

A Generic Database Schema for CIDOC-CRM Data Management

Kai Jannaschk¹, Claas Anders Rathje¹, Bernhard Thalheim¹ and Frank Förster²

¹ Christian-Albrechts-University at Kiel, Information Systems Engineering
{kaja,car,thalheim}@is.informatik.uni-kiel.de

² Christian-Albrechts-University at Kiel, Institute of Classics
ffoerster@gshdl.uni-kiel.de
Kiel, Germany

Abstract. Database management currently assumes a relative stability of database structures and supports user viewpoints through a number of views. New users must either adopt their data demand to existing views or issue a request for support of their demands. The support for the second option is not feasible whenever user communities often change or users agile change their viewpoints. Further applications continuously evolve. This evolution also results in change requests to database structures, to database views and to database features. We support evolution of databases and changes of user viewpoints by a generic database system, and automatic generation of generation of views. This approach supports modification of the database according to the special viewpoint of users. Queries issued by the user are transformed to generic database queries. Results that are computed for these queries are translated back to user-specific answers.

Key words: knowledge model, semantic data analytics, exploratory semantic searching and browsing, database modeling, scientific data management

1 Motivating Applications

1.1 CIDOC-CRM

The International Committee on Documentation of the International Council of Museums (ICOM-CIDOC) focuses on the documentation requirements and standards of museums, archives, and similar organisations. The CIDOC Conceptual Reference Model (CRM) [4] provides an extensible ontology for concepts and information in cultural heritage and museum documentation. It became an international standard (ISO 21127:2006) for the controlled exchange of cultural heritage information.

The goal of CIDOC-CRM is to provide the semantic definitions and clarifications needed to transform disparate, localized information sources into a coherent global resource, be it within an institution, an intranet, or on the Internet. In

order to achieve this, the CRM adopts a supra-institutional perspective, abstracted from any particular local context. This abstraction is derived from the underlying semantics of the database schemata and document structures found in museum and cultural heritage documentation. The CRM is descriptive rather than prescriptive: it explains the logic of what cultural heritage institutions do in fact document rather than telling them what they should document. Thus it enables semantic interoperability.

This CIDOC-CRM model is a typical example of a pixel schema. Pixel schemata are simple typed, heterogeneous, agile evolving and large schemata with relatively small populations. A diagram [3] displays one approach of its main structure without providing an insight into the structure of each of more than 240 entity and relationship types. Its printout in the A0 format results in a pixel picture that is not surveyable.

An example for a CIDOC-CRM conform description of an object is given at a work [7] by Doerr. In these example the subject is an Epitaphios shown in the Museum Benaki. In Figure 1 we give an overview about the (shortened) main structure of that object Epitaphios. Each node containing $E[0-9]^*$ in the label like $E19.Physical_Object$ defines a type in the CIDOC-CRM standard. The labels on the edges define relationships between entities.

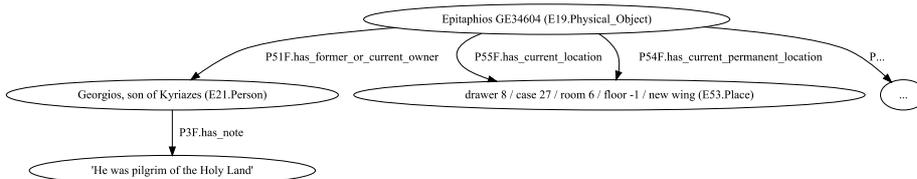


Fig. 1. Main structure of CIDOC-CRM structure for Epitaphios

1.2 Related Work

Approaches for storage, retrieval, and description of data with heterogeneous and evolving types exist, e.g. Amazon's SimpleDB. The data of different users are wrapped and stored in an on-line database. A user of Amazon's SimpleDB can define his/her own data structure. Access to the data is given by predefined APIs. The Resource Description Framework (RDF) from W3C is described e.g. by Klyne's work [9]. An RDF expression typically connects two resources by a predicate. Each subject or object can be a RDF expression by itself, and a resource is a link for a source or a literal, that contains a value. The relationships between a subject and an object are limited to binary relationships. But it is not possible to decompose all relationships into binary relationships without the loss of information as shown in [5] by Date et al. West gives some hints for a good modelling of a stable, but flexible to changing business practices, database

