Extended Entity-Relationship Model

Bernhard Thalheim
Christian-Albrechts University Kiel, http://www.informatik.uni-kiel.de/∼thalheim

SYNONYMS

EERM, HERM; higher-order entity-relationship model; hierarchical entity-relationship model

DEFINITION

The extended entity-relationship (EER) model is a language for definition of structuring (and functionality) of database or information systems. It uses inductive development of structuring. Basic attributes are assigned to base data types. Complex attributes can be constructed by applying constructors such as tuple, list or set constructors to attributes that have already been constructed. Entity types conceptualise structuring of things of reality through attributes. Cluster types allow the generalise or to combine types into singleton types. Relationship types associate types that have already been constructed into an association type. The types may be restricted by integrity constraints and by specification of identification of objects defined on the corresponding type. Typical integrity constraint of the extended entity-relationship model are participation, lookup and general cardinality constraints. Entity, cluster and relationship classes contain a finite set of objects defined on these types. The types of an EER schema are typically depicted by an EER diagram.

HISTORICAL BACKGROUND

The entity-relationship model has been introduced by P.P. Chen in 1976 [1] as a generalisation of the network model formalisation of C. Bachmann. The model conceptualises and graphically represents structuring of the relational model and is currently used as the main conceptual model. A large number of extensions to this model has been proposed in the 80ies and 90ies due to its extensive usage. Cardinality constraints [1, 3, 4, 8] are the most important generalisation of relational database constraints [7]. These proposals have been evaluated, integrated or explicitly neglected in an intensive research discussion. The semantical foundation proposed in [2, 5, 8] and the various generalisations and extensions of the entity-relationship model have led to the introduction of the higher-order or hierarchical entity-relationship model [8] which integrates most of the extensions and also supports conceptualisation of functionality, distribution [9] and interactivity [6] for information systems. Class diagrams of the UML standard are a special variant of extended entity-relationship models.
The ER conferences (annually; since 1996: International Conference on Conceptual Modeling, http://www.conceptualmodeling.org/) are the main forum for conceptual models and modelling.

SCIENTIFIC FUNDAMENTALS

The extended entity-relationship model is mainly used as a language for conceptualisation of the structure of an information systems applications. Conceptualisation of database or information systems aims in a representation of the logical and physical structure of an information system in a given database management system (or for a database paradigm), so that it contains all the information required by the user and required for the efficient behavior of the whole information system for all users. Furthermore, conceptualisation may target to specify the database application processes and the user interaction. Description of structuring is currently the main use of the extended ER model. The diagram representation of EER schemata uses rectangles and diamonds for the entity and relationship types.

Brief description of structures of the extended ER model.
The extended entity-relationship model uses a data type system for its attribute types, allows to construct entity types $E \stackrel{\circ}{=} (attr(E), \Sigma_E)$ through a set of attributes, and inductively builds relationship types $R \stackrel{\circ}{=} (T_1, ..., T_n, attr(R), \Sigma_R)$ of order i ($i \geq 1$) through a set of (labelled) entity types and of relationship types of order less than i and a set of attribute types. Additionally, cluster types $C \stackrel{\circ}{=} T_1 \dot{\cup} ... \dot{\cup} T_n$ of order i can be defined through a disjoint union $\dot{\cup}$ of relationship types of order less than i or of entity types. Entity/relationship/cluster classes contain a set of tuples of the entity/relationship/cluster types. The data type system is typically inductively constructed on a base type B system by application of constructors such as the tuple constructor (..), set constructor {..} and the list constructor $< .. >$. Types may be optional [..]. The types T can be labelled $l : T$. The types T are typically extended by a set Σ_T of constraints which are defined for components of the type. Only those classes are considered for which the constraints of their types are valid. An entity-relationship schema consists of a set of data, entity, relationship and cluster types which types are inductively built on the basis of the base types.
Given a base type B system. The types of the ER schema are defined through the type equation:
$T \quad = \quad B \mid (l_1 : T, ..., l_n : T) \mid \{T\} \mid <T> \mid [T] \mid T \dot{\cup} T \mid l : T \mid N \stackrel{\circ}{=} T$
A set of entity, cluster and relationship types can be represented by ER diagrams. Entity types are represented graphically by rectangles. Attribute types are associated with the corresponding type. Attributes identifying a type are underlined. Relationship types are represented graphically by diamonds and associated by arcs to their components. Clusters are represented by diamonds labelled with root illustrated $\oplus$ or simply as a common input point labelled by $\oplus$ to a diamond.

