THE MODELING PRIMITIVES FOR COMPONENT RELATIONSHIPS AND A 'DESIGN BY EXAMPLES' METHOD

Marko Junkkari

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1998-13

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1998-13, DECEMBER 1998

THE MODELING PRIMITIVES FOR COMPONENT RELATIONSHIPS AND A'DESIGN BY EXAMPLES' METHOD

Marko Junkkari

University of Tampere
Department of Computer Science
P.O. Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-4475-2
ISSN 0783-6910

The modeling primitives for component relationships and a `design by examples´ method

Marko Junkkari

junken@cs.uta.fi

University of Tampere, Finland

Department of Computer Science

Abstract: In this paper we present essential primitives for abstraction of relationships among composed objects and their components. We start out from modality and then present whether an object, by nature, occurs only in some specific construct or whether it is an independent one. These approaches are well-known in many areas of computer science, but the integrated and exact systematic presentation of these is generally lacking, especially in modeling approaches for component based systems. We represent a 'design by example' method for modeling and designing an abstraction from single component structures of objects. This approach enables us to modify the object type structure during the designing process. Primitives of modeling methods are generally defined only on graphical level, but more accurate presentation is needed because the result of this modeling analysis gives input for detailed design and implementation. For this reason, our definitions are based on an exact and implementation-independent presentation language, that is, set theory.

1. Introduction

The description of relationships among components is a complex question in the component-oriented approaches to information systems engineering. When a designer discusses the reuse of class libraries, environments, components, or frameworks, the consideration is often rather imprecise. It can be said that one reason for this is that the essential primitives of *object type structures* (aggregation schema) for components are not well defined. Here, we present both the basic primitives for describing the conceptual structure of object types (classes) and a method for modeling the structure on the basis of primitives, a method we call the 'design by example' method. Similar approaches have been used for example in database design [e.g. Kantola *et al.*, 1992] and in formulation of complicated queries[1] and in designing of frameworks [Koskimies and Mössenböck, 1995].

To begin with, we need two kinds of relations among the object types for describing the modality of containment in the application domain at hand. One to describe whether a complex object *necessarily* has some component of the type at hand and one for established whether a relationship is a *possible* one. Moreover, we need two kinds of object types: *weak* and *independent* ones. The first describes the situation when an object of the type cannot appear without a composed object. The second describes the situation when objects can appear independently from any composed object. The presented primitives are well known in different approaches, but as far as we know this is the first time when the primitives are put together by a systematic and integrated view in order to describe relationships for component based systems.

The object type structure at hand functions as the framework for every single object structure. Under the type structure, necessary relationships form the minimal object structure that all the object structures have to satisfy. Together, necessary and possible relationships form the maximal framework, into which every object structure has to fit.

Modeling begins typically from a type structure. The modeler tries to find a model that every single object structure has to satisfy. The problem in this approach is that it requires both a knowledge of every future structure of objects and the testing of the model during the modeling and designing processes. To bypass such complications, we suggest the use of the 'design by example' method for describing the object type structure. This method enables the testing of the type structure and, analogously, of the new object structure so that we can determine whether it satisfies the general model.

Most of the modeling methods are defined only on a graphical level, the formal definitions are generally missing. Because the modeling of application domains gives the input for design and implementation of information systems, it should be clear that the exact definition of application specific concepts is an essential aspect in the analysis of application domains. Our formalisms are, therefore, presented in terms of set theory which is an exact, well-know and implementation-independent

[1] The first presentation: Moshé M. Zloof: *Query-by-Example: the Invocation and Definition of Tables and Forms*. VLDB 1975: 1-24

presentation method. In this paper, we assume the reader to be familiar with standard set theory. Some notational conventions have alternative ways of representing in set theory. According to [Niemi and Järvelin, 1992], we use the following notational conventions.

- The *ordered pairs* are denoted between angle brackets, e.g. <a,b>.
- The *power set* of a set S is denoted by P(S).
- The *signature* of a function is denoted by f: D→ R where f is a *function symbol*, D is a *domain set* and R is a *range set*. In complex cases a domain set is a Cartesian product.

Section 2 discusses related work especially from the point of view of the problems with relationships between parts and wholes in general. The definitions of modeling primitives are presented in section 3. In section 4, then we present our modeling method, 'design by example'. In conclusion, further studies are discussed briefly in section 5.