schema, that can be shared by different software products in his work [14]. But these principles are based on the knowledge, which data types are needed by the user within the model for all cooperating software products at the time of developing the database schema. Different normal forms are defined for the developing of database structures. Fagin defined in his work [8] a domain/key normal form (DK/NF). A relation schema in first normal form is in the DK/NF, if each constraint can be inferred from key and domain. Date presents in his work [6] a definition for the 6th normal form (6NF). A relation type fulfills the 6NF, if it consists of a single key, plus at least one additional attribute. A normalisation of a relational database model to the 6NF results in a huge set of entities with lots of different relationships between them. We call such a model in the graphical representation a pixel schema.

1.3 Outline of the Paper

In this paper we define a generic database schema for CIDOC-CRM data management. We start in Section 2 by describing the theoretical background for our generic databases. We characterise the binarisation of entity and relationship types and the generic functions. Furthermore, we present the bulk composition type as a data exchange format between a database and an application and we define generic views for import and export. In Section 3 we present a practical solution for our ideas. This includes an architecture for a system, the presentation of our generic database schema in the (H)ERM notation and the depending relational realisation, and the explanation of user query transformation on a generic database. At last, we summarise in Section 4 the work and give an outlook for future research.

2 Generic Database Modelling

In following subsections we define the background of our approach towards a generic database for CIDOC-CRM conform data management. We specify the restrictions and the challenges of the application domain, define the generic functions and the bulk type for the communication between a database and an application. At last we present the views for the im- and export of data.

2.1 Problems to be Solved for Pixel Schemata

Pixel schemata are challenging current technology due to their evolution. At the same time they can be handled by everybody who knows basics of database technology. However, they are a nightmare for everybody who wants to share or integrate data or who strives for developing interfaces for data import or export. We need a *sophisticated programming support* for such agile evolution.

A rather simple but heterogeneous type structure is used by pixel schemata. Each type typically has less than four attributes. We may *combine* these types by horizontal denormalisation as given by Thalheim in his work [13].

Pixel schemata contain their own metadata within the naming structure and thus hint to its developers and the kind of types used. Thus, we need an *explicit reference to the source and to the kind of type*.

Databases for pixel schemata are populated at different periods of time in a different form. The quality of the data depends on the import of the data to the database. Data may be cleansed by other people. Therefore, we should use temporal data for *explicit time reference*.

The following restrictions observed for the application domain are essential for our approach.

Simple heterogeneous application types: There are a lot of different entity and relationship types in the application domain, that are defined with a relatively small set of attributes.

Evolving application types: The practical sciences produce a rapidly evolving amount of different application types. New types are created, older types are changed or will be never used again.

Simple relationship constraints: The cardinality constraints between two or more entities are (0,m) or (1,n).

Simple functional dependencies: The functional dependencies in the entity and relationship types are simple. That means, entity types fulfill the DK/NF.

Form-oriented input and output: The input data for the database and the output data requested by users is gathered in defined XML or HTML forms.

Small query result sets: A result set for a user query is very small.

2.2 Binarisation of Relations.

It is well known in mathematics that the language of binary predicates will be universal if identifiers are permitted. Binarisation is also the basis for the triple format (Type, Attribute, Value) that is widely used in Computer Engineering. For instance, the job control language JCL uses such triples for universal commands. In a similar form, RDF is based on such triples.

Consider a predicate in the topic of the storage of an archaeological find $Storage(Place, Building, Room)$. This can be enhanced to a predicate with an identifier-backed predicate $Storage(ID, Place, Building, Room)$. The ID-enhanced predicate can be replaced by a set of three predicates $Storage_Place(ID, Place)$, $Storage_Building(ID, Building)$, and $Storage_Room(ID, Room)$.