Structures of the extended ER model in detail.
We use a classical four-layered approach to inductive specification of database structures. The first layer is the data environment, called the basic data type scheme, which is defined by the system or is the assumed set of available basic data. The second layer is the schema of a given database. The third layer is the database itself representing a state of the application's data often called micro-data. The fourth layer consists of the macro-data that are generated from the micro-data by application of view queries to the micro-data.
We use Figure 1 for illustration of the concepts of the extended ER model.

Attribute types and attribute values of the extended ER model.
The classical ER model uses basic (first normal form) attributes. Given a a base type B system BT and type constructors such as the tuple constructor (..), set constructor {..} and the list constructor $< .. >$. These type constructors can be applied for construction of more complex attribute types over the base type system BT.
Given a set of names $\mathcal{N}$. A basic attribute (type) $A :: T$ is given by an (attribute) name $A \in \mathcal{N}$ and a base type B. The base type B is often called the domain of A, i.e. $dom(A) = B$. Complex attributes are constructed on base attributes by application of the type constructors. We may extend this notion of a domain to complex attributes, i.e. the domain of the complex attribute A is given by $dom(A)$. Components of complex attributes may be optional, e.g. the *NameOfBirth* in the attribute *Name*.
Typical examples of complex and basic attributes in Figure 1 are

Name $\stackrel{\circ}{=}$ *(FirstNames <FirstName>, FamName, [NameOfBirth,], Title)*
Title $\stackrel{\circ}{=}$ {*AcadTitle*} $\dot{\cup}$ *FamTitle*
Contact $\stackrel{\circ}{=}$ *(Phone({PhoneAtWork}, private), email, ...)*
PostalAddress $\stackrel{\circ}{=}$ *(Zip, City, Street, HouseNumber)*
for *DateOfBirth :: date*, *AcadTitel :: titleType* , *Zip :: String7*,
City :: VarString, *Street :: VarString*, *HouseNumber :: SmallInt*.

The complex attribute *Name* is structured into a sequence of first names, a family name, an optional component for the name of birth and a complex attribute for titles. Titles consist of a number of academic titles and potentially one family title that can be distinguished from all academic titles.
We may also shorten the notion by contracting the definition, e.g. instead of the specification above we write
PostalAddress (Zip, City, Street, HouseNumber) .

Entity types and entity classes of the extended ER model.
Entity types are characterized by their attributes. Entity types have a subset of the set of attributes which serve

to identify the elements of the class of the type. This concept is similar to the concept of key known for relational databases. Additionally, entity types E can also be characterised by set Σ_E of integrity constraints. We may omit trivial elements of the definition. Identifying attributes may be underlined instead of explicit specification.
Formally, an entity type is given by a name E, a set of attributes $attr(E)$, a subset $id(E)$ of $attr(E)$, and a set Σ_E of integrity constraints, i.e.

$E \quad \hat{=} \quad (attr(E),\ id(E),\ \Sigma_E)$.

The following types are examples of entity types in Figure 1:

Person $\hat{=}$ *({ Name, Login, URL, Address, Contact, DateOfBirth, PersNo* $\hat{=}$ *EmplNo* $\dot{\cup}$ *... , ... })*
Course $\hat{=}$ *({ CourseID, Title, URL } , { CourseID }),*
Room $\hat{=}$ *({Building, Number, Capacity } , {Building, Number }),*
Semester $\hat{=}$ *({ Term, Date(Starts, Ends) }, { Term }).*

An ER schema may use the same attribute name with different entity types. For instance, the attribute *URL* in Figure 1 is used for characterising additional information for the type *Person* and the type *Course*. If we want to distinguish them then we can also use complex names such as *CourseURL* and *PersonURL*.
Objects on type E are tuples with the components specified by a type. For instance, the object (or *entity*)

(HRS3, 408A, 15)

represents data on a seminar room in a university.
An entity class E^C of type E consists of a finite *set* of objects on type E for which $id(E)$ is a key and the set Σ_E of integrity constraints is valid.

Cluster types and cluster classes of the extended ER model.

