6th International Conference on Applied Informatics

Eger, Hungary, January 27–31, 2004.

Generalized Document Data Model for Integrating Autonomous Applications

Zsolt Hernáth, Zoltán Vincellér

Abstract

A document data model is a semi-structured data model, where any data are organized into ordered pairs as (R, S). R and S are called *raw data*, and *schema* respectively. Both of those are simple byte streams, but S, if given, is always an XML document or document fragment. The role of S is to describe the structuring of R which may occasionally give proper semantics as well. The advantage of such a data model is the easily structuring, decomposing, and occasionally restructuring the raw data component for the ever application that process it. This is important since any data may be processed by different applications, e.g. a particular application may be storing the raw data. Data in a document data model are generalized, if their schema components describe their raw data components' structuring for any given processing case. A generalized document data model is a document data model where all data are generalized and, in addition, there is a G schema interpreter which governs data, always with adequate structuring, through all processing phase including retrieval and storage.

In a generalized document data model principled logical database, document data modelled data instances are represented by their schema components rather than as ordered pairs. Such a database, beyond the hierarchies of real data instances, necessarily contains multiple levels of metadata, and also multiple levels of granularities of metadata and those together with the G schema interpreter enable heterogeneously structured real data being shared over different applications and application systems on the WEB. One way to easily integrate autonomous applications that follow using document data model may be automatically generating adequate generalized document data model in build-time from a formal data description, commonly used by the applications in question. Such a formal language is e.g. EXPRESS that is used in industrial automation systems and integration for product data representation and exchange, and defined by ISO 10303-11.

Introduction

Applied computer science and informatics by now plays a central role in our everyday life. The possibility of getting scientific, technical, business and quite general everyday on-line information accessing the World Wide Web database, together with having personal computers and Internet access possibilities at reasonable prices has changed our everyday life. In professional life, beyond information's access, processing and integrating information coming from different fields and from heterogeneous data sources is more important, so that, information processing needs integrated application systems, rather than standalone applications. Instead of integrating data and information coming from different data sources in order to achieve information processing actually needed, information industry vendors provide for various integrated application and information processing systems. By now, time is spent over standalone applications, the future and real needs are integrated application and information processing systems.

There is lots of already existing and efficiently used standalone applications over the same or different technical fields, and some of them are reasonable to be brought together with others. To do that, however, needs, among others, a common data access layer and data model which makes both data sources and processed results being shared and accessible over applications in question. In case of even well-architected applications or application systems, establishing both their common data model and the layer that gives access to data costs a relative large human and computer development resources since implementing such models needs bidirectional data transformations between the new common data model and the data model the applications actually use. Figure 1 below shows two standalone applications with heterogeneous data sources, Figure 2 and 3 present two versions of an integrated system consisting of the standalone applications on Figure 1. As it is seen, the integrated system on Figure 2 does not implement a unified common shared data model. The only thing that happened is a common I/O layer established which makes bidirectional mapping, and even

cross mapping between data source models and application data models. The integrated system on Figure 3 provides for a unified common data model and data source for the applications and a common unified I/O layer as well.

Integrating standalone applications or integrated application systems over heterogeneously stored and structured different data sources and processing results needs always data conversions mentioned above. The necessary knowledge to perform such conversions and data mapping is only partly application specific, yet applications has to have non-application specific knowledge, and even a significant part of those is hard coded (e.g. data retrieve and storage), that is, applications can not be seemingly made independent of data retrieval and storage environment. In contrast, we call applications autonomous, if they are independent of the all-time data access environment, including data source structuring, storage model, and data access engine. To design and implement autonomous applications, it is necessary that the non-application specific knowledge should not be the part of the applications. Concerning autonomous applications, one may think that if we have