These predicates carry the same information if additional integrity constraints are applied:

$$Storage_Place[ID] \subseteq \supseteq Storage_Building[ID] \subseteq \supseteq Storage_Room[ID]$$

If this set of paired inclusion constraints is maintained in the database then the two databases become equivalent. We call this set of constraints for a decomposed type *unit constraints*.

A second kind of constraints is necessary for maintenance of identifiers for enhanced types. We may require that identifier sets of enhanced predicates are disjoint. We call these constraints *exclusivity constraints*.

Therefore, we may use binary predicates as long as we are able to maintain unit and exclusivity constraints. This integrity constraint enforcement may be transferred to database functions that offer the unique opportunity for database changes.

2.3 Generic Functions.

Bienemann et al. introduce in their work [1] the conception of generic functions and generic workflows. Similar to situations in applications, tasks may be specified on the basis of a general description of a possible way for satisfaction and completion. In traditional function and workflow development approaches, a task is supported by its own workflow and functions. The traditional approach will lead to a huge number of processes that have a similar behaviour.

Instead of these classical approaches we use a different way. We associate a task with a generic workflow and generic functions that accommodate all possible different ways of completing the task. The generic workflow may be unfolded to a workflow suite by consideration of the context.

The formal definition of generic functions uses function applications and functionalisations. Consider types $t_1, \dots, t_k, t'_1, \dots, t'_n$ for $k, n \in \mathbb{N}$. Consider $Dom := Col^{t_1} \times \dots \times Col^{t_k}$ and $Rng := Col^{t'_1} \times \dots \times Col^{t'_n}$ to be sets of tuples of collections of objects of the corresponding types, for $k, n \in \mathbb{N}$. Consider a function $f : Dom \rightarrow Rng$ mapping a tuple of collections of types t_1, \dots, t_k into a tuple of collections of types t'_1, \dots, t'_n . Consider two formulae ϕ, ψ defined over the domain and the co-domain of f , respectively. The quintuple $(Dom, \phi, f, \psi, Rng)$ is called a *function application* with precondition ϕ and postcondition ψ [2]. Function applications can be defined recursively by means of operators of the function algebra of choice, i.e., $(Dom, \phi, \theta(F_1, \dots, F_m), \psi, Rng)$ is a function application, if θ is a corresponding operator of the function algebra, and F_1, \dots, F_m are function applications.

A *functionalisation* is a quadruple $(\mathcal{S}, \mathcal{F}, \Sigma, s_0)$ with

1. a specification of *structuring* denoted by $\mathcal{S} = (S, V)$ and consisting of a *database schema* $S = (T^S, \Sigma^S)$ with T^S being a set of types according to the type system as stated above, and Σ^S being a set of static integrity constraints, and a set of views V upon schema S defining collections in domains, and co-domains of function applications,
2. a function application \mathcal{F} ,
3. a set of dynamic integrity constraints Σ on \mathcal{S} ,
4. a distinguished initial state s_0 .

Dimensions of functionalisation are *refinement of structuring*, *context embedding*, and *instantiation*. Any function may be adapted to the specific needs by application of one of these refinement strategies.

2.4 The Bulk Composition.

A bulk composition builds a unified container for instantiations of different types and is used as a transfer type between different schemes or applications. A de-

scription of a bulk composition is given in a work [10] from Ma et al.. The bulk composition type BC is a triple (t, ID, l) with

1. t as a name of a type,
2. an identifier ID for a type instantiation,
3. and a set of attributes $l = \{A_1, A_2, \dots\}$.

Look at the given example $Storage(ID, Place, Building, Room)$. We identify the type $t : Storage$, with the given identifier $id : ID$ and the set of attributes $l = \{Place, Building, Room\}$. A tuple $bc = (Storage, ID, \{Place, Building, Room\})$ is the result.