A disjoint union $\dot{\cup}$ of types whose identification type is domain compatible is called a cluster. We restrict the union operation to disjoint unions since we need to preserve identification. Furthermore, we require that the identification types of the components of a cluster are domain-compatible. Cluster types can be considered as a generalisation of their component types.
A cluster type (or "category")

$C \hat{=} l_1 : R_1 \dot{\cup} l_2 : R_2 \dot{\cup} ... \dot{\cup} l_k : R_k$

is the (labelled) disjoint union of types $R_1, ..., R_k$. Labels can be omitted if the types can be distinguished.
An example of a cluster type is the following type

JuristicalPerson $\hat{=}$ *Person* $\dot{\cup}$ *Company* $\dot{\cup}$ *Association* .

The cluster class C^C is the 'disjoint' union of the sets $R_1^C, ..., R_k^C$. It is defined if $R_1^C, ...R_k^C$ are disjoint on their identification subtypes. If the sets $R_1^C, ..., R_k^C$ are not disjoint then we can use the labels for differentiating the objects of clusters. In this case, an object uses a pair representation (l_i, o_i) for objects o_i from R_i^C.

Relationship types and relationship classes of the extended ER model.

First-order relationship types are defined as associations between single entity types or clusters of entity types. They can also be characterized by attributes. The relationship type of order i is defined as an association of relationship types of order less than i or of entity types and can also be characterized by attributes.
Formally, a relationship type is given by a name R, a set $compon(R)$ of labelled components a set of attributes $attr(R)$, a subset $id(R)$ of $compon(R) \cup attr(R)$, and a set Σ_R of integrity constraints, i.e.

$R \quad \hat{=} \quad (compon(R),\ attr(R),\ id(R),\ \Sigma_R)$.

Component types that are not used for identification within the relationship type can be optional. We may use a specific extension for translation of optional components. For instance, in Figure 1 we assume that the room is inherited to *PlannedCourse* from *ProposedCourse* if this component is optional.
The extended ER model uses layering of relationship types. In this case, cyclic structuring can be avoided. A relationship type of first order has only entity types for the component types. The components of a relationship type of order i are either entity types or relationship types of order less than i or cluster types which components are of order less than i or entity types.
It is often assumed that the identification of relationship types is defined exclusively through their component types. Relationship types that have only one component type are unary types. These relationship types define subtypes. If we want to explicitly represent subtypes then we also can use binary relationship types named by *IsA* between the subtype and the supertype. For instance, the type *Professor* in Figure 1 is a subtype of the type *Person*.

Higher-order types allow a convenient description of classes that are based on other classes. Let us consider a course planning application in Figure 1. Lectures are courses given by a professor within a semester and for a number of programs. Proposed courses extend courses by description of who has made the proposal, who is responsible for the course, which room is requested and which time proposals and restrictions are made. Planing of courses assigns a room to a course that has been proposed and assigns a time frame for scheduling. The kind of the course may be changed. Courses that are held are based on courses planned. The room may be changed for a course. We use the following types for the specification:

ProposedCourse $\triangleq$ *(*Teacher*: Professor,* Course*, Proposal : Kind, Request : Room,*
Semester, Set2 : { *Program* }*, Responsible4Course: Person,*
InsertedBy : Person, { *Time(Proposal, SideCondition)* }*),*

PlannedCourse $\triangleq$ *(ProposedCourse, [Kind] , [Room],* { *TimeFrame,* TermCourseID }*),*

CourseHeld $\triangleq$ *(*PlannedCourse*, [Room],* { *StartDate, EndDate, AssistetBy* }*).*

The last two types use optional components in the case that a proposal or a planning of rooms or kinds is changed. Typically, planned courses are identified by their own term specific identification. We omitted integrity constraints until they have been defined.

An object (or a "relationship") on the relationship type $R \triangleq (R_1, ..., R_n, \{B_1, ..., B_k\},\ id(R),\ \Sigma_R)$ is an element of the Cartesian product $R_1^C \times ... \times R_n^C \times dom(B_1) \times ... \times dom(B_k)$.

A relationship class R^C consists of a finite *set* $R^C \subseteq R_1^C \times ... \times R_n^C \times dom(B_1) \times ... \times dom(B_k)$ of objects on R for which $id(R)$ is a key of R^C and which obeys the constraints Σ_R.

Integrity constraints of the extended ER model.

Each database model also uses a set of implicit model-inherent integrity constraints. For instance, relationship types are defined on top of their component types and a (relationship) object presumes the existence of corresponding component objects. We typically consider only finite classes. The EER schema is acyclic. Often names or labels are associated with a minimal semantics that can be derived from the meaning of the words used for names or labels. This minimal semantics allows to derive synonym, homonym, antonym, troponym, hypernym, and holynym associations among the constructs used.

