Authors:	Niemi Timo, Junkkari Marko, Järvelin Kalervo, Viita Samu
Name of article:	Advanced query language for manipulating complex entities
Year of publication:	2004
Name of journal:	Information Processing & Management
Volume:	40
Number of issue:	6
Pages:	869-889
ISSN:	0306-4573
Discipline:	Natural sciences / Computer and information sciences
Language:	en
School/Other Unit:	School of Information Sciences

URN: http://urn.fi/urn:nbn:uta-3-738
DOI: http://dx.doi.org/10.1016/S0306-4573(03)00065-7

Advanced query language for manipulating complex entities

Timo Niemi[a,*], Marko Junkkari[a], Kalervo Järvelin[b] and Samu Viita[a]

[a]*Department of Computer and Information Sciences, FIN-33014 University of Tampere, Finland*

[b]*Department of Information Studies, FIN-33014 University of Tampere,, Finland*

**Corresponding author*

Dr. Timo Niemi
Department of Computer and Information Sciences
FIN-33014 University of Tampere
Finland

E-mail: tn@cs.uta.fi
Tel: +358 3 215 6782
Fax: +358 3 215 6070

Abstract

Complex entities are one of the most popular ways to model relationships among data. Especially complex entities, known as physical assemblies, are popular in several applications. Typically, complex entities consist of several parts organized at many nested levels. Contemporary query languages intended for manipulating complex entities support only extensional queries. Likewise, the user has to master the structures of complex entities completely, which is impossible if a physical assembly consists of a huge number of parts. Further, query languages do not support the manipulation of documents related to parts of physical assemblies. In this paper we introduce a novel, declarative and powerful query language, in which the above deficiencies have been eliminated. Our query language supports text information retrieval related to parts and it contains intensional and combined extensional-intensional query features. These features support making queries of new types. In the paper we give several sample queries, which demonstrate the usefulness of these query types. In addition, we show that conventional extensional queries can be formulated intuitively and compactly in our query language. Among other things this is due to our query primitives allowing removal of the explicit specification of navigation from the user.

Keywords: Complex entities, physical assembly, query language, information retrieval, XML documents

1. Introduction

An information system consists of entities, their properties and relationships. Similar entities, are grouped into an entity type with a unique name. Entities belong to the extensional level (the instance level) whereas entity types belong to the intensional level (the schema level). The properties (attributes) of entity types belong to the intensional level whereas their values to the extensional level. Relationships may also be represented both at the extensional and intensional level. At the intensional level a relationship is represented through entity types and the representation at the extensional level is based on entities. A relationship may contain, in addition to participating entity types, attributes to express its characteristics.

In modeling relationships among entities three basic relationships are usually distinguished: the is-a relationship (or specialization/generalization), the association (or member-of relationship) and the part-of relationship (Rumbaugh et al., 1991; Rumbaugh et al., 1999; Motschnig-Pitrik & Kaasböl, 1999; Renguo et al. 2000; Wand et al., 1999). In the is-a relationship one organizes similar entity types hierarchically. If X and Y are entity types and X *is-a* Y holds then X is called the subentity type of Y and Y the superentity type of X. At the extensional level, each entity belonging to the entity type X also belongs to the entity type Y.

Association models an event, a phenomenon or a fact among independent entity types / entities. Typically, each entity type / entity participating in an association plays some role. In an association the participating entity types are assumed to be conceptually at the same level (Renguo et al., 2000).

In a part-of relationship entities / entity types are not conceptually at the same level because they have different complexity. In modeling a part-of relationship it is essential to recognize the entities / entity types that play the roles of parts in an entity / entity type which is the whole. The modeling of a part-of relationship requires structuring among entities / entity types and results in several hierarchy levels among them. Therefore transitivity is a primary characteristic of the part-of relationship. In this paper we deal only with part-of relationships.

The part-of relationship has no established terminology. For example, it has been called whole-part association (Civello, 1993), part-whole relationship (Motschnig-Pitrik & Kaasböl, 1999), whole-part relationship (Barbier et al., 2000), part-whole hierarchy (Pazzi, 1999), part-of structure (Rousset & Hors, 1996), aggregation (Rumbaugh et al., 1991), complex object (Savnik et al., 1999) and composition relation (Urtado & Oussalah, 1998). Here we call, by following Järvelin and Niemi (1999), a complex entity such an entity modeled by the part-of relationship whose parts are organized as several nested substructures. The context tells whether we mean its intensional or extensional level or both. Our notion of complex entity refers to a single unit, which contains all its parts and which has its own function in the real world. It is typical of physical assemblies that they are complex entities which have been constructed for a specific purpose in the real world. For example, we can consider a car as a physical assembly, which consists immediately of a body, engine, transmission, etc. and in turn, a body consists of a frame, doors and windows etc. In other words, a car is a single unit in the real world, which is capable of moving in a controlled way whereas any of its parts has not this property.

Most database query languages support only extensional queries, i.e., query results consist only of extensional level information. We develop a declarative query language for complex entities, which supports both extensional and intensional queries.

In the semantic sense, a component and its immediate components often are of special interest in complex entities. For example, the assembly and disassembly of a complex entity usually happens in phases where a component and its immediate components are treated in one phase only. In many applications it is natural to associate a document with each part of the physical assembly at hand, e.g. for giving instructions for assembly or maintenance. Further, in many applications the manipulation of complex entities and their documentation is needed at the same time. To the best of our knowledge, our query language is the first proposal for this purpose.

The rest of the paper is organized as follows. In Section 2 we review both approaches to model complex entities and query languages for manipulating them. We shall also present the goals of our query language. In Section 3 we introduce complex entity modeling and related XML documentation of our system. In this section we also illustrate our sample application. The primitives of our query language and the notion of variable are introduced in Section 4. In Section 5 we formulate sample queries of different types. The properties and implementation of our language are discussed in Section 6. Summary is given in Section 7.