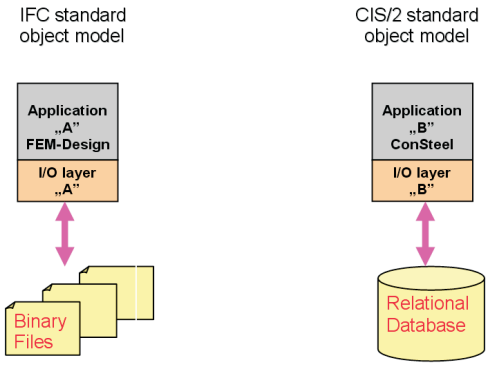

Figure 1

Integrated applications 1

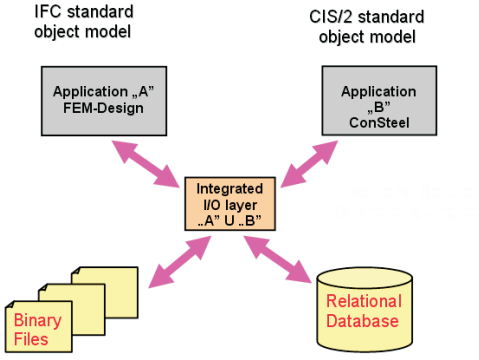

Figure 2

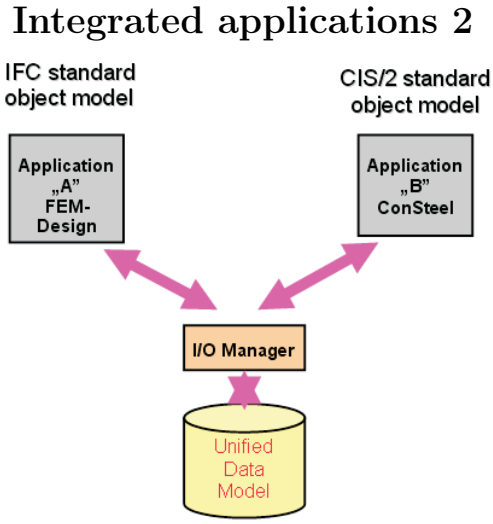

Figure 3

such an application, we can place it in any data environment where necessary data are present, independently of their all-time structuring. It sounds surprising and idealistic, but true. The question is where and how the necessary knowledge about the mapping between differently structured data is implemented. To find adequate answer to the questions above, we have to investigate applications or integrated application systems from a quite general perspective. Any applications in general consist of three steps: optional data retrieval, processing the data, and putting results out. From this perspective, themselves data may be quite natural media that can carry information about conversion rules between data source, storage model and application-specific structuring.

The Document Data Model – DDM

DDM is a data representation which can naturally code information and structuring knowledge (also out of scope of any application logic), and so it is one possible form favouring our need in achieving autonomous applications. DDM is a semi-structured data model, where information about structuring is separated from the real content of data.

Data in DDM are represented in forms of ordered pairs *(R, S)*. *R* and *S* are called *raw data* and *schema*, respectively. Both of those are simple byte streams, but *S*, if given, is always an XML document or document fragment [3]. The above organization of semi-structured data has two advantages: the same raw data can be

structured in several way, so that there might be a number of pairs where only the schema components are different, each of them providing for adequate structuring and occasionally additional semantics the all-time processing phase needs, so that they can be separately stored with storing only one instance of the raw data; the other is that the data representation above enables to define structured data types and primitive abstract operations on data.

For instance, data types and type templates in DDM are represented by pairs like T(Ø, S), where S is non-empty [4]. To instantiate such types or templates one should provide for the raw data component for such pairs. If the raw data component itself is a kind of schema, i.e. an XML document or XML document fragment, a real type is instantiated.

Pairs like V(R, Ø) where R is non-empty are classless (i.e. void) data. Classless data, if meaningful, can be converted to any type by providing for a schema component. One has to realize, that the schema components are not only for structuring the raw data. The schema components may describe or identify also methods characteristic for the structured type represented. Changing the schema components can be considered as a cast operation.

Last, pairs D(R, S), neither R nor S is empty, constitute typed data instances.