The bulk composition type is used as a uniform type and can contain any instantiation of different types. Each type has its own attribute list.

2.5 Generic Views for Import and Export of Data.

In database design one has to specify the requirements for data. The result is a conceptual data model with entities, attributes, relationships, and integrity constraints. This conceptual model is the basis for a logical data model with tables, columns, keys, and triggers of a database. Refinement of the conceptual model results in a refinement of the logical model. We can use views to define entities and relationships based on a static relational database model.

Terwilliger describes in his thesis [12] a set of transformations for the transformation process from one database schema S to another database schema S' . In our case the two transformations $UNPIVOT$ and $PIVOT$ are of special interest. The transformation $UNPIVOT$ translates instances of a type T into a key-attribute-value set, and the transformation $PIVOT$ translates a key-attribute-value set into instances of a type T .

We adopt these transformations and define a set of generic views for import of CIDOC-CRM conform data \mathcal{VI} . The views are based on a static relation schema of the database. We distinguish between views for importing entities and views for importing relationships. A view for importing entities VI_e is defined as a triple $(type(VI_e), id(VI_e), attr(VI_e))$, where VI_e is a view name for importing entities, $type(VI_e)$ means the type of an entity, $attr(VI_e)$ is a set of attributes and $id(VI_e)$ is a non-empty subset of $attr(VI_e)$ called identifier of the entity. We define a view for importing relationships VI_r as a triple $(type(VI_r), ent(VI_r), attr(VI_r))$, where VI_r is a view name for importing relationships, $type(VI_r)$ means the type of a relationship type, $attr(VI_r)$ is a set of attributes and $ent(VI_r)$ is a non-empty sequence of entity types. Each entity type plays its own defined role in a relationship type.

Similar to the definition of a generic view for import \mathcal{VI} we specify a generic view for export of CIDOC-CRM conform data \mathcal{VE} .

3 Realisation of Generic Databases

In this section we present the architecture of an actual implementation. Furthermore, we show our generic database schema in the (H)ERM notation and the resulting relational realisation and query translations for searching the database.

3.1 The V-Architecture Model

To handle objects described by the CIDOC-CRM standard, we developed a system, which is based on a generic relational database schema. This architecture is given in Figure 2. We use one function as an interface for data import, and one function as an interface to receive a user query and build an answering result set. Each interface has access to the set of views for import or export, which capsule the relations of the database. The function scans the given request and chooses the right views for answering. These views are not physically implemented in the database. The view for import is generated by a generic function, that also implements the *UNPIVOT* transformation. The simple views for export are generated by a generic function. Furthermore, we can specify more complex views for export by a physical definition in the DBMS. Such physical views are needed in order to map different type hierarchies, and they are only readable. The structure of attributes of these views is the same as in the underlying physical relations. The physical views are listed in the system tables of a DBMS. Both export views capsule the transformation *PIVOT*.

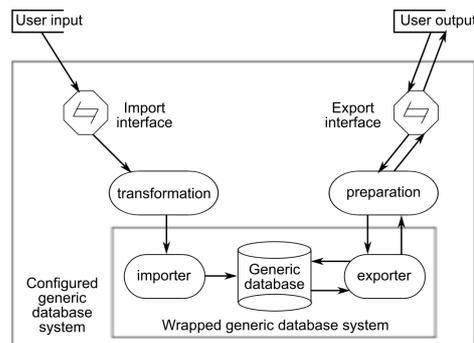


Fig. 2. The V-architecture for the generic database system

This architecture for the storing and searching of CIDOC-CRM conform data can be enhanced by a system for the type definition and the type management as shown by Rathje in his thesis [11].

3.2 Generic ER-Schemata and Relational Realisation

The generic views for import and export of CIDOC-CRM conform data and the generic functions are based on a generic entity-relationship schema for CIDOC-CRM conform data. The (H)ER schema is given in Figure 3 in the (H)ERM notion. We give an explanation of the schema after that.