2. Related work

Complex entities are common in the real world. They have an important role in many advanced applications in engineering, manufacturing and graphics design. They have been also used for organizing medical terminologies (Liu et al., 1996). Niemi and Järvelin (1995; Järvelin & Niemi 1999), like several other authors (e.g. Sacks-Davis et al., 1995; Zobel et al., 1991; Lambrix & Padgham, 2000), have proposed complex entities for representing and manipulating hierarchical documents. Järvelin and others (2000) have shown that complex entities are natural structures for informetrics. Complex entities have also proven useful in the Web as embedded in Web pages defined in the OHTML syntax and in the Object Exchange Model (OEM) (Riet, 1998).

Complex entities are popular because they support the modeling of semantically different relationships in applications. Many authors (e.g. Winston et al., 1987; Kim et al., 1987; Civello, 1993; Motschnig-Pitrik & Kaasböll, 1999; Halper et al., 1998; Barbier et al., 2000) have focused on the semantics of complex entities and on categorizing them. For us, the most important semantic distinction is whether a complex entity is exclusive or shared. A component is exclusive if it can be attached as an immediate part to at most one more complex entity whereas a shared component may be attached to any number of more complex entities (Artale et al. 1996; Halper et al., 1998). In this paper we assume that a complex entity consists only of exclusive components. This constraint is characteristic of physical assemblies such as vehicles or buildings (Halper et al., 1998). In this paper we consider physical assemblies with their related textual documents.

2.1. Approaches to modeling and representing complex entities

There are two basic approaches to represent information in databases: the value-oriented and the object-oriented approach (Ullman, 1988). In the former, values of some attributes are used for the identification of entities or relationships, i.e., these attributes act as their keys. In the object-oriented approach a unique identifier is assigned to each entity (called object) and it is used to refer to the entity. In addition to attributes, an entity may have functional properties. These functional properties are called methods and they are implemented as pieces of code.

The relational model is based on the value-oriented approach. It does not support the representation of complex entities directly. In relational databases a complex entity is represented in several relations. Thus the same values of attributes must be stored in several relations in order to maintain semantic connections among data. The manipulation of a complex entity as a single unit therefore requires data collection from several relations. The construction of complex entities of several hierarchy levels presupposes the specification of many relational joins for which the user is responsible. Yet the result of a relational query is always a flat relation which does not make the structure of a complex entity obvious. Therefore better ways of representing complex entities are needed.

NF^2 relations or non-first normal form relations are also based on the value-oriented approach, but support the complex entity notion (Roth et al., 1988). They allow relation-valued attributes, which may contain relation-valued attributes, etc. Järvelin and Niemi (1995; 1999) review the use of NF^2 relations in the IR area. NF^2 relations make the structuring among component entities of a complex entity explicit.

From the perspective of complex entities the NF^2 relational model has two essential disadvantages. First, it does not contain operations for analyzing complex entity types. Second, it requires that each atomic-valued and relation-valued attribute has a unique name. However, the same component entity type may appear in several composite types of a complex type, e.g. bolts may belong to several parts of an airplane. From the semantic perspective, it is desirable to use the same entity type name wherever the entity type appears.

Object-orientation has concentrated on modeling and manipulating the is-a relationship rather than complex entities (complex objects in the object-oriented terminology) (Renguo et al., 2000). One indication on this is that the support of object-oriented programming for complex entities, similar to the support for the is-a relationship, is desired (Motschnig-Pitrik & Kaasböll, 1999). Likewise, the indexing mechanisms of object-oriented databases mainly support the manipulation of the is-a and the association relationships. An exception is the work by Renguo and others (2000) who developed the indexing of complex entities.

Several authors have realized that in object-orientation complex entities are often treated as a kind of association although they usually require particular semantics and update mechanisms (see e.g. Motschnig-Pitrik & Kaasböll, 1999; Renguo et al., 2000). They can neither be represented through ordinary attributes because the distinction between a property and a component is lost in this case (Civello, 1993; Artale et al., 1996). Unfortunately, this is a very common practice in object-orientation (e.g. Cattel & Barry, 2000; Cluet, 1998; Hua & Tripathy, 1994). Often the attributes containing the identifiers of the objects are called complex attributes (Lee & Lee, 1998) or object-valued attributes (Pazzi, 1999). However, this kind of implementation makes the traversal of complex entities difficult in an order other than the established one.

Although the popular object-oriented modeling language UML (Rumbaugh et al., 1999) distinguishes the modeling of complex entities from the modeling of other relationships, it falls short in modeling several essential details. It is important to specify all constraints, which complex entities must satisfy. Civello (1993) discusses constraints for their representation (see also Halper et al., 1998; Motschnig-Pitrik & Kaasböll, 1999). Likewise the object-oriented modeling methods OMT (Rumbaugh et al., 1991) and UML do not deal with the inheritance of properties in complex objects. In the is-a relationship the inheritance mechanism is always downward, i.e., all properties of an entity type are also properties of its subentity types. The inheritance mechanism in complex entities may be both downward and upward (inheritance happens from a part to its composite entity). For example, if a car obtains its color from the color of its body then the upward inheritance appears. If the date of an article is the date of the newspaper publishing it we have downward inheritance. Different kinds of inheritances within complex entities are discussed in (Halper et al., 1993; 1998).

Both constructor-oriented formalisms and description logics have been proposed for the exact representation of complex entities. The former emphasize structural aspects among entities. There are both value-oriented (Riet, 1998) and object-oriented (Bancilhon & Khoshafian, 1986) constructor-based formalisms. Description logics emphasize logic-based representation and reasoning. Description logic systems have been extended to represent complex entities (Rousset & Hors, 1996; Lambrix & Padgham, 2000).

2.2. Query languages for manipulating complex entities

In current query languages proposed for the manipulation of complex entities the user must know what entities/entity types (s)he manipulates. In many applications (e.g. the processing of hierarchical documents) it is impossible to find one stable structure suitable to all user needs. Therefore the restructuring capability of query languages is necessary in these applications. However physical assemblies tend to possess a stable structure but need a greater analyzing power than contemporary query languages offer. Further, these languages are difficult to use, as argued below.

