., Jeng T.-S., Assal H., Cho M. & Chase S. 1995. *EDM-2 Reference Manual.* Tech. Report, Center for Design and Computation, UCLA, Los Angeles, CA, 50 p.

Ekholm A. 1996. A conceptual framework for classification of construction works. *Electronic Journal of Information Technology in Construction (ITcon),* vol. 1., pp. 25-50, ISSN 1400-6529. Available from: http://www.itcon.org/1996/2

EPISTLE 2001. *EPISTLE Core Model, Version 4.0*. European Process Industries STEP Technical Liaison Executive. Available from: http://www.epistle.ws

Eschenbach C. & Heydrich W. 1995. Classical Mereology and Restricted Domains. *International Journal of Human-Computer Studies*, 43(5-6), Academic Press Limited, Elsevier Computer Science, ISSN 1071-5819, pp. 723-740.

Gamma E., Helm R., Johnson R. & Vlissides J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, ISBN 0201633612, 395 p.

Gehre A. & Katranuschkov, P. 2000. *Engineering Ontology*. Deliverable D5-1, EU Project IST-1999-11508 ISTforCE, 31 p.

Genesereth M. R. & Nilsson N. J. 1987. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, ISBN 0934613311, 406 p.

Grosser A. 2003. *Vernetzt kooperative Planung: Entwurf und Validierung eines Programms zur ingenieursprachlichen Abbildung, Klassifikation und Internet-gerechten Darstellung des IFC-Produktmodells*.

Student Research Project October 2002 - January 2003, TU Dresden, Institute of Applied Informatics in Civil Engineering, 57 p. (in German).

Gruber T. R. 1993. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In: Guarino N. & Poli R. (eds.): *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, Deventer, the Netherlands.

Hay D. C. & Barker R. 1996. *Data Model Patterns: Conventions of Thought*. Dorset House, NY, ISBN 0932633293, 268 p.

Hobbs J. R. 1995. Sketch of an Ontology Underlying the Way We Talk About the World. *International Journal of Human-Computer Studies*, 43(5-6), Academic Press Limited, Elsevier Computer Science, ISSN 1071-5819, pp. 819-830.

IAI 2000. *Industry Foundation Classes: IFC 2x, Technical Specification*. © IAI publ. Available from: http://www.iai-international.org

ISO/TC184/SC4/WG11/N088 1999. *EXPRESS-X Language Reference Manual,* 72 p.

Katranuschkov P. 2001. *A Mapping Language for Concurrent Engineering Processes*. Diss. Report, Fakultät Bauingenieurwesen, Schriftenreihe des Lehrstuhls für Computeranwendung im Bauwesen, Heft 1, TU Dresden, Germany, 372 p.

Katranuschkov P., Gehre A. & Eisfeld M. 2001. *Engineering Ontology – Formal Representation of the Data Structures*. Deliverable D5-2, EU Project IST-1999-11508 ISTforCE, 97 p.

Katranuschkov P. & Gehre A. 2002. Integrated Model Access Services for Human-Centred Engineering. In: Rezgui Y., Ingiride B. & Aouad G. (eds.) *Towards a European Knowledge Economy in the Construction and Related Sectors*, Proc. of the eSM@RT 2002 Conference, Salford, 18-21 November 2002, University of Salford, UK, ISBN 0902896415, pp. 44-54 (part B).

Liebich T. 2001. *ifcXML – XML Schema Language Binding of EXPRESS for IFC*. IAI Publ. MSG-01-001, 76 p. Available from: www.iai-international.org/iai_international/Technical_Documents/documentation/IFCXML

Liebich T. & Wix J. 2002. *Standards for Building Models: Analysis of Current Situation*. Deliverable D4.1.2, EU project IST-2001-32035 prodAEC, 60 p.

Lima C., Fies B., Zarli A., Bourdeau, M., Wetherill M. & Rezgui Y. 2002. Towards an IFC-enabled ontology for the Building and Construction Industry: the e-COGNOS approach. In: Rezgui Y., Ingiride B. & Aouad G. (eds.) *Towards a European Knowledge Economy in the Construction and Related Sectors*, Proc. of the eSM@RT 2002 Conference, Salford, 18-21 November 2002, University of Salford, UK, ISBN 0902896415, pp. 254-264 (part A).

Newell A. 1982. *The Knowledge Level*. Artificial Intelligence 18(1), pp. 87-127.

Ogden C. K. & Richards I. A. 1994. The Meaning of Meaning. In: Gordon W. T. (ed.) *C. K. Ogden and Linguistics*, Routledge/Thoemmes Press, London, 1994, ISBN 0415103533, (first published: 1923).

Ouksel A. & Ahmed I. 1999. Ontologies Are Not the Panacea in Data Integration: A Flexible Coordinator to Mediate Context Construction. *Journal of Distributed and Parallel Databases* 7(1), pp. 7–35.

Owen J. 1997. *STEP: An Introduction*. Second ed., Product Data Engineering Series, Information Geometers, Winchester, UK, 224 p.

Reddy M. P., Prasad B. E., Reddy P. G. & Gupta A. 1994. A Methodology for Integration of Heterogeneous Databases, *IEEE Transactions on Knowledge and Data Engineering* 6(6), pp. 920-933.

Spaccapietra S., Parent C. & Dupont Y. 1992. Model Independent Assertions for Integration of Heterogeneous Schemas. *VLDB Journal* 1(1), pp. 81-126.

Schodek D.L. 1992. *Structures*. Second ed. Prentice Hall, Englewood Cliffs, NJ, ISBN 0-13-855313-0, 557 p.

Schueller W. 1995. *The Design of Building Structures*. Prentice Hall, Englewood Cliffs, NJ, 868 p.

Turk Ž. 1998. Limits of Information Technology in Engineering (Or Why Computers Might Not Replace the Engineers). In: *Proceedings of the 4th Symposium on Intelligent Systems*, Croatian Systems Society, Zagreb, Croatia.

de Vries B. 1991. The Minimal Approach. In: Wagter H. & Spoonamore J. (eds.) *Proceedings of the CIB W78 workshop "The Computer Integrated Future"*, Eindhoven University of Technology, Calibre, The Netherlands, Sept. 16-17.

W3C 1997. *Web Interface Definition Language (WIDL)*.
Available from: http://www.w3.org/TR/NOTE-widl-970922

W3C 2001. *XML Schema (Part 0: Primer*, *Part 1: Structures*, *Part 2: Datatypes)*. W3C Recommendation.
Available from: http://www.w3.org/TR/2001/

Wix J. 1996. *Purpose of a Core Model*. Tech. Report, Research Project "Computerised Exchange of Information in Construction", Dept. of the Environment, UK, 12 p.
Available from: http://helios.bre.co.uk/ccit/info/bccm/purpose.htm

Woestenenk K. 1998. A Common Construction Vocabulary. In: Amor R. (ed.) *Product and Process Modelling in the Building Industry - Proceedings of ECPPM'98,* Building Research Establishment, Watford, 19-21 Oct., Clowes Group, Beccles, Suffolk, UK, pp. 561-568.

Woestenenk K. 2000. The LexiCon: An Update. In: Goncalves R., Steiger-Garcao A. & Scherer R. (eds.) *Product and Process Modelling in Building and Construction - Proceedings of ECPPM'2000,* Lisbon, 25-27 Sept., Balkema, Rotterdam, the Netherlands, pp. 263-266.

XMLSpy 2001. *XMLSpy v.3.5* © 1998-2001 Altova Inc. Newer version available from: http://www.xmlspy.com .