The most important class of integrity constraints of the EER model is the class of cardinality constraints. Other classes of importance for the EER model are multivalued dependencies, inclusion and exclusion constraints and existence dependencies[7]. Functional dependencies, keys and referential constraints (or key-based inclusion dependencies) can be expressed through cardinality constraints.

We distinguish three main kinds of cardinality constraints. Given a relationship type $R \triangleq (compon(R), attr(R), \Sigma)$, a component R' of R, the remaining substructure $R'' = R \setminus R'$ and the remaining substructure $R''' = R'' \sqcap_R compon(R)$ without attributes of R.

The participation constraint $card(R, R') = (m, n)$ restricts the number of occurrences of R' objects by the lower bound m and the upper bound n. It holds in a relationship class R^C if for any object $o' \in R'^C$ there are at least m and at most n objects o with $o|_{R'} = o'$, i.e., $m \leq |\{ o \in R^C \,|\, o|_{R'} = o' \}| \leq n$ for any $o' \in (R^C)[R']$ and the projection $(R^C)[R']$ of R^C to R'.

The lookup constraint $look(R, R') = m..n$ describes how many objects o''' from R'''^C may potentially 'see' an object o' from R'^C. It holds in a relationship class R^C if for any object $o''' \in dom(R''')$ there are at least m and at most n related objects o' with $o|_{R'} = o'$, i.e., $m \leq |\{ o' \in \pi_{R'}(R^C) \,|\, o \in R^C \wedge o|_{R'} = o' \wedge o|_{R'''} = o''' \}| \leq n$ for any $o''' \in Dom(R''')$.

Participation and lookup constraints can be extended to substructures and intervals and to other types such as entity and cluster types. Given a relationship type R, a substructure R' of R, R'' and R''' as above. Given furthermore an interval $I \subseteq \mathbb{N}_0$ of natural numbers including 0. The (general) cardinality constraint $card(R, R') = I$ holds in a relationship class R^C if for any object $o' \in \pi_{R'}(R^C)$ there are $i \in I$ objects o with $o|_{R'} = o'$, i.e., $|\{ o \in R^C \,|\, o|_{R'} = o' \}| \in I$ for any $o' \in \pi_{R'}(R^C)$.

The following participation and lookup constraints are examples in Figure 1:

For any $R' \in$ { Responsible4Course, InsertedBy, Semester, Course, Kind}
card(*ProposedCourse*, R') = (0,n),
card(*ProposedCourse*, *Semester Course Teacher*) = (0,1),
card(*ProposedCourse*, *Teacher Semester*) = (2,7),
card(*CourseHeld*, *PlannedCourse*) = (!,1)
card(*PlannedCourse*, *ProposedCourse*[*Semester*] *Room TimeFrame*) = (0,1)

The first constraint is trivial. The second constraint expresses a key or functional dependency. The third constraint requires that any professor has to give at least 2 courses per semester but at most 7 courses per semester. The fourth constraint requires that any planned course must be given. The last constraint requires that rooms are not overbooked.
Lookup constraints were originally introduced by P.P. Chen [1] as cardinality constraints. UML uses lookup constraints. They are easy to understand for binary relationship types without attributes but difficult for all other types. Lookup constraints do not consider attributes of a relationship type. Participation and lookup constraints cannot be axiomatised through a Hilbert- or Gentzen-type logical calculus. If only upper bound are of interest then an axiomatisation can be found in [3, 4]. General cardinality constraints combine equality-generating and object-generating constraints such as keys, functional dependencies and referential constraints into a singleton construct.
Logical operators can be defined for each type. A set of logical formulas using this operator can define the integrity constraints which are valid for each object of the type.