Up to now we have focused on the problem of implementing knowledge about mappings between data-sources/storage-model and application-specific structuring of data. The schema components of data in DDM, however, offer more general facilities for data processing. Indeed, in addition to describing structural mappings, they may contain specifications about data processing phases, the order of those, or in quite general, they may describe the whole life cycle of data within the applications. A datum in DDM is called generalized, if its schema component specifies all non-application-specific knowledge about processing it within an application.

Generalized Document Data Model – GDDM

A generalized document data model is a document data model where all data are generalized and, in addition, there is a G schema interpreter which governs data, always with adequate structuring, through all processing phase including retrieval and storage.

Using GDDM, knowledge about data processing is carried by data themselves, and the knowledge is interpreted by the G schema interpreter. To get and interpret such knowledge, the knowledge carrier media i.e. data are first to be accessed, but to access the data in question needs knowledge about where and how data is stored. That seems, however, a non-resolvable recursion, unless we state: in a generalized document data model principled storage environment or database, data instances

are represented by their schema components rather than as ordered pairs, or with other words autonomous applications' data access is access to a schema catalogue.

GDDM based data environments, beyond the hierarchies of real data instances, necessarily contains multiple levels of metadata, and also multiple levels of granularities of metadata and those together with the G schema interpreter enable heterogeneously structured real data being shared over different applications and application systems on the WEB. The only non-application-specific knowledge that autonomous applications have to have in GDDM based data environments is where and how to access the desired schema catalogue.

Any GDDM based standalone application may be considered as an integrated application system consisting of the application logic, the corresponding GDDM, and the data source and storage environment including the data provider and storage manager, which all three may also be considered as autonomous standalone applications.

GDDM vs. Integrated Application Development Environments

We are just in a getting started phase of investigating this approach. The objective is to provide an application independent mechanism of describing product data throughout their life cycle within applications. Before all we would like to outline that the model we introduced above is only a possible approach, only one way to supply developing applications that from the perspective of data access and data exchange can easily be integrated. Our model, however, does not offer tools for designing and implementing autonomous applications and GDDM based data environment, so that can not be considered as Autonomous Application Development Environment.

The standalone industrial applications on figure 1 – a FEM-Design and STEP-based IFC standard on the left hand side, and a CIS/2 standard Consteel application on the right hand side – are based on similar philosophy and methodology. The need of integrating those two industrial CAD applications has arisen two years ago, and was a real industrial need in a field of architecture in order to bring together iron-concrete and steel frames to plan and design mixed and complex architectural items. The two standalone applications have their own data models, data sources, and beyond the need of importing different kinds and portions of data from each other, both had the need of employing different computations and results on data being shared over the integrated system.

Integrating a set of standalone applications with a need of sharing data from their own data sources and data result sets need always some kind of common data

model and an integrated i/o component to access and store data. Any standalone application or integrated application system has to have a mapping between an application specific data or object model and a storage representation of the data in question. This mapping may be concentrated in a separate data access manager – the case of a well-architected and well-designed integration – or it is spread about over the application components. In both case, accessing the data is application specific, so that integrating already existing applications which were not developed in a similar or the same integrated development environment always needs and produces the change of a certain amount of existing program codes. The goal of establishing both standards CIS/2 and IFC intended to minimize the amount of existing program code to be changed by defining implementation independent product data exchange format, however, neither of them concerned with establishing self-based integrated application development environment.

The **CIMsteel Integration Standards** (CIS/2) is a standard for facilitating an integrated method of sharing and managing information within and between companies involved in the planning, design, analysis and construction of steel framed structures. Although the CIS is not a formal STEP (ISO 10303) standard, they make extensive use of STEP technologies, and it has been intended to develop the CIS into a full ISO standard (10303-230).