There are several SQL-like language proposals based on the NF^2 relational model for manipulating complex entities. These languages have the value-oriented origin. Niemi and Järvelin (1995) give a thorough survey on these languages and analyze query formulation difficulty in them. Typically, in addition to the conventional SQL specification, users are required to master both the semantics of the restructuring operations and the design of large nested expressions where restructuring expressions are embedded in conventional SQL expressions. Niemi and Järvelin (1995; Järvelin and Niemi, 1995; 1999) introduce a truly declarative query language minimizing end-user effort in query formulation. However, from the viewpoint of manipulating physical assemblies the proposed language has several disadvantages. For example, it lacks primitives for analyzing the intensional level of complex entities.

It is important in the object-oriented approach that complex entities can be manipulated bidirectionally (see e.g. Halper et al., 1994), i.e. forward and backward traversal of complex entities is needed. In object-orientation two basic alternatives have been proposed for this. One is based on applying methods of component entity types (e.g. Halper et al., 1994). In these alternatives the user controls the use of methods through hierarchy levels of a complex entity, i.e. it requires programming skills. In the second alternative complex attributes are allowed. If a complex entity is implemented in a unidirectional way, then the expression of forward traversal is straightforward but the expression of backward traversal is troublesome and requires procedural thinking (Lee & Lee, 1998). When a complex entity is implemented bidirectionally, there is no difference between backward and forward traversals (e.g., Cattel & Barry, 2000). However, the synchronization of both traversals is very demanding (Lee & Lee, 1998).

Although considerable progress in developing object-oriented SQL-like query languages has taken place they are not yet suitable for lay users. We discuss this by using OQL (Cluet, 1998), a typical object-oriented query language, as an example. Although the user need not master actual algorithmic programming (s)he must master many aspects of object-orientation such as object identity, class, attribute, method, inheritance, literal etc. In addition, in the OQL the user must combine iterators (e.g., select-from-where, grouping and sorting iterators) with each other. Therefore (s)he must also understand how a variable within an iterator will be instantiated with different entities until all entities belonging to the entity type to which the variable refers have been processed. In complex queries the user must nest iterators whereby there may be several instantiations of one variable for one instantiation of another variable. Thus the OQL user must think iteratively. In object-oriented query languages complex entities are typically constructed by applying available constructors like the set, list, tuple and tree constructors on atomic data types. Dar and Agrawal (1993) argue that constructors make query formulation quite complicated for lay users. This is particularly true when users must nest constructors within each other.

QAL is a functional object-oriented query language supporting the manipulation of complex entities (Savnik et al., 1999). QAL supports typical database queries such as 'retrieve the values of selected attributes from any nesting levels of complex entities'. In addition, QAL is able to query the intensional level and to express the connection between the extensional and intensional levels as well. These are useful features in a query language intended for physical assemblies. However, QAL has three disadvantages. First, the QAL user must know the entities of interest and give path expressions leading to them. More powerful primitives are needed for analyzing complex structures because in many queries one needs to find entities / entity types which satisfy specific properties without knowing where they reside in a complex entity. Second, the user must master relevant constructors. Third, QAL does not contain any mechanism to combine text retrieval with the manipulation of complex entities.

2.3. Goals for an advanced query language for complex entities

Physical assemblies have their own special characteristics and needs. A physical assembly may consist of a huge number of components at several hierarchy levels. The same component type can reside in several different constructs. Therefore the management of structural aspects needs particular support. Next we describe two practical situations where such support is needed. First, assume that a user must change some parts of a complex entity based on their age for maintenance reasons. It is likely that the user does not know where precisely such decaying components reside. Second, assume that the production process of a company is improved by changing some tools and methods. Now one should find all components affected by this change. If information on tools and methods related to components has been stored in documents, one must manipulate complex entities and their documents at the same time.

The examples above demonstrate that a query language for complex entities should support queries where the user cannot express which entities or entity types (s)he should manipulate. Likewise primitives to manipulate complex entities and their documentation together are needed. Our purpose is to offer such a query language for advanced manipulation of physical assemblies. This query language has the following goals:

- The degree of declarativity must be much higher than in the contemporary query languages for complex entities. Therefore the user need not master programming (e.g., iterative or recursive thinking). The user also should not need to specify complex nested expressions.

- The user need not apply constructors in queries.
- It is possible to express extensional, intensional and combined extensional-intensional queries. The language must also contain primitives connecting the intensional and extensional levels in a straightforward way.
- The language must support both forward and backward traversal in complex entities. This support must be general so that the user may refer to any component of a given entity / entity type at any hierarchy level.
- The query language must be able to process textual documentation of complex entities. Therefore text retrieval must be integrated with the manipulation of complex entities.

3. A database for complex entities and related documents

Our system contains two components: a database component consisting of physical assemblies and a document database consisting of documents related to entity types in the physical assemblies. First we consider how a physical assembly is represented as a complex entity and next we describe how documents are associated with physical assemblies.

3.1. The representation of a complex entity

In the representation of a complex entity we have combined the advantages of NF^2 relations and object-oriented approaches. Niemi and others (2002) discuss these advantages in detail. Figure 1 presents our sample database. It contains only three complex entities: one tricycle and two bicycle entities. It is a user-oriented view for the data and we assume that its information content is self-explanatory except for the columns “oid”, “W.” and “M.”. The “oid” columns express the identities of entities and the user need not be aware of them whereas “W.” and “M.” are the abbreviations for the attributes Weight and Material, respectively.

TRICYCLE																															
oid	Price	W.	FRAME				SADDLE			STEERING													REAR								
			oid	Frame-No	M.	W.	oid	Pad	W.	oid	W.	FRONT AXLE		HANDLEBAR		PEDALS			WHEEL				oid	W.	REAR AXLE			WHEEL			
												oid	W.	oid	W.	oid	Diam.	W.	oid	Diam.	R_type	W.			oid	Diam.	W.	oid	Diam.	R_type	W.
o12	100	10	o9	4566545	steel	4	o10	plastic	1	o11	3	o5	0.5	o6	1	o7	5	0.5	o8	8	united	1	o4	2	o1	0.5	1	o2	6	united	0.5
																												o3	6	united	0.5