2. Related work

The basic principles we define are well known in computer science. The concept of modality is familiar from several contexts. Therefore we overlook references to modal logic as well as to modality with an entity and its possible attributes. The concept of weakness (an object depends on another object) is familiar from the basics of ER-model [Chen, 1976] as well as from object oriented analysis, design, and databases.

Problems that are related to component relation are, thus, common within several areas of computer science, in areas such as AI, object oriented analysis and design, semantic data models and database design. Transitivity and strong dependency between related objects are the characteristics of a component relation. These make the processing of the relation complicated and systematic and exact management rules are typically missing. Especially the consideration of component relations among *concepts* (object types, entity types) is an open problem with *part-whole* relations [Artale, Franconi and Guarino, 1996]. The component relationship between objects is a kind of a fact in the real world and the conceptual component relation is a kind of a generalization of single object relationships.

Together with generalization, aggregation has been understood as on of the most important methods for conceptualization with database abstractions [Smith and

Smith, 1977]. Therefore, in semantic data models, there is a need to present component relationships. In IFO [Abiteboul and Hull, 1987] and in GSM [Hull and King, 1987], a kind of a component relationship is described by aggregation with so called cross-vertex and formally the connection of the components is described by tuples. For example the domain of the motorboat is described as a set of ordered pairs which have a hull and a motor as their elements. In some enlargements of ER-model [see e.g. Spaccapietra and Parent, 1992] the problems of component relations have come current with the so-called complex attributes. One purpose of the object oriented database development has been the presentation and management of complex object [e.g. Hua and Tripathy, 1994; Cluet, 1998].

In object oriented analysis the component hierarchy is called a w*hole-part structure*, an *assembly structure* or an *aggregation structure* [Coad and Yourdon, 1991]. In OMT [Rumbaugh *et al.*, 1991] and in UML [Harmon and Watson, 1997] aggregation is understood as a kind of an association and it is defined among classes (types). In the object-oriented approach, our *weak type* refers to the kind of aggregation that is also sometimes called *composition* or *composite aggregation*. In this case when the composed object is deleted also its components are necessarily deleted. The purposes of aggregation do still mostly fall under definitions on the graphical level and generally accurate definitions are missing.

Description logics[2] [e.g. Brachman and Levesque, 1984; Borgida *et al,* 1989; Borgida, 1995] are languages for presenting structural knowledge by is-a and part-whole hierarchies. The ontological primitives of description logics are individual (object), concept and role. A concept refers a set of individuals and the role is a principle that restricts extension of the concept. A part of a whole can also be seen as a kind of role. Traditionally, role is presented as a binary relation but there are also extensions for transitive and multi-placed roles [e.g. Borgida, 1995; Sattler, 1996]. With part-whole relations, one purpose of the description logics has been in distribution of different kinds of part-whole relations [Sattler, 1995]. Another purpose for it has been the transitive management of one basic *part-of* relation to objects and concepts [Padgham and Lambrix, 1994; Lambrix, 1996]. The *direct part* relation between two concepts can be defined informally [Lambrix, 1996, p. 81]: ‘A concept C1 is a direct part of another C2 if individuals belonging to C1 can be parts

[2] Description logics have also been referred to as *terminological logics, KL-ONE-like languages* and *frame-based systems*.

of individuals belonging to C2.’ This corresponds considerably with the union of necessary and possible component relations in our terminology.

Theoretical aspects of part-whole relations are generally based on mereology [Lesniewski, 1984; Simons and Dement, 1996, see also Varzi, 1996] or mereotopology [Varzi, 1996; Smith, 1996] that study only composed objects instead of types or concepts. From the conceptual modeling perspective we can consider the objects of a domain by simply considering the connections of objects. Another modeling choice is the use of an attribute ”has-component” when connections between individuals are ensued from this [Artale *et al.*, 1996]. We can make a hierarchy for attributes, but this approach is more like an ‘ad hoc’ solution.

In this paper, we adapt the term ‘design by examples’ (DBE) from relational database design [Mannila and Räihä, 1986; Kantola *et al*., 1992]. In that appraoch the extensional level is called *database instances* whereas the term *database schema* can be interpreted to correspond with the conceptual level. The database design is typically begun by using an example ER-model. If the modeler does not know what kind of demands the database paradigm makes, then the actualization of ER-schema can be impossible to satisfy as a database. That is why particular database instances are said to satisfy the *integrity constrains*. One aim of DBE has been to provide real-time feedback to the designer. If given examples indicate a failure in the design then the designer can immediately modify the examples or modify the database schema.