The **International Alliance for Interoperability** (IAI) was launched in September 1995 with the aim of promoting interoperability in the AEC/FM (Architectural, Engineering, Construction and Facilities Management) industries through the use of Industry Foundation Classes (IFC). The IAI intend to realize this vision by defining, promoting, and publishing IFC specifications for information sharing: throughout project life cycles, across disciplines, and between technical applications. The IAI makes use of a number of STEP technologies (particularly EXPRESS and EXPRESS-extensions) in the definition of IFC.
EXPRESS is a formal information requirements specification language, and it is an open industry standard for modelling data [1]. It consists of language elements that allow an unambiguous data definition and including constraint specifications on the defined data. It has textual and graphical notation, readable to humans and fully computer interpretable. The EXPRESS language is implementation independent and declarative, making it well suited for the definition of standardized data models. EXPRESS is used to define schemas (data models) through the definition of entity types and the allowed relationships among them, entities are collections of attributes, schemas are combined to form models, and a defined model can be instantiated to create populations of entity instances. The principle capabilities of EXPRESS language are provided by the next global scope declarations: data structure declarations (SCHEMA, ENTITY, TYPE, INTERFACE), constraint declaration (RULE), and miscellaneous declarations (FUNCTION, PROCEDURE), furthermore data types can be classified as: simple, aggregation, named, constructed and generalized data types.

EXPRESS-C was developed to extend and enhance the capabilities of EXPRESS by enabling the modelling of both static and dynamic properties of a domain.
EXPRESS-G (a graphical subset of EXPRESS) is a formal graphical notation for the display of data specifications defined in the EXPRESS language. It was created as a way to graphically describe the EXPRESS model to make it easier to understand. It supports only a subset of EXPRESS, the definitions of the entities and their attributes, and the relationships between them. It does not depict the constraints and constraint mechanisms provided by EXPRESS.
EXPRESS-M is a mapping definition language being developed to solve the problem of application protocol inter-operability in the STEP standard.
EXPRESS-X is defined as a formal language for specifying mapping of information that is modelled in the EXPRESS language. Generally there is some semantic equivalence but structural differences between the source schema and the target schema. The correctness of an EXPRESS-X schema will be checked by an EXPRESS-X compiler.

STEP (STandard for the Exchange of Product data) is targeted at the exchange of data describing a product between applications. STEP data can be exchanged and shared across international boundaries. This means that products designed in one country could be produced anywhere in the world. STEP is intended to deal with all types of products and all data related to those products. It also forces the integration of product data. Integration of data and the systems that deal with those data eliminates or at least severely reduces redundancy. The main objective of STEP is to provide information sharing technology through neutral mechanisms that are independent from any particular application, and allow the description of product data throughout the entire life-cycle of a product.

STEP and IFC (EXPRESS) based development toolkits and environments

There are two different kinds of binding implementations: early binding and late binding. The early binding is a static approach, in which the named components of the generated format correspond exactly to the named components of the data model. The late binding is a fully generic and dynamic approach, where the named components of the generated format correspond to the meta-model for the data model.

The **NIST STEP Class Library (SCL)** is a collection of C++ class libraries designed to be used as a starting point for building EXPRESS based applications. The software provides a dictionary of EXPRESS schema information and functionality representing and manipulation of EXPRESS objects. SCL includes components for generation of a C++ class representation based on SDAI C++ language binding, or objects based on the SDAI IDL language binding [2].

The **SBIKIT** toolkit is based upon the principles of early binding. Two basic developments have been prepared in order to facilitate STEP data exchange: A code generator called Expresso, and a C++ class library for access of data to a STEP file. Expresso is a stand-alone program which takes the data definition of an EXPRESS file producing C++ classes for each entity in the EXPRESS file's data model. In addition, a special C++ class is created for the data model as such. The Expresso generated code makes up the basis for the programmer's own code for a new application as it mainly consists of the classes he will build upon when writing the functionality of the program.

The **EDM (**EXPRESS Data Manager**)** database management system is a new technology to improve functionality, performance and quality of the implementation of data models defined in EXPRESS. It is an instance data storage, manipulation and validation according to the underlying EXPRESS definition, supports modelling, application development, database management, quality assurance, and data sharing using Standard Data Access Interface (SDAI). SDAI includes operations to manipulate instances of the EXPRESS entities in the application model. EDM has several XML supports, it can:

- generate EXPRESS schemas from XML models
- import XML data into EXPRESS data model
- export XML data from EXPRESS data model
- generate Query schema from XML model
- export Query results in XML.