BICYCLE																																		
oid	Price	W.	FRAME				DRIVE GEAR											SADDLE			STEERING										WHEEL			
			oid	Frame-No	Mat.	W.	oid	C_type	W.	CHAIN		CHAIN RING			PEDALS			oid	Pad	W.	oid	W.	FRONT AXLE		HANDLEBAR		WHEEL				oid	Diam	R_type	W.
										oid	W.	oid	Diam	W.	oid	Diam	W.						oid	W.	oid	W.	oid	Diam	R_type	W.				
o37	500	18.2	o27	8265	steel	10	o28	1-speed	1.7	o13	0.5	o14	10	0.5	o16	16	0.5	o29	leather	0.5	o30	4	o17	1	o18	1	o19	26	spoke	2	o31	26	spoke	2
												o15	4	0.2																				
o38	400	16.2	o32	43285	steel	8	o33	1-speed	1.7	o20	0.5	o21	9	0.5	o23	14	0.5	o34	plastic	0.5	o35	4	o24	1	o25	1	o26	20	spoke	2	o36	20	spoke	2
												o22	4	0.2																				

Fig. 1. Extensional level of our sample database.

3.2. Documents related to complex entities

In many applications one needs to associate documents with entity types in physical assemblies. A document is attached to each entity type and it contains information common to all entities belonging to this type. Therefore our document database contains one document for each entity type in the database. In the context of physical assemblies these documents can be, e.g., instructions for assembly or service of parts. In our system the documents are represented as XML documents (http://www.w3.org/XML/). Note that in some applications documents could also be associated with entities, too – e.g., giving their use and maintenance history.

Thus our sample document database contains XML documents for each entity type in TRICYCLE and BICYCLE. These documents have the same structure. The tools, which are needed in disassembling an entity belonging to an entity type are listed between the tags <tool> and </tool>. The text identified by the tags <disassembly_instructions> describes the disassembly instructions of the entities of this entity type. The skills, which are needed in the disassembly, are indicated by the tags <skill_requirements>. Safety instructions for the disassembly are coded by the tags <safety_requirements>. The string nil indicates missing information. The Appendix presents the documents for BICYCLE. Sample query evaluations will be based on this document.

4. Query language for manipulating complex entities

Users of contemporary query languages must know which entities they manipulate. They must also know exactly the structure of complex entities and specify navigation in this structure. An advanced query language should also support queries where the user does not know these aspects in detail. For that reason our query language offers primitives for analyzing both the intensional and extensional levels and for moving between them.

4.1. The notion of variable in query formulation

In our approach, users may refer to unknown factors in their queries. For example, entity types, their properties, entities, documents, subdocuments, and values of properties may be such unknown factors. Each variable in a query refers to some unknown factor. A variable may be associated with a construct at the intensional or the extensional level. The role of query primitives is to express semantic associations among variables. This guarantees a declarative query language. Query processing is responsible for finding the values of variables satisfying the criteria given in primitives.

The notion of variable in our query language was borrowed from deductive databases (e.g., Liu, 1999). A variable starts by an uppercase letter whereas constants are numbers or strings, which start by lowercase letters. Query primitives are connected by commas or semicolons, which indicate logical conjunctions and disjunctions, respectively. If the same variable appears in two or more query primitives it is a shared variable. Such a variable must be instantiated to the same value in all query primitives. From the user viewpoint, query formulation consists of combining query primitives containing variables.

4.2. *The intensional primitives*

The intensional level query primitives are as follows:

(1) Arg1 *is_whole_type_of_type* Arg2
(2) Arg1 *is_part_type_of_type* Arg2
(3) Arg *is_top_type*
(4) Arg *is_basic_type*
(5) Arg1 *is_property_of* Arg2
(6) Arg1 *is_path_to* Arg2

In these primitives, Arg, Arg1 and Arg2 are arguments that the user specifies when applying primitives. The user may explicate an argument or refer to it by a variable. In the explicit specification the user gives a constant possibly found in some complex entity. Thus, for example, the names of known entity types and properties are expressed with constants. The primitive (1) expresses that Arg1 is an immediate or indirect composite entity type for the entity type Arg2. The primitive (2) is the opposite, i.e., Arg1 is an immediate or indirect component of the entity type Arg2. The primitives (3) and (4) are used to find an entity type (Arg) that has no composite or component entity type in the complex entities considered, respectively. The primitive (5) refers to any property (Arg1) of the entity type (Arg2). The primitive (6) forms a path Arg1 that starts from a top entity type and leads to the entity type Arg2.

Let us assume that we apply primitives in the database given in Figure 1. For example, a composite entity type for the entity type PEDALS may be found by the expression X *is_whole_type_of_type* pedals. This means that X may be instantiated to the values tricycle, steering, bicycle, drivegear. The expression Z *is_whole_type_of_type* Y finds all pairs Z and Y where Y is an entity type and Z its any (perhaps indirect) composite entity type. The primitive B *is_top_type* instantiates the variable B to the values tricycle and bicycle. In the primitive P *is_property_of* frame, possible instantiations for the variable P are frame_no, material, and weight. The primitive P *is_property_of* E refers to any property (P) of any entity type (E).

4.3. *The extensional primitives*

Our query language contains the following primitives for manipulating information at the extensional level:

(7) Arg *is_basic_entity*
(8) Arg *is_top_entity*
(9) Arg1 *is_whole_entity_of_entity* Arg2
(10) Arg1 *is_part_entity_of_entity* Arg2

The primitives (7), (8), (9) and (10) correspond to the primitives (4), (3), (1) and (2) of the intensional level. For example, assume that the variable Z has been instantiated to the entity with object identifier o16, i.e. it is an entity of type PEDALS in the complex entity BICYCLE. Now the variable Y in the primitive Y *is_whole_entity_of_entity* Z may be instantiated to the entities with identifiers o28 (DRIVE GEAR) and o37 (BICYCLE).