Schemata for the extended ER model.
A set $\{E_1, ...E_n, C_1, ..., C_l, R_1, ..., R_m\}$ of entity, cluster and (higher-order) relationship types on a data scheme DD is called saturated if any relationship and cluster type use only the types from $\{E_1, ...E_n, C_1, ..., C_l, R_1, ..., R_m\}$ as components and cluster and relationship types are layered.
A saturated set of types is also called EER schema $\mathcal{S}$. The EER schema is going to be extended by constraints. The EER schema is defined by the pair $\mathcal{D} = (\mathcal{S}, \Sigma)$. A database $\mathcal{D}^C$ on $\mathcal{D}$ consists of classes for each type in $\mathcal{D}$ such that the constraints Σ are valid.
The classes of the extended ER model have been defined through sets of objects on the types. We may instead use lists, multi-sets or other collections of objects. The definitions used above can easily be extended by structural recursion [8].
A number of domain-specific extensions have been introduced to the ER model. One of the most important extension is the extension of the base types by spatial data types such as: point, line, oriented line, surface, complex surface, oriented surface, line bunch, and surface bunch. These types are supported by a large variety of functions such as: meets, intersects, overlaps, contains, adjacent, planar operations, and a variety of equality predicates.
The translation of the schema to (object-)relational or XML schemata can be based on profile [8]. For instance, for relationship types we may use the identifying components of the component types instead of the components themselves. The treatment of optional components is also specified through the translation profile of the types of the schema. Another profile may require the introduction of identifier types and base the identification on the identifier. Attribute types may be translated into data formats that are supported by the target system.
The EER schema can be used to define views. The generic functions insert, delete, update, projection, union, join, selection and renaming can be defined in a way similarly to the relational model. Additionally, we may use nesting and unnesting functions. These functions form the algebra of functions of the schema and are the basis for defining queries. A singleton view is defined by a query that maps the EER schema to new types. We also may consider combined views which consist of singleton views and which form together another EER schema.
A view schema is specified on top of an EER schema $\mathcal{D}$ by a schema $\mathcal{V} = \{S_1, ..., S_m\}$, an auxiliary schema $\mathcal{A}$ and a (complex) query $q : \mathcal{D} \times \mathcal{A} \rightarrow \mathcal{V}$ defined on $\mathcal{D}$ and $\mathcal{V}$. Given a database $\mathcal{D}^C$ and the auxiliary database $\mathcal{A}^C$. The view is defined by $q(\mathcal{D}^C \times \mathcal{A}^C)$.

Graphical representation for the extended ER model.

We can represent a saturated set of entity, cluster and relationship types by ER diagrams. One possible kind of diagram is displayed in Figure 1. Entity types are represented graphically by rectangles. Attribute types are associated with the corresponding type. Attributes identifying a type are underlined. Relationship vertices are represented graphically by diamonds. Clusters are represented by diamonds labelled with root illustrated $\oplus$ or simply as a common input point labelled by $\oplus$ to a diamond.
This style of drawing diagrams is one of many variants that have been considered in the literature. The main difference of representation is the style of drawing unary types. Tools often do not allow to introduce cluster types and relationship types of order higher than 1. In this case, those types can be objectified, i.e. represented by a