The "*Generic Entity Type*"(GET) is used for the storage of the instantiations of types. The attribute "*Entity Type*"(E_Type) stores the given view type *type*(VI_e) for entities. The "*Generic Entity Type Attribute*"(GETA) is used for

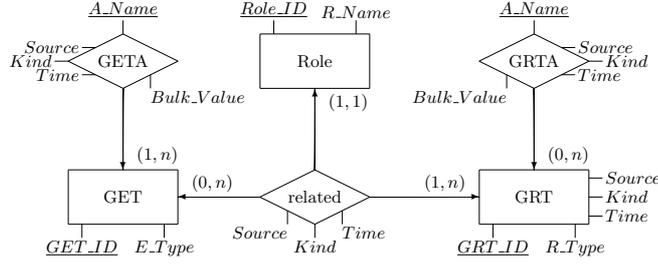


Fig. 3. Generic database schema in (H)ERM notion

the storage of the attribute values of instantiations of types. The attribute "Attribute Name" (*A_Name*) marks one of the attributes of the view type for the entities *attr(VI)*. The attribute "Bulk Value" (*Bulk_Value*) stores the value of the attribute. Each instantiation of a generic entity type needs minimum one instantiation of a generic entity type attribute, the identifier of an entity. The "Generic Relationship Type" (*GRT*) is used for the storage of the instantiations of relationships. The attribute "Relationship Type" (*R_Type*) stores the given view type *type(VI_r)* for relationships. Similar to the generic entity type attribute, there is a "Generic Relationship Type Attribute" (*GRTA*) for storing possible attributes of the view for the relationships *attr(VI)*. In a relationship, entities are referenced by a "Role" (*Role*). The attribute "Role Name" (*R_Name*) characterises the role of an instantiation of an entity in an instantiation of a relationship. The role, an entity and a relationship are related in the relationship type "related". In each instantiation of the type "related", a role identifies the instantiation of an entity in an instantiation of an relationship.

The three meta attributes "Source", "Time", and "Kind" are used to store the source information, a timestamp of the creation or update of an information, and a state of completion.

We use for declaration of the relational schema a table representation where columns are given by the name and the value domain. We assume the existence of a most general value domain, e.g., varchar.

The relational type for the "Generic Entity Type" is stores tuples of entities of an (H)ER model.

Generic_	<u>GET_ID</u>	E_Type
Entity_Type	ID_Type	varchar

The relational type for the "Generic Entity Type Attribute" stores tuples of entity attributes of an (H)ER model.

Generic_Entity_	<u>GET_ID</u>	<u>A_Name</u>	Bulk_Value	Kind	Time	Source
Type_Attribute	ID_Type	varchar	varchar	Ref_Type	timestamp	Ref_Type

In the relational type "Generic Relationship Type" the relationship types of an (H)ER model are mapped.

Generic_Relationship_Type	GRT_ID ID_Type	R_Type varchar	Kind Ref_Type	Time timestamp	Source Ref_Type
...

In the case of attributes on a relationship type of an (H)ER model we introduce the relational type for the "Generic Relationship Type Attributes".

Generic_Relationship_Type_Attribute	GRT_ID ID_Type	A_Name varchar	Bulk_Value varchar	Kind Ref_Type	Time timestamp	Source Ref_Type
...

The relationship between entities of "Generic Entity Type" and "Generic Relationship Type" are mapped to the relational type of "Related By Role".

Related_By_Role	Role_ID ID_Type	GET_ID ID_Type	GRT_ID ID_Type	R_Name varchar	Kind Ref_Type	Time timestamp	Source Ref_Type
...