4.4. *The primitives for connecting the intensional and extensional levels*

It is important to be able to analyze the structure of complex physical assemblies and to manipulate data based on this analysis. The following primitives for connecting the intensional and extensional levels. are therefore needed:

(11) Arg1 *:* Arg2
(12) Arg1 *is_instance_of* Arg2
(13) Arg1 *is_whole_entity_of_type* Arg2
(14) Arg1 *is_part_entity_of_type* Arg2

The primitive (11) refers to the value of a property (Arg2) of an entity (Arg1), i.e., Arg1 is at the extensional level whereas Arg2 belongs to the intensional. Assume that the variable Y has been instantiated to the entity with object identifier o15 (of type CHAIN RING). Now the primitive Y*:* diam means the value 4. The primitive (12) can be used to refer to any entity (Arg1) belonging to entity type given in Arg2. For example, in the primitive Z *is_instance_of* drivegear the variable can be instantiated to the entities with object identifiers o28 and o33. Primitives (13) and (14) refer to a composite and component entity (Arg1) of the entity type given in Arg2, respectively.

4.5. *The primitives for integrating complex entities with documents*

Present query languages for manipulating complex entities do not offer mechanisms for formulating queries involving information in text documents describing entity types. Our query language contains primitives for manipulating structured text documents together with related complex entities:

(15) Arg1 *is_doc_of* Arg2
(16) Arg1 *is_sub_doc* Arg2
(17) Arg1 *contains* Arg2

The primitive (15) expresses that Arg1 is the document attached to the entity type Arg2. For example, in the primitive X *is_doc_of* frame the variable X denotes the document attached to the entity type frame (see Appendix).

Our text documents are structured into four parts by the tags <tools>, <disassembly_instructions>, <skill_requirements> and <safety_requirements>. Each part can be seen as a subdocument. In the primitive (16) the argument Arg2 has the form [Tag, Doc_name]. This primitive assigns to Arg1 the subdocument, which has been structured by Tag in the document Doc_name. The primitive (17) expresses that the string Arg2 is included in the document or subdocument Arg1. A string is expressed between apostrophes.

5. Sample queries

In our query language a query has the following structure:

<form of result> where <primitive sequence>.

The construct <*form of result*> expresses the content of the result. It has the form *res(x1,x2, …, xn)* where *res* is the name selected for the result. The components *x1, x2, …, xn* are the columns of the result. The string *where* is a reserved word separating the form from the conditions the result must satisfy. In the construct <*primitive sequence*> primitives are connected by conjunction (comma) or disjunction (semicolon). Typically the components *x1, x2, …, xn* of the result and primitives in <*primitive sequence*> may contain several shared variables. Query processing finds the variable instantiations in *x1, x2, …, xn,* which satisfy the criteria given.

The database component may contain several complex entities. A background assumption is that the primitives are applied to all complex entities in it. If the user wants to apply primitives only to some complex entities (s)he can restrict them by the expression *apply_to [e1 , …, en]*. Here the list *[e1, …, en]* expresses the scope of primitives. For example, the expression *apply_to [bicycle]* means that the primitives are applied only within the entity BICYCLE. This expression is a part of <*primitive sequence*>.

5.1. Extensional queries

Most present query languages support only extensional queries of this type, where the result consists only of extensional level data. The background assumption of extensional queries is that the user knows exactly what information (s)he needs from complex entities. Unlike in the present approaches, in our language the user need not specify paths in complex entities. In Sample Query 1 the user wants to know the prices and weights of bicycles and the frame numbers of their frames. We number the lines of our queries in order to indicate their parts.

Sample Query 1

(1) bicycle_info(Bi:weight,Bi:price,Fr:frame_no) *where*
(2) Bi *is_instance_of* bicycle,
(3) Fr *is_instance_of* frame,
(4) Bi *is_whole_entity_of_entity* Fr.

On line (1) the user expresses the form of the result based on the variables used in the where -part. The variable Bi denotes an instance of the bicycle type (see (2)) whereas the variable Fr (see (3)) refers to any entity of type frame. In (4) (s)he specifies that the specific frame Fr must be a part of the bicycle Bi. Figure 2 gives the query result based on the information in Figure 1.

bicycle_info		
weight	**price**	**frame_no**
16.2	400	43285
18.2	500	8265

Fig. 2. The result of Sample Query 1.

Sample Query 1 demonstrates how a query can be specified simply although the result contains information from several hierarchy levels of complex entities. Also selection conditions could be specified simply in the *where* - part of our query language.

5.2. Intensional queries

In intensional queries the result contains only intensional information. Conventional query languages do not support intensional queries. Intensional queries are necessary in analyzing the structure of complex entities. Our query language offers powerful primitives for structural analysis. These primitives allow the formulation of queries without knowing the structure of complex entities exactly. Sample Query 2 demonstrates this. Assume that the user is interested in the physical assembly TRICYCLE and (s)he wants to know which basic component types (i.e., these component types have no parts) are associated with the entity type REAR. In addition (s)he wants to know the properties of these basic component types.

Sample Query 2

(1) basic_info(Btype,Prop) *where*
(2) *apply_to* [tricycle],
(3) Btype *is_basic_type*,
(4) rear *is_whole_type_of_type* Btype,
(5) Prop *is_property_of* Btype.

Through (2) and (3) Btype stands for any basic component type in the complex entity TRICYCLE. Line (4) specifies that any basic component type must be an immediate or indirect part of the entity type REAR. The variable Prop stands for any property of the basic component type. The result is in Figure 3.

basic_info	
Btype	**Prop**
rear_axle	diam
rear_axle	weight
wheel	diam
wheel	r_type
wheel	weight

Fig. 3. The result of Sample Query 2.

The result of Sample Query 2 is produced by manipulating only information at the intensional level. Sometimes intensional queries may also require the manipulation of the extensional level. There often is a need to find component types of complex entities, which contain specific values. For example, if some material (at the extensional level) has been found hazardous, it would be nice to find entity types, which contain this material. The user may want to find, for example, all component types of BICYCLE, which may contain plastic. Because of space limitations we leave the formulation of this query as an exercise.

5.3. Combined extensional-intensional queries

Queries producing both extensional and intensional information are *combined extensional-intensional queries.* Such queries are usual in physical assemblies when the user wants to know both the result of structural analysis and the information related to the corresponding entities. In Sample Query 3 the user wants to find all component entity types of the entity type DRIVE GEAR of bicycles and the corresponding component entities. In the result he is interested only in the values of the properties Diameter and Weight of the component.