EDM has an SDAI API interface, it provides a late binding for SDAI with extensions for database functionality, also has an EXPRESS compiler, it is used to instantiate a dictionary model from an EXPRESS schema. The EDM dictionary model can store any legal EXPRESS schema without loss of information.

The **ECCO** toolkit is an integrated development environment, which allows to check the syntactic and semantic correctness o fan EXPRESS data model and to translate the specification to an executable application or library that can be linked to an existing application.

Autonomous Application Development Environment

The architecture of an autonomous application that uses GDDM is very similar to that of a well-architected and well-designed standalone application or integrated application system. The main difference is that using GDDM data themselves

carry all necessary knowledge about structuring and processing phases, so the data access layer is independent of both storage and application data representation. Implementing autonomous applications and their GDDM is, however, very hard task, if necessary data types and possible processing phases of data instances have to be coded by hand using XML. Instead of following a today's fashionable and usual way, i.e., establishing a GDDM based integrated application environment with providing for a GDDM oriented new programming environment, we believe that traditional programming languages and their development environment extended by a computer aided implementation of a desired GDDM will result in much more satisfaction for the community of professional application developers.

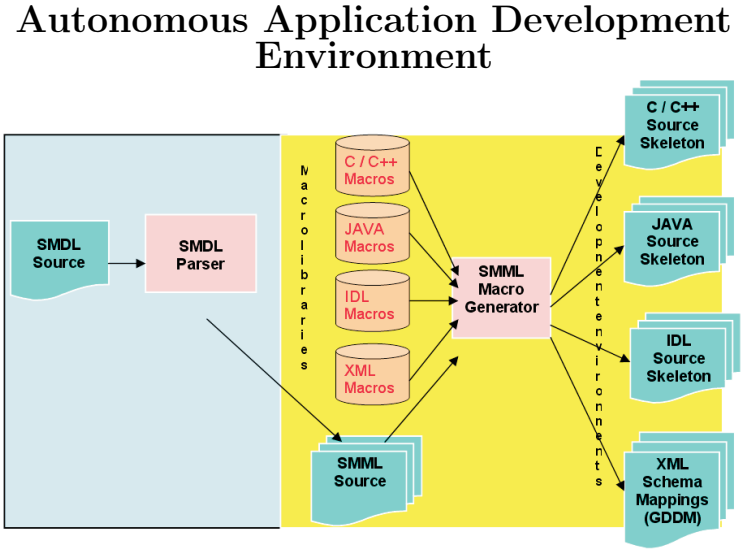

Figure 4: Autonomous Application Development Environment

In order to supply a computer aided implementation of a desired GDDM mapped to some desired program development environment, the following abstract autonomous application development environment is proposed (Figure 4):

- A high-level (EXPRESS-like) description language called SMDL (Schema Modelling Description Language) that is appropriate for describing data types and data populations and their life cycles within the application to be developed.

- A high-level macro language called SMML (Schema Modelling Macro Language) that is appropriate for describing any SDML sourced construction on a desired programming language and possible macro libraries for traditional programming languages.

- The SMDL parser that translates an SMDL sourced application schema into an SMML schema description.
- The SMML macro generator that from SMML schema descriptions generates source program skeletons together with the desired GDDM according to the all-time specified macro library.

References

[1] Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual Reference Number ISO 10303-11:1994, ISO, Switzerland 1994.

[2] Industrial automation systems and integration – Product data representation and exchange – Part 22: Implementation methods: Standard Data Access Interface specification, Reference Number ISO/DIS 10303-22, ISO, Switzerland 1993.

[3] Extensible Markup Language (XML) 1.0 (Third Edition)
W3C Proposed Edited Recommendation 30 October 2003
`http://www.w3.org/TR/2003/PER-xml-20031030`

[4] Data on the WEB – From Relations to Semistructured Data and XML, Serge Abiteboul, Peter Buneman, Dan Suciu
ISBN 1-55860-622-X, San Francisco 2000.