3. The modeling primitives of a component based system.

An *object* is an extensional level entity with identity. The *object type,* shortly *type,* is understood to be an abstract type that refers to objects. The set of objects in the application domain at hand is represented by an object set, denoted by O-set. Of the component relationships in the application domain we present only the immediate component relationships explicitly. If an object b is a part of another object a, we say that b is an *object-component* of a.

Definition 1: If a ($\in$ O-set) and b ($\in$ O-set) are two objects in the application domain at hand so that a has the *immediate object-component* b or b is the immediate object-component of a then this immediate relationship is denoted by the pair $<a,b>$.

The component system of objects is called the *object system* and it contains the set of objects and immediate component relationships O-rel on this set. The interpretation of O-rel is: if <a,b> $\in$ O-rel, then the object b is an immediate component of a.

Definition 2: Component hierarchies among objects related to the application domain at hand are represented as an acyclic binary relation consisting of immediate object-component relationships. This binary relation is called by *object system* and it is denoted by O-rel, i.e. O-rel $\subseteq$ O-set $\times$ O-set or O-rel $\in$ P(O-set $\times$ O-set).

The set of types related to the application domain at hand is denoted by T-set. We assume that every object of the application domain has some object type. When an object a is of the type A we say that a inheres in the extension of the object type A or the object a is type A.

Definition 3: If A is an object type, i.e. A $\in$ T-set then *extension* of A (denoted by ext(A)) means the set consisting of all the objects to be type A. In other words the notation ext(A) contains all instances of A.

As well as we define the connection between two objects by ordered pair, analogously we present immediate relationships between two object types. The set of object types abstracted from the application domain at hand is denoted by T-set. We will give the semantic of immediate relationships among object types later in definitions 5-7.

Definition 4: If A ($\in$ T-set) and B ($\in$ T-set) are two object types abstracted from the application domain at hand so that A has *immediate type-component* B or B is the immediate type-component of A then this immediate relationship is denoted by the pair <A,B>.

If the type A has a *necessary type-component* B then all the objects of type A have a component that is type B. The immediate necessary component relation between two types A and B is informally defined as follows: A contains immediately B as a component if and only if every object of A has a direct component that inheres in the extension of B. The binary relation T_N-rel contains all the immediate necessary type-component relationships.

Definition 5: The object type A has *immediate necessary type-component* B (denoted by <A,B> $\in$ T_N-rel) iff every objects of A has object-component to be type B, i.e. <A,B> $\in$ T_N-rel $\leftrightarrow \forall a \in$ ext(A) $\exists b \in$ ext(B): <a,b> $\in$ O-rel.

If the type A has a *possible type-component* B then some objects of the type A have a component of type B but not all.

Definition 6: The object type A has *immediate possible type-component* B (denoted by $<A,B> \in T_P\text{-rel}$) iff some object of A has object-component to be type B, but not all, i.e. $<A,B> \in T_P\text{-rel} \leftrightarrow \exists a \in ext(A)\ \exists b,c(\neq b) \in ext(B)$: $<a,b> \in O\text{-rel} \wedge <a,c> \notin O\text{-rel}$.

Naturally, the sets of necessary and possible relations are mutually disjoint. *Type system* T-rel is now defined as a union of necessary and possible relation.

Definition 7: The component relation among object types contains immediate necessary and possible component relationships and it is denoted by T-rel, i.e. $T\text{-rel} = T_N\text{-rel} \cup T_P\text{-rel}$.

A *weak type* is an object type whose all occurrences appear only as a part of some construct. If a type is not weak, it is independent.

Definition 8: The object type A is a *weak type* iff all the objects of A are immediate object-component of some objects, i.e. $\forall a \in ext(A)\ \exists b \in O\text{-set}$: $<b,a> \in O\text{-rel}$.

Definition 9: The object type A is an *independent type* iff some objects of A are not immediate object- components of any object, $\exists a \in ext(A)\ \neg\exists b \in O\text{-set}$: $<b,a> \in O\text{-rel}$.