Sample Query 3

(1) result(Comp,Inst:diam,Inst:weight) *where*
(2) *apply_to* [bicycle],
(3) Comp *is_part_type_of_type* drivegear,
(4) Inst *is_instance_of* Comp.

In (3) the variable Comp stands for any component entity type of DRIVE GEAR. In (4) the variable Inst is instantiated to an entity belonging to the entity type represented by Comp. The result is in Figure 4. The entity type CHAIN does not have the property Diam (see Figure 1) and thus this property has the value 'null'.

result		
Comp	**diam**	**weight**
chain	null	0.5
chainring	9	0.5
chainring	10	0.5
chainring	4	0.2
pedals	14	0.5
pedals	16	0.5

Fig. 4. The result of Sample Query 3.

5.4. Queries for integrating complex entities with their documents

Integration of complex entities and their documentation is needed in two query types. There are queries only finding information in complex entities but using documents in the selection of this information. Moreover, there are queries finding documents attached to entity types. Sample Query 4 and Sample Query 5 demonstrate these two query types. Information in Appendix is needed in their evaluation. Sample

Query 4 finds those component entity types of BICYCLE the disassembly of which requires tongs or gloves.

Sample Query 4

(1) result(Part) where
(2) Part is_part_type_of_type bicycle,
(3) Part_Doc is_doc_of Part,Tools is_sub_doc [tools,Part_Doc],
(4) Safe is_sub_doc [safety_requirements,Part_Doc],
(5) (Tools contains \`tongs\`; Safe contains \`gloves\`).

The variable Part_Doc refers to the document attached to any component type of BICYCLE. The variables Tools and Safe stand for its subdocuments. They contain information on tools and safety aspects. In (5) we test that either the subdocument on tools contains the string \`tongs\` or the subdocument on safety instructions contains the string \`gloves\`. The result of this intensional query is in Figure 5.

result
Part
drivegear
steering
chain
chainring
front_axle

Fig. 5. The result of Sample Query 4.

Sample Query 5 finds composite entity types containing as indirect or immediate parts the entity types CHAIN and PEDALS. In addition, it retrieves the subdocuments of these entity types containing the disassembly instructions.

Sample Query 5

(1) type_subdoc(Type,SubDoc) where

(2) Type *is_whole_type_of_type* chain,

(3) Type *is_whole_type_of_type* pedals,

(4) Doc is_doc_of Type,

(5) SubDoc *is_sub_doc* [disassembly_instructions,Doc].

By using the shared variable Type in (2) and (3) we specify that Type is a common immediate or indirect composite entity type for CHAIN and PEDALS. The variable SubDoc is instantiated to the subdocument of each entity type of this kind, which provides the disassembly instructions. The result is in Figure 6.

type_subdoc

Type	**SubDoc**
bicycle	The saddle is separated as follows: Loosen the screw using an adjustable wrench and draw the saddle from the frame. If the saddle does not move then hit lightly the downward side of the saddle by a hammer. When separating the steering it has to be partially disassembled. However, when separating drive gear it has to be fully disassembled. See their disassembly instructions.
drivegear	First, the rear wheel has to be released using an adjustable wrench. Next, the chain is separated. The chain ring in the wheel is loosened using gear wrench. When separating pedals and the front chain ring the boss must be disassembled.

Fig. 6. The result of Sample Query 5.

6. Discussion

In contemporary database query languages queries are represented mainly with intensional elements, which produce answers composed entirely of extensional information. This also applies to most query languages intended for manipulating complex entities. However, increasing attention has recently been paid to the possibility of supporting intensional queries. Intensional queries increase the expressive power and offer more intelligent query languages (Motro, 1994). In this paper we deal with complex entities called physical assemblies. Large physical assemblies may consist of a large number (possibly thousands) of parts. In practice the users are not able to master such structures in detail. However, most query languages expect the user to know exactly the structure of complex entities in query formulation. Through intensional queries and intensional query primitives of our query language one may refer to unspecified structural information and manipulate it in physical assemblies. This requires the capability of analyzing structural aspects of physical assemblies and this kind of mechanism is necessary in any advanced query language for managing the complexity of physical assemblies. Our sample queries showed that the primitives of our query language remove the explicit specification of navigation from the user.

We have presented several sample queries showing that extensional, intensional and combined extensional-intensional queries are needed in the context of physical assemblies. In order to support queries of different types our query language contains primitives connecting the intensional and extensional levels. Through these primitives the user can easily transfer data manipulation from the extensional level to the intensional level and vice versa. For example, the user may first analyze entity types satisfying given structural criteria and then manipulate entities belonging to these entity types. Our sample queries showed that often both forward and backward traversal in complex entities is needed. Such traversals may occur both at the extensional and the intensional level. In contemporary object-oriented approaches the integration of forward and backward traversal is very troublesome (Lee & Lee, 1998). Although our approach is also object-oriented, forward and backward traversal is very straightforward. This is because our representation of physical assemblies combines the strengths of the value-oriented and object-oriented representations. As in NF^2

relations we store subentities in the entities which contain them. Therefore forward/backward traversal among subentities is based on nesting. In other words link manipulation, common in the object-oriented approaches, is unnecessary. As Junkkari (2001) has shown, our indexing mechanism enables one to find all component or composite entities or entity types related to a specific entity or entity type.

Physical assemblies often have several instructions associated with their parts such as assembly, disassembly and service instructions. Such information is usually represented as text documents. As our sample queries exemplified, information both from complex entities and their related documents is often needed. Our query language contains primitives for referring to a document or subdocument related to a specific entity type. Further there is a primitive for investigating the content of text documents. The extensions to more comprehensive IR primitives are obvious. By using shared variables in query components the user can express semantically related information intuitively and compactly.

There are several query language proposals for complex entities. Some have prototype implementations. A prototype implementation of the present language was programmed in Prolog++ (Moss, 1994), which combines logic programming and object-oriented programming into one homogeneous deductive object-oriented programming framework. This kind of framework was ideal for the implementation of our query language because it supports, in addition to object-oriented features, the notion of variable used in our query language.