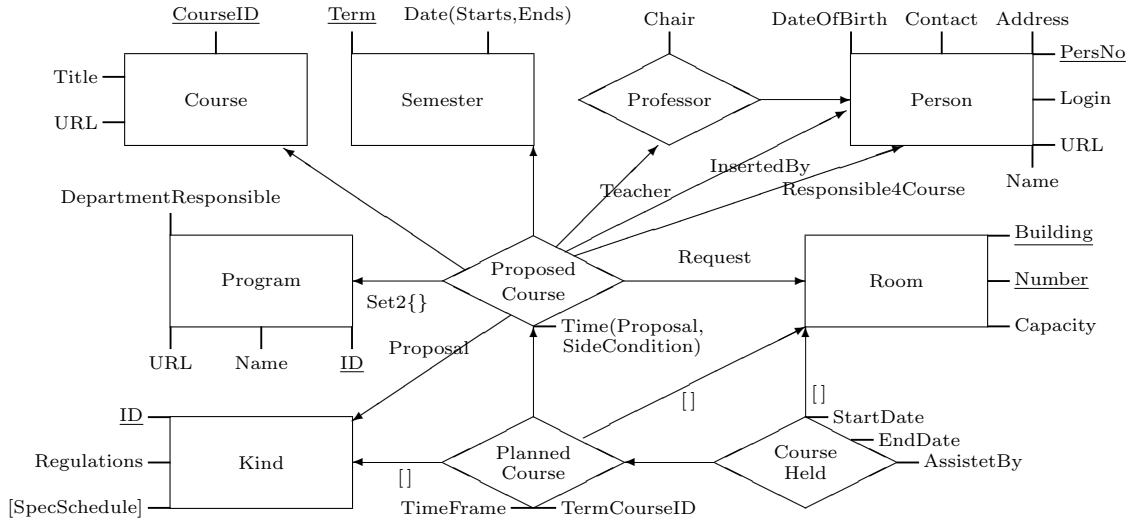

Figure 1: Extended Entity-Relationship Diagram for Course Management

new (abstract) entity type that is associated through binary relationship types to the components of the original type. In this case, identification of objects of the new type is either inherited from the component types or is provided through a new (surrogate) attribute. The first option results in the introduction of so-called weak types. The direct translation of these weak types to object-relational models must be combined with the introduction of rather complex constraint sets. Typically, this complexity can be avoided if the abstract entity type is mapped together with the new relationship types to a singleton object-relational type. This singleton type is also the result of a direct mapping of the original higher-order relationship type.
The diagram can be enhanced by an explicit representation of cardinality and other constraints. If we use participation constraints $card(R, R') = (m, n)$ then the arc from R to R' is labelled by (m, n) If we use lookup constraints $look(R, R') = m..n$ for binary relationship types then the arc from R to R' is labelled by $m..n$.

KEY APPLICATIONS

The main application area for extended ER models is the conceptualisation of database applications.
Database schemata can be translated to relational, XML or other schemata based on transformation profile that incorporate properties of the target systems.

FUTURE DIRECTIONS

The ER model has had a deep impact on the development of diagramming techniques in the past and is still influencing extensions of the unified modelling language UML. UML was starting with binary relationship types with lookup constraints and without relationship type attributes. Class diagrams currently allow n-ary relationship types with attributes. Relationship types may be layered. Cluster types and unary relationship types allow to distinguish generalisation from specialisation.
ER models did not get their native database management systems and are mainly used for modelling of applications at the conceptual or requirements level. ER schemata are translated to logical models such as XML schemata or relational schemata or object-relational schemata. Some of the specifics of the target models are not well supported by ER models and must be added after translating ER schemata to target schemata, e.g., specific type semantics such as list semantics (XML) or as special ordering or aggregation treatment of OLAP applications.
The ER model has attracted a lot of research over the last 30 years. Due to novel applications and to evolution of technology old problems and novel problems are challenging the research on this model. Typical old problems that are still not solved in a satisfactory manner are: development of a science of modelling, quality of ER

schemata, consistent refinement of schemata, complex constraints, normalisation of ER schemata, normalisation of schemata in the presence of incomplete constraint sets. Novel topics for ER research are for instance: evolving schema architectures, collaboration of databases based on collaboration schemata, layered information systems and their structuring, schemata with redundant types, ER schemata for OLAP applications.
Structures of database applications are often represented through ER models. Due to the complexity of applications, a large number of extensions have recently been proposed, e.g., temporal data types, spatial data types, OLAP types and stream types. Additionally, database applications must be integrated and cooperate in a consistent form. The harmonisation of extensions and the integration of schemata is therefore a never ending task for database research.
ER models are currently extended for support of (web) content management that is based on structuring of data, on aggregation of data, on founding data by concepts and on annotating data sets for simple reference and usage. These applications require novel modelling facilities and separation of syntactical, semantical and pragmatical issues. The ER model can be extended to cope with this variety of aspects.
The ER model is mainly used for conceptual specification of database structuring. It can be enhanced by operations and a query algebra. Operations and the queries can also be displayed in a graphical form, e.g. on the basis of VisualSQL. Most tools supporting ER models do currently not use this option. Enhancement of ER models by functionality is however a must if the conceptualisation is used for database development. Based on such enhancement view management facilities can easily be incorporated into these tools.
ER models are becoming a basis for workflow systems data. The standards that have been developed for the specification of workflows did not yet integrate a sophisticated data management.

URL TO CODE
http://www.informatik.uni-kiel.de/∼thalheim/HERM.htm
http://www.is.informatik.uni-kiel.de/∼thalheim/indeeerm.htm
Readings on the RADD project (Rapid Application and Database Development)
Authors: M. Albrecht, M. Altus, E. Buchholz, H. Cyriaks, A. Düsterhöft, J. Lewerenz, H. Mehlan, M. Steeg, K.-D. Schewe, and B. Thalheim.

CROSS REFERENCE
I. DATABASE FUNDAMENTALS
a. Data models (including semantic data models)
III. THEORETICAL ASPECTS
b. Relational Theory

RECOMMENDED READING
Between 3 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.