The description of object type depends on the demand of the application domain and environment at hand. In the example of real life, where the object type CAR has type-component MOTOR, there can be alternative purposes relating the environment of the information system. At first MOTOR can be independent type, when the instances of MOTOR can exist without any car. However, if in our information system one sells and buys used cars but not any components of these, then MOTOR is by nature of a weak type. In general the modeler of an information system determines whether a type-component is a weak or independent one.

4. Design by example and structure testing

From the modeling perspective we are primarily interested in general concept structures or type structures, not just merely objects. A type structure is the frame for structural objects that satisfy the relevant part of the object type model. We can examine an arbitrary object structure and the type hierarchy which is the frame at hand. Within the component structure, necessary relationships form the minimal object structure that every object of the current type has to satisfy. By appending possible relationships into the system we get the maximal structure in which every object of the type has to fit in.

On the basis of our primitives, we present a method for designing and testing a component type hierarchy during the modeling process. This method is called 'design by example'. In this model there is first a component structure of objects from which the type model is generalized on the basis of given object structures. We start out from extensional level i.e. single object structures and basing on these we generalize a type model basing on example object structures. This approach enables changes in the type model during the designing process. Moreover, in this method we can test both the particular object structures and our general type structure during the modeling and designing process. A few arbitrary example object structures can hardly provide all the information for the general model, but these are valuable pieces of knowledge. A suitable example can clearly illustrate the problems in a suggested design. From particular object structures we can automatically generate a type model that is a generalisation of given structures.

With the operations we define, it is presumed that the set of object types is implicitly given and for every object there is a known object type. These primitives are typically modelled by classification so that every relevant object has a current object type, and inversely.

We start out with the function *necessary_relations* that yields the set of ordered pairs of types basing on the definition 5. The function takes two arguments: the type set T-set and an object relation O-rel, where all the object structures are given by one object relation. If an object relation O-rel$_i$ presents one structure instance, then the whole relation O-rel is the union of every single structure.

Operation 1:

necessary_relations: P(T-set) × P(O-set × O-set) → P(T-set × T-set)
necessary_relations(T-set, O-rel) =

$$\{\langle A,B \rangle \mid \forall a \in ext(A)\ \exists b \in ext(B): \langle a,b \rangle \in \text{O-rel} \wedge A,B \in \text{T-set}\}$$

Analogously the function *possible_relations* yields possible ordered pairs of types on the basis of definition 6.

Operation 2:

possible_relations: $P(\text{T-set}) \times P(\text{O-set} \times \text{O-set}) \rightarrow P(\text{T-set} \times \text{T-set})$

possible_relations $(\text{T-set, O-rel}) = \{\langle A,B\rangle \mid \exists a \in ext(A)\ \exists b,c(\neq b) \in ext(B)$: $\langle a,b\rangle \in \text{O-rel} \wedge \langle a,c\rangle \notin \text{O-rel} \wedge A,B \in \text{T-set}\}$

Weak types can be generated from an object system at hand by using the function *weak_types*. On the basis of definition 8, an object type inheres in the set which the function returns if for every object of the current type, there exists a construct in the object system. The function *independent_types* yields, analogously, the set of these types that are independent according to definition 9.

Operation 3:

weak_types: $P(\text{O-set} \times \text{O-set}) \times P(\text{T-set}) \rightarrow P(\text{T-set})$

weak_types$(\text{O-rel, T-set}) = \{X \in \text{T-set} \mid \forall z \in ext(X)\ \exists y \in \text{O-set}: \langle y,z\rangle \in \text{O-rel}\}$

Operation 4:

independent_types: $P(\text{O-set} \times \text{O-set}) \times P(\text{T-set}) \rightarrow P(\text{T-set})$

independent_types$(\text{O-rel, T-set}) = \{X \in \text{T-set} \mid \exists z \in ext(X)\ \neg\exists y \in \text{O-set}: \langle y,z\rangle \in \text{O-rel}\}$