7. Conclusions

A powerful and declarative query language is needed for manipulating complex entities - especially physical assemblies. In contemporary query language approaches the user needs to know exactly all information in complex entities. For example, the user is expected to specify navigation paths leading to the data of interest. This is an unrealistic assumption in such physical assemblies, which consist of a large number of parts. Therefore the primitives of our query language were designed so that the user can manipulate data and structures which (s)he does not know. For example, (s)he may easily refer to any composite or component entity / entity type of a specific entity / entity type. This feature facilitates considerably the specification of forward and backward traversal in complex entities in comparison to the other languages. Our language contains primitives of three kinds. In addition to extensional and intensional query primitives, it contains primitives for transferring data manipulation from the extensional level to the intensional level and vice versa. These primitives support the formulation of intensional and combined extensional-intensional queries, in addition to conventional extensional queries. Our sample queries demonstrated that this support has a great practical significance. Unlike contemporary query languages, our query language also contains primitives for manipulating structured text documents related to parts of physical assemblies. The possibility to use information in documents increases the expressive power of our language. This feature is necessary in many applications of physical assemblies. We borrowed the notion of variable for our query language from deductive databases and showed that this makes query formulation intuitive and compact.

Acknowledgement

This research was supported by the Academy of Finland under the grant number 52894.

References

Artale, A., Franconi, E., Guarino, N. & Pazzi, L. (1996). Part-whole relations in object-centered systems: An overview. *Data & Knowledge Engineering*, 20(3), 347-383.

Bancilhon, F. & Khoshafian, S. (1986). A calculus for complex objects. In Nori, A. (Ed.), *Proceedings of the 5th ACM Symposium on Principles of Database Systems*, (pp. 53-59). Cambridge, MA: ACM Press.

Barbier, F., Henderson-Sellers, B., Opdahl, A.L. & Gogolla, M. (2000). The whole-part relationship in the unified modeling language: a new approach. In Siau, K. & Halpin, T. (eds.), *Unified Modeling Language: Systems Analysis, Design, and Development Issues*: (pp. 1-20). Hershey, PA: Idea Group Publishing.

Cattel, R. & Barry, D. (2000). *The object data standard: ODMG 3.0.* San Francisco: Morgan Kaufmann.

Civello, F. Roles for composite objects in object-oriented analysis and design. In Andreas P. (Ed.), *Proceedings of the 8th Conference on Object-Oriented Programming Systems, Languages, and Applications*, (pp. 376-393). WA: ACM Press.

Cluet, S. (1998). Designing OQL: Allowing objects to be queried. *Information Systems*, 23(5), 279 - 305.

Dar, S. & Agrawal, R. (1993). Extending SQL with generalized transitive closure. *IEEE Trans. Knowledge and Data Engineering*, 5(5), 799 – 812.

Halper, M., Geller, J. & Perl, Y. (1993). Value propagation in object-oriented database part hierarchies. In Bhargava, B. K., Finin, T. W. & Yesha, Y. (Eds.), *Proceedings of the 2nd International Conference on Information and Knowledge Management*, (pp. 606-614). WA: ACM Press.

Halper, M., Geller, J., Perl, Y. & Klas, W. (1994) Integrating a part relationships into an open OODB system using metaclasses. In Nicholas, C. K. & Mayfield, J. (Eds.), *Proceedings of the 3rd International Conference on Information and Knowledge Management*, (pp. 10-17). Gaithersburg, MD: ACM Press.

Halper, M., Geller, J. & Perl, Y. (1998). An OODB part-whole model: Semantics, notation and implementation. *Data & Knowledge Engineering*, 27(1), 59-95.

Hua, K. & Tripathy, C. (1994). Objects skeletons: An efficient navigation structure for object-oriented database systems. *Proceedings of the10th International Conference on Data Engineering*, (pp. 508-517). Houston, TE: IEEE Computer Society.

Junkkari, M. (2001). *The systematic object-oriented representation for managing intensional and extensional aspects in modeling part-of relationships.* Tampere, Finland: Department of Computer and Information Sciences, University of Tampere.

Järvelin, K., Ingwersen, P. & Niemi, T. (2000). A user-oriented interface for generalized informetric analysis based on applying advanced data modelling techniques. *Journal of Documentation*, 56(3), 250-278.

Järvelin, K. & Niemi, T. (1995). An NF^2 relational interface for document retrieval, restructuring and aggregation. In Fox, E. A., Ingwersen, P. & Fidel R. (Eds.), *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, (pp. 102-109) Seattle, WA: ACM Press.

Järvelin, K. & Niemi, T. (1999). Integration of complex objects and transitive relationships for information retrieval. *Information Processing & Management*, 35(5), 655 - 678.

Kim, W., Banerjee, J. & Chou, H.-T. (1997). Composite object support in an object-oriented database system. *Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications* (pp. 118-125), Atlanta, Georgia: ACM Press.

Lambrix, P. & Padgham, L. (2000). Conceptual modeling in a document management environment using part-of reasoning in description logics. *Data & Knowledge Engineering*, 32(1), 51-86.

Lee, W.-C. & Lee, D. (1998). Path Dictionary: A new access method for query processing in object-oriented databases. *IEEE Trans. Knowledge and Data Engineering*, 10(3), 371 – 388.

Liu, M. (1999). Deductive Database Languages: Problems and Solutions. *ACM Computing Surveys,* 31(1), 27-62.

Liu, L., Halper, M., Gu, H., Geller, J. & Perl, Y. (1996). Modeling a vocabulary in an object-oriented database. *Proceedings of the 5th International Conference on Information and Knowledge Management*, (pp. 179-188). Rockville, MD: ACM Press.

Moss, C. (1994). *Prolog++ the power of object-oriented and logic programming*. Cambridge: Addison-Wesley.

Motro, A. (1994). Intensional answers to database queries. *IEEE Trans. Knowledge and Data Engineering,* 6(3), 444 – 454.

Motschnig-Pitrik, R. & Kaasböll, J. (1999). Part-whole relationship categories and their application in object-oriented analysis. *IEEE Trans. Knowledge and Data Engineering,* 11(5), 779 – 797.

Niemi, T., Junkkari, M., Järvelin, K. & Viita S. (2002). *Advanced Query Language for Manipulating Complex Entities*, Report A-2002-17, Department of Computer and Information Sciences, University of Tampere, Finland.