[1] P. P. Chen. The entity-relationship model: Toward a unified view of data. *ACM TODS*, 1(1):9–36, 1976.
[2] M. Gogolla. *An extended entity-relationship model - fundamentals and pragmatics.* LNCS 767. Springer, Berlin, 1994.
[3] S. Hartmann. Reasoning about participation constraints and Chen's constraints. In *ADC*, volume 17 of *CRPIT*, pages 105–113. Australian Computer Society, 2003.
[4] S. Hartmann, A. Hoffmann, S. Link, and K.-D. Schewe. Axiomatizing functional dependencies in the higher-order entity-relationship model. *Inf. Process. Lett.*, 87(3):133–137, 2003.
[5] U. Hohenstein. *Formale Semantik eines erweiterten Entity-Relationship-Modells.* Teubner, Stuttgart, 1993.
[6] K.-D. Schewe and B. Thalheim. Conceptual modelling of web information systems. *Data and Knowledge Engineering*, 54:147–188, 2005.
[7] B. Thalheim. *Dependencies in relational databases.* Teubner, Leipzig, 1991.
[8] B. Thalheim. *Entity-relationship modeling – Foundations of database technology.* Springer, Berlin, 2000.
[9] B. Thalheim. Codesign of structuring, functionality, distribution and interactivity. *Australian Computer Science Comm. 31*, 6 (2004), 3–12. Proc. APCCM'2004.

Extended Entity-Relationship Model

Bernhard Thalheim
Christian-Albrechts University Kiel, http://www.informatik.uni-kiel.de/∼thalheim/HERM.htm

SYNONYMS

EERM, HERM; higher-order entity-relationship model; hierarchical entity-relationship model

DEFINITION

The extended entity-relationship (EER) model is a language for definition of structuring (and functionality) of database or information systems. It uses inductive development of structuring. Basic attributes are assigned to base data types. Complex attributes can be constructed by applying constructors such as tuple, list or set constructors to attributes that have already been constructed. Entity types conceptualise structuring of things of reality through attributes. Cluster types allow the generalise or to combine types into singleton types. Relationship types associate types that have already been constructed into an association type. The types may be restricted by integrity constraints and by specification of identification of objects defined on the corresponding type. Typical integrity constraint of the extended entity-relationship model are participation, lookup and general cardinality constraints. Entity, cluster and relationship classes contain a finite set of objects defined on these types. The types of an EER schema are typically depicted by an EER diagram.

MAIN TEXT

The extended entity-relationship model uses a data type system for its attribute types, allows to construct entity types $E \stackrel{\circ}{=} (attr(E), \Sigma_E)$ through a set of attributes, and inductively builds relationship types $R \stackrel{\circ}{=} (T_1, ..., T_n, attr(R), \Sigma_R)$ of order i ($i \geq 1$) through a set of (labelled) entity types and of relationship types of order less than i and a set of attribute types. Additionally, cluster types $C \stackrel{\circ}{=} T_1 \dot\cup ... \dot\cup T_n$ of order i can be defined through a disjoint union $\dot\cup$ of relationship types of order less than i or of entity types. Entity/relationship/cluster classes contain a set of tuples of the entity/relationship/cluster types. The data type system is typically inductively constructed on a base type B system by application of constructors such as the tuple constructor (..), set constructor {..} and the list constructor $< .. >$. Types may be optional [..]. The types T can be labelled $l : T$. The types T are typically extended by a set Σ_T of constraints which are defined for components of the type. Only those classes are considered for which the constraints of their types are valid. An entity-relationship schema consists of a set of data, entity, relationship and cluster types which types are inductively built on the basis of the base types.

Given a base type B system. The types of the ER schema are defined through the type equation:

$$T \quad = \quad B \mid (l_1 : T, ..., l_n : T) \mid \{T\} \mid <T> \mid [T] \mid T \dot\cup T \mid l : T \mid N \stackrel{\circ}{=} T$$

A set of entity, cluster and relationship types can be represented by ER diagrams. The diagram representation of EER schemata uses rectangles and diamonds for the entity and relationship types. Attribute types are associated with the corresponding type.

CROSS REFERENCE

I. DATABASE FUNDAMENTALS

a. Data models (including semantic data models)

REFERENCES

ER conferences: http://www.conceptualmodeling.org/
ER history: http://cs-exhibitions.uni-klu.ac.at/index.php?id=185
B. Thalheim. *Entity-relationship modeling – Foundations of database technology.* Springer, Berlin, 2000.