Let us take an example, where the O-set_1 is {a1, a2, b1, b2, b3, c1, c2, d1, d2, d3, d4, e1, e2, e3, f1}. Objects a1 and a2 are type A, b1, b2, b3 are type B, c1 and c2 are type C, d1, d2, d3 and d4 are type D, e1, e2 and e3 are type E and f1 is type F. Let object system O-rel_1 be the following binary relation {<a1,b1>, <a1,c1>, <b1,d1>, <b1,e1>, <b1,f1>, <a2,b2>, <a2,c2>, <b2,d2>, <b2,e2>, <b3,d3>, <b3,e3>}. The object types C, E and F are weak ones, because the objects of these do not appear without the construct in the object system. The types A, B and D are independent ones, because there are independent objects that inhere in the extensions of these. The type A has two necessary type-components B and C, because for every object of type A there is an object of the types B and C. Analogously, the type B has necessary type-components D and F. Moreover, B has a possible type-component F, because b1 has the component f1, but b2 and b3 have no components of the type F. Now our type structure T-rel1 contains necessary component relationships <A,B>, <A,C>, <C,D>, <C,E> that the function *necessary_relations* yields and a possible component relationship <C,F> that the function *possible_relations* yields. As a whole, the type hierarchy is {<A,B>, <A,C>, <C,D>, <C,E>, <C,F>}.

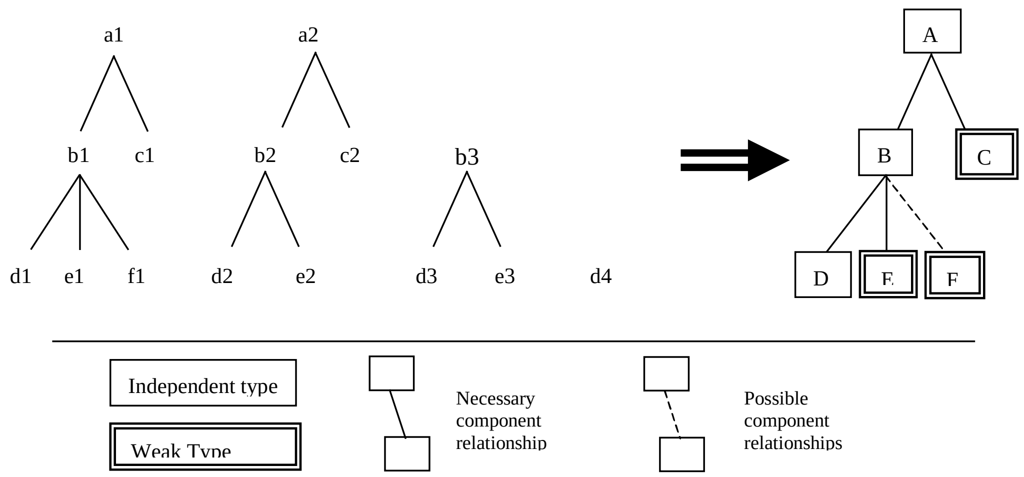

Figure 1. Conceptualisation from the object system O-rel$_1$to the type system T-rel$_1$.

In graphical representation in figure 1 a lower level object or type is a component of the upper level ones. Independent type is presented by a rectangle and a weak type is presented by a double-line rectangle. The possible component relation between types is described by a dotted line.

For the design system, we need of course a method to check whether an arbitrary object structure satisfies the type model at hand. Here we present only the function that yields true-value if the given object structure satisfy previous model. In 'design by example' method false-value means that we have to change our type system for demand that the new object structure gives. For detailed consideration we would need more detailed information about what these parts of structures are that do not comply with each other. Because of the vast number of possible situations, it is not possible to take these into consideration in this context.

The principles of testing are derived from our definitions. We consider the composed object a (type A) and its component structure. The structure of a is legal if the following terms are met:

1. The object type (= A) of a has to be an independent type.
2. For all the immediate object-components of a there has to be a type that is an immediate type-component of A.
3. If A has a necessary type-component B then there has to be an object that is of the type B and is an immediate object-component of a.
4. All the immediate and indirect object-components of a have to partially satisfy points 2 and 3 such that the current object is instance of a and its type

is instance of A.

In formal representation the function *legal_check* agrees with six arguments. The object a is the composed object that we want to insert into our system and A is its type. O-rel' is the union of the old object structure O-rel and the structure of the object a. T-rel is the type system and T_N-rel is the relation of necessary type-component relationships based on the previous object system. Similarly, T_I-set is the set of independent types in the existent type system. To simplify the presentation of the function and its signature, we assume that T-rel, T_N-rel and T_I-set are given implicitly.