3.3 Query Transformation

Users create queries about the conceptual model. These queries have to be transformed into queries about the relational model. In our case we have a simple relational model, so the query has to be transformed into a query for the view model. The views transform a query into a query on the underlying relational model. The result of a query will be transformed into a tuple of the bulk composition. Queries on the presented generic database schema are possible on the elements of the bulk composition type. The following search conditions are possible:

- *t*: search for instantiations of entities or relationships by a given type
- *ID*: search for instantiations of entities by a given set of identifiers, or for instantiations of relationships by a given sequence of identifying entities
- *l*: search for instantiations of entities or relationships by defining a set of attributes

A combination of the three elements as a search condition is achievable.

4 Conclusion

4.1 Achievements of our Approach

The main trick of this approach is the direct data storage of means used in the data dictionary instead of maintaining a data dictionary. This approach is somehow similar to the approach used for Amazon's Simple DB and generalizes the RDF approach in a way that is easier to maintain. Our approach is not useful for data-intensive science, but for easily manageable research projects with mostly stringent scientific objectives represented by the self-defined single views. Therefore, it supports clearly outlined, agile evolving, and even large schemata while storing heterogeneous data. The integration of new simple entity and relationship types is very easy because no new relations are needed in the DBMS. The requirements for entity and relationship types are a small amount of attributes, the functional dependencies in the types are reduced to the DK/NF. Furthermore, we can not specify complex cardinality constraints, and we do not expect mass updates or huge user queries. Only very complex views require a physical redefinition in the database at the moment.

4.2 Research Agenda

One open aspect in this paper is the definition of functions for updating or deleting entities and relationships.

Furthermore, the CIDOC-CRM ontology contains a hierarchy of entity types. The views for export need an explicit definition to consider these hierarchies in user queries because the generic database is used for the storage of instantiations of entities and relationships. Otherwise, we have a system for the definition of forms for data input and output. We are working on integrating these hierarchy definitions and user queries.

Furthermore, we will expand the query transformation to answer logical connections in the search condition. At last, we need performance measures to avoid bottlenecks in the usage of the system, if the amount of data is increasing.

References

1. A. Bienemann, K.-D. Schewe, and B. Thalheim: Towards a theory of genericity based on government and binding. In *Proc. ER'06, LNCS 4215*, pages 311–324. Springer (2006)
2. A. Bienemann: A generative approach to functionality of interactive information systems. PhD thesis, CAU Kiel, Dept. of Computer Science (2008)
3. CIDOC CRM Special Interest Group: http://www.cidoc-crm.org/docs/AppendixA_DiagramV9.pdf (2004)
4. N. Crofts, M. Doerr, T. Gill, S. Stead, and M. Stiff: Definition of the CIDOC Conceptual Reference Model. ICOM/CIDOC CRM Special Interest Group (2010)
5. C. J. Date, H. Darwen, and N.A. Lorentzos: *Temporal Data and the Relational Model*. Morgan Kaufmann Publishers, USA (2003)
6. C. J. Date: *Database in Depth: Relational Theory for Practitioners*. O'Reilly Media, Inc. (2005)
7. M. Doerr and I. Dionissiadou: Data Example of the CIDOC Reference Model - Epitaphios GE34604 -. ICOM/CIDOC CRM Special Interest Group (1998)
8. R. Fagin: A normal form for relational databases that is based on domains and keys. *ACM Trans. Database Syst.*, New York, NY, USA (1981)
9. G. Klyne and J. J. Carroll: *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C (2004)
10. Hiu Ma, R. Noack, K.-D. Schewe, and B. Thalheim: Using meta-structures in database design. *Informatica*, 34:387403 (2010)
11. C.A. Rathje: *Kompakte Datenbankschemata mit Sichtensuits*. Bachelor Thesis, CAU Kiel, Kiel (2010)
12. J. F. Terwilliger: *Graphical User Interfaces As Updatable Views*. Dissertation, Portland State University (2009)
13. B. Thalheim: *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin (2000)
14. M. West: *Developing High Quality Data Models*. The European Process Industries STEP Technical Liaison Executive (EPISTLE), London (1999)