Niemi, T. & Järvelin, K. (1995). A straightforward NF^2 relational interface with applications in information retrieval. *Information Processing & Management,* 31(2), 215 - 231.

Pazzi, L. (1999). Implicit versus explicit characterization of complex entities and events. *Data & Knowledge Engineering*, 31(2), 115-134.

Renguo, X., Dillon, T. S., Rahayu, W., Chang, E. & Gorla, N. (2000). An indexing structure for aggregation relationship in OODB. In Ibrahim, M.T., Küng, J. & Revell, N. (Eds.). *Proceedings of the 11th International Conference on Database and Expert Systems Applications*, (pp. 21-30). London, UK: Springer.

van de Riet, R.P. (1998). Complex objects: theory and practice from a data- and knowledge engineering perspective, as seen in and from Yellowstone Park. *Data & Knowledge Engineering*, 25(1-2), 217-238.

Roth, M., Korth, H. & Silberschatz, A. (1988). Extended algebra and calculus for nested relational databases. *ACM Trans. Database Syst.* 13(4), 389 - 417.

Rousset, M-C. & Hors, P. (1996). Modeling and verifying complex objects: A declarative approach based on description logics. In Wahlster, W. (Ed.): *Proceedings of the 12th European Conference on Artificial Intelligence*, (pp. 329-332). Budapest, Hungary: John Wiley and Sons.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W. (1991). *Object-oriented modeling and design.* Prentice Hall.

Rumbaugh, J., Jacobson, I. & Booch G. (1999) *The unified modeling language reference manual*. Reading: Addison-Wesley.

Sacks-Davis, R., Kent, A., Ramamohanarao, K., Thom, J. & Zobel, J. (1995). Atlas: A nested relational database system for text applications. *IEEE Trans. Knowledge and Data Engineering*, 7(3), 454 – 470.

Savnik, I., Tari, Z. & Mohoric, T. (1999). QAL: A query algebra of complex objects. *Data & Knowledge Engineering*, 30(1), 57-94.

Ullman, J. (1988). *Principles of database and knowledge-base systems*, vol. I. Rockville: Computer Science Press.

Urtado, C. & Oussalah, C. (1998). Complex entity versioning at two granurality levels. *Information Systems*, 23(3/4), 197 - 216.

Wand, Y., Storey, V. C. & Weber, R. (1999). An ontological analysis of the relationship construct in conceptual modeling. *ACM Trans. Database Systems*, 24(4), 494 - 528.

Winston, M., Chaffin, R. & Hermann, D. (1987). A taxonomy of part-whole relations. *Cognitive Science*, 11(4), 417-444.

Zobel, J., Thom, J. & Sacks-Davis, R. (1991). Efficiency of nested relational document database systems. In Lohman, G.M., Sernadas, A. & Camps R. (Eds.), *Proceeding of the 17th International Conference on Very Large Data Bases*, (pp. 91-102). Barcelona, Catalonia: Morgan Kaufmann.

APPENDIX

The sample document database consists of the XML documents related to entity types in TRICYCLE and BICYCLE. Only the latter document is given below. Here […] indicates suppressed components.

Doc. of BICYCLE:
<doc>
<tools>
screwdriver, adjustable wrench, cotter pin extractors, tongs, hammer
</tools>

<disassembly_instructions>
The saddle is separated as follows: Loosen the screw using an adjustable wrench and draw the saddle from the frame. If the saddle does not move then hit lightly the downward side of the saddle by a hammer. When separating the steering it has to be partially disassembled. However, when separating drive gear it has to be fully disassembled. See their disassembly instructions.
</disassembly_instructions>

<skill_requirements>
High technical skills
</skill_requirements>

<safety_requirements>
Protective gloves are required in disassembling the drive gear.
</safety_requirements>
</doc>

Doc. of DRIVE GEAR:
<doc>
<tools>
gear wrench, adjustable wrench
</tools>

<disassembly_instructions>
First, the rear wheel has to be released using an adjustable wrench. Next, the chain is separated. The chain ring in the wheel is loosened using gear wrench. When separating pedals and the front chain ring the boss must be disassembled.
</disassembly_instructions>

<skill_requirements>
Releasing the chain requires low technical skills. Separating of pedals and chain rings requires the skills of a cycle mechanic.
</skill_requirements>

<safety_requirements>

Protective gloves are required in disassembling the chain and chain rings.
</safety_requirements>
</doc>

Doc. of STEERING:
<doc>
<tools>
adjustable wrench, cotter pin extractors, tongs
</tools>

<disassembly_instructions>
The disassembly of the steering starts by separating of the wheel from the fork of the bicycle. For this the nuts must be loosened using adjustable wrench. The handlebars are separated by turning the bolt on top of the holder by an adjustable wrench. The axle is separated by turning the guard using tongs. After that the cotter must be disconnect using cotter pin extractors. Finally the axle is drawn out from the shaft tunnel.
</disassembly_instructions>

<skill_requirements>
Low technical skills.
</skill_requirements>

<safety_requirements>
nil
</safety_requirements>
<doc>

Doc. of FRAME:
[…]
Doc. of CHAIN:
<doc>
<tools> nil *</tools>*

<disassembly_instructions> nil *</disassembly_instructions>*

<skill_requirements> nil *</skill_requirements>*

<safety_requirements>
Protective gloves are required.
</safety_requirements>
</doc>

Doc. of CHAIN RING:
<doc>
<tools> nil *</tools>*

<disassembly_instructions> nil *</disassembly_instructions>*

<skill_requirements> nil *</skill_requirements>*

<safety_requirements>
Protective gloves are required.
</safety_requirements>
</doc>

Doc. of SADDLE:
[…]

Doc. of FRONT AXLE:
<doc>
<tools> nil *</tools>*

<disassembly_instructions> nil *</disassembly_instructions>*

<skill_requirements> nil *</skill_requirements>*

<safety_requirements>
Protective gloves are required.
</safety_requirements>
</doc>

Doc. of HANDLEBAR:
[…]

Doc. of PEDALS:
[…]

Doc. of WHEEL:
[…]