Operation 5:

legal_check: O-set' × T-set × P(O-set' × O-set') → {true, false}

legal_check (a, A, O-rel') =

$$\begin{cases} \text{true, if } A \in T_I\text{-set} \wedge \textit{structure\_check}(a, A, \text{O-rel'}) \\ \text{false, otherwise} \end{cases}$$

where

structure_check(a, A, O-rel') =

$$\begin{cases} \textit{structure\_check}(x, B, \text{O-rel'}), \\ \qquad \text{if } \exists B \in \{Y \mid \langle A,Y\rangle \in \text{T-rel}\} \wedge \exists x \in \{z \mid \langle a,z\rangle \in \text{O-rel'}\}: x \in ext(B) \\ \text{true, if } \neg\exists\langle a,b\rangle \in \text{O-rel'} \wedge \neg\exists\langle A,B\rangle \in T_N\text{-rel} \\ \text{false, otherwise} \end{cases}$$

To enhance the running example we insert into the system the object b5 that has immediate object-components d1, d5 and e5, so O-rel_1' is $\text{O-rel}_1 \cup$ {<b5,d1>, <b5,d5>, <b5,e5>}. The current object set O-set_1' is now {b5, d5, e5} $\cup$ O-set_1. Let object b5 be of the type B, d5 of the type D and e5 of the type E. In this case the function *legal_check*(b5, B, O-rel_1') yields true, because all the conditions is satisfied. Note that the legal check operation accepts this example, although the composed object has more than one component being of the same type, provided they partially satisfy the conditions.

The form of operation 5 enables us to apply it independently from the modeling method we use. In other words, if we have modelled by conceptual approach, we can check whether the type structure corresponds to the instance level. We can just as well test whether the construct satisfies the demands we have given to the type system.

5. Discussion

We have here presented and defined the basic primitives that are needed to describe component relationships among object types. By making a difference whether the type is a weak or an independent one and whether the component relation among object types is a possible or a necessary one, we can consider two kinds of approaches within the same system. Firstly, a component can depend on its constructs, and secondly, the composed object can be dependent on its components. We have presented how the component relation among object types can be understood and we have also formally presented the functions by which we can generate the structure of types on the basis of single object structures. By using the modeling method we have presented, a designer can test and evaluate the model under the designing process.

Although some important particulars are not presented here, such as disjoitnes, number value restrictions, and updating questions, the basic primitives, we have defined give a framework for presenting also those in an exact and systematic manner. Furthermore, our definitions give a general framework for the analysis of different indirect connections among objects and object types. We can for example generate the common components of two objects or object types and we can ask whether two components are contained the in same construct. With these primitives, we can analyze both object structures and object type hierarchies.

From the modeling perspective, the type structure gives a general framework for a structure of objects. The system we have presented implicitly gives the rules for object structures and for updating of component system. These rules are based either on description of object types or on description of relationships among object types. If an object of a weak type is inserted or deleted then it is necessary that we also insert or delete some composed object that contains the inserted/deleted object. Moreover, if one inserts an object then one also has to insert its necessary components.

References

[Abiteboul and Hull, 1987] Serge Abiteboul and Richard Hull, IFO: A formal semantic database model, *ACM Transaction on database systems.* Vol. 12, No. 4, 1987. 525-565

[Artale *et al.*, 1996] Alessandro Artale, Enrico Franconi, Nicola Guarino and Luca Pazzi, Part-Whole relations in object-centered systems: An overview, *Data and knowledge engineering*, Vol. 20, No 3, 1996. 347-383

[Artale, Franconi and Guarino, 1996] Alessandro Artale, Enrico Franconi and Nicola Guarino, Open problems for Part-Whole relations. *International Workshop on Description Logics, DL-96*. Boston MA, November 1996.

[Borgida *et al,* 1989] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness and Lori Alperin Resnick, Classic: A structural data model for objects, *ACM Sigmod record.* Vol. 18, No. 2, 1989. 58-67

[Borgida, 1995] Alexander Borgida, Description logics in data management, *IEEE Transactions on knowledge and data engineering.* Vol. 7, No. 5, 1995. 671-682

[Brachman and Levesque, 1984] Ronald J. Brachman and H. Levesque, The tractability of subsumption in frame-based description languages, *AAAI-84*, 1984. 34-37

[Cluet, 1998] Sophie Cluet, Designing OQL: Allowing objects to be queried, *Information systems*. Vol. 23, No. 5, 1998. 279-305

[Chen, 1976] Peter Chen, The Entity-Relationship model -toward a unified view of data, ACM Transactions on database systems, Vol. 1 No. 1, 1976. 9-36

[Coad and Yourdon, 1991] Peter Coad and Edward Yourdon, *Object-Oriented Analysis.* Second edition, Prentice Hall (Yourdon Press), 1991.

[Harmon and Watson, 1997] Paul Harmon and Mark Watson, *Understanding UML: The Developer's Guide*. Morgan Kaufmann, 1997.

[Hua and Tripathy, 1994] Kien A. Hua, Chinmoy Tripathy, Object Skeletons: An Efficient Navigation Structure for Object-Oriented Database Systems, *IEEE: ICDE*. 1994. 508-517

[Hull and King, 1987] R. Hull and R. King, Semantic database modeling: Survey, applications and research issues, *ACM Computing surveys.* Vol. 19 No. 3, 1987. 201-260

[Kantola *et al*., 1992] Martti Kantola, Heikki Mannila, Kari-Jouko Räihä and Harri Siirtola, Discovering functional and inclusion dependencies in relational databases, *International journal of intelligent systems*. Vol. 7, 1992. 591-607

[Koskimies and Mössenböck, 1995] Kai Koskimies and H. Mössenböck, Designing a framework by stepwise generalizaton, Proc. of 5th European Software Engineering Conference, *Lecture Notes in Computer Science 989*. 1995. 479-498.

[Lambrix, 1996] Patrick Lambrix, *Part-Whole Reasoning in Description Logics*. Doctoral Dissertation, Department of Computer and Information Science, Linköping University, Sweden, 1996.

[Lesniewski, 1984] *Lesniewski's Systems: Ontology and Mereology*. eds. by Srzednicki and Rickey, Nijhoff International Philosophy Series, Martinus Nijhoff Publishers, Poland, 1984.

[Mannila and Räihä, 1986] Heikki Mannila and Kari-Jouko Räihä. Design by example: An application of Armstrong relations. *Journal of computer and system sciences,* Vol. 33, No. 2, 1986. 126-141

[Niemi and Järvelin, 1992] Timo Niemi and Kalervo Järvelin, Operation-oriented query language approach for recursive queries - Part 1: Functional definition, *Information systems*. Vol. 17 No. 1, 1992. 49-75

[Padgham and Lambrix, 1994] Lin Padgham and Patrick Lambrix, A framework for part-of hierarchies in terminological logics, *Proc. of the 4th International Conference on Principles of Knowledge Representation and Reasoning, KR-94*. 1994. 485-496

[Rumbaugh *et al*., 1991] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy and William Lorensen, *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[Sattler, 1995] Ulrike Sattler, A concept language for an engineering application with Part-Whole relations, *Proceedings of the International Workshop on Description Logics - DL-95*, Roma, Italy, 1995. 119-123

[Sattler, 1996] Ulrike Sattler, A concept language extended with different kinds of transitive roles. *Lecture Notes in Artificial Intelligence1137*. 1996.

[Simons and Dement, 1996] Peter Simons and Charles Dement, Aspects of merolology of artifacts, In Roberto Poli and Peter Simons (eds.), *Formal Ontology*. Kluwer Academic Publishers, 1996. 255-276

[Smith and Smith, 1977] J. M. Smith and D. C. Smith, Database abstraction: Aggregation and generalization, *ACM Transaction on database systems.* Vol. 2, No. 2, 1977. 105-133

[Smith, 1996] Barry Smith, Mereotopology: A theory of parts and boundaries, *Data and knowledge engineering* Vol. 20, No 3, 1996. 287-303

[Spaccapietra and Parent, 1992] Stefano Spaccapietra and Christine Parent, ERC+: An object based entity relationship approach, in Loucopoulos and Zicari (Eds.), *Conceptual Modelling, Database and Case: An integrated View of Information Systems Development*. John Wiley, 1992.

[Varzi, 1996] Achille Varzi, Parts, wholes, and part-whole relations: The prospects of mereotopology, *Data and knowledge engineering*. Vol. 20, No 3, 1996. 259-286