

Architecture of a Semantic Data Integration System Based on a Semantically Complete Model and a Semantically Complete Query Language

V. V. Ovchinnikov

Lipetsk State Technical University, Lipetsk, Russia
e-mail: ovch@lipetsk.ru

Received August 25, 2005

Abstract—The availability of numerous sources of structured data on the Internet poses the problem of their integration into a unified information space just as the unstructured data and weakly structured data sources are integrated in the framework of the WWW. The main requirement for such an information space is the simplicity of operation for the users that are not trained IT experts. The architecture of a system of semantic integration of distributed and heterogeneous data sources is proposed in the framework of a unified semantic access interface. The semantic nature of this interface lies in the fact that one can interact with such a system in terms of the concepts of the application domain completely ignoring the implementation details of the systems being integrated. The proposed architecture simplifies the users' work, the integration process, and the development of user forms with a rich functionality including a semantic navigation between the forms. A distinctive feature of this architecture is that the system integration and the development of user forms are performed declaratively in the interactive mode without programming. The simplification of the users' work with the system is achieved due to some special properties of the semantically complete model (SCM) and of the semantically complete query language (SCQL), which provide a basis for the system. A prototype of the system under study is briefly described. The prototype is implemented as a type of the client–server technology based on the SCM–SCQL.

DOI: 10.1134/S0361768806040050

1. INTRODUCTION

The increase in the number of available information sources on the Internet and within various organizations prompts many researchers to seek a solution of the data integration problem, that is, to develop a unified access interface to distributed heterogeneous data sources. By a unified interface, we mean unified requirements for the information structure (the global schema), which is published by the so-called data integration system, and for the structure and syntax of the queries processed by that system. The main tasks performed by the data integration system are as follows: (a) mapping of the local schemata defined at the level of each information system being integrated to the global schema; (b) executing queries on the global schema.

Processing of a query assumes that it is subdivided into the parts corresponding to the local schemata; the parts are translated into the query languages of the systems being integrated; the translated queries are executed within the corresponding systems; and finally the results are consolidated to produce the result of the initial query.

Various methods of data and knowledge modeling can be used to describe the structure of the global and the local schemata. Currently, much attention is given to the use of ontologies [1, 2]. Ontologies are well suited to integration problems because they provide an

abstract description of the terminology of the application domain (AD) but ignore the details of the implementations of the particular information systems. Another advantage of ontologies is the fact that the vocabulary description language RDFS [3] and the ontology description language [4] based on it have been standardized by the W3C. In addition to the ontology-based integration, other approaches to data integration are available in the literature [5–9]. In all the approaches known to the author of this paper, the formulation of the queries on the global schema is relatively complicated, which implies high requirements for the training of the experts involved in the integration project. In this paper, we propose a data integration technology that facilitates the understanding and the formulation of queries without detriment to the expressive power of the query language. Two methods of the consolidation of the local (initial) schemata into a global (target) schema are known: (a) The local schemata are an invariable part of the global schema, and the global schema is a simple combination of the local schemata. (b) The global and the local schemata have an independent contents and are partially mapped one to the other [10].

In the framework of the second method, there are two main approaches to the description of schemata: (a) The local schema is declared as a view on the global

schema (local-as-view approach) [11]. (b) The global schema is declared as a view on the local schemata (global-as-view approach) [12].

There are also intermediate approaches to schema mappings [13–15].

At first glance, the second method of consolidation has a greater modularity because each local schema may be designed independently. However, it turns out on closer examination that the integration of local schemata requires additional efforts that exceed the efforts needed to design a unified global schema in the first method. This fact explains a high activity of researchers in the development and standardization of unified global schemata in various application domains; such global schemata make it possible to minimize the efforts to design and integrate the local schemata.

The existing data integration systems have some common drawbacks that complicate their practical application. These drawbacks are as follows: (a) The user screen forms can be designed only by IT experts that know the details of the integration project implementation. (b) To integrate several information systems, a complex project must be implemented, which requires highly qualified experts acquainted with the implementation of the systems being integrated. (c) Programming of powerful data navigation and filtration mechanisms requires considerable effort from the developers, especially in a distributed and heterogeneous environment. (d) If the structures used to store the data evolve, all the applications that use the changed structures must be modified.

Among the data integration systems, the semantic integration systems (see, e.g., [5]) deserve a special attention. A specific feature of such systems is that the model used to construct the global schema (here and in what follows, we use the term *model* to mean a modeling method, and the term *schema* is used to mean the modeling result) does not include mechanisms aimed at controlling the implementation of the target system that are not required to represent the semantics of the AD. The sole purpose of designing the global schema in such systems is to give the most adequate description of the semantics of the AD (the concepts of the AD and the relationships between them) without taking into account the efficiency of the target system implementation. The fact that the efficiency issues are completely ignored leads to high requirements for the flexibility of mapping the decisions concerning data storage into the global schema (through the local schemata).

Any model that ignores the implementation details of the target system may be used as a basis for a semantic data integration system (SDIS). In addition to ontologies mentioned above, such properties are characteristic of the conceptual models, which were first developed to formalize the semantics of ADs in designing information systems. The most widespread conceptual models are the ER model [16, 17], the ORM [18–20], particular cases of the ORM [21–26], and the FCO-IM

[27]. Important extensions of the conceptual models are the conceptual query languages, such as LISA_D [28], Conquer [29, 30], RDQL [31], and CQL [32]. In the framework of SDIS, these languages are used as a unified means for accessing the data published through the global schema.

The models known to the author of this paper, including the conceptual ones, use associations (which are also called relationships, fact types, and so on) as objects in their own right. The associations not only represent relationships between the concepts (also called entities, types of objects, and the like), but also are concepts in their own right that have names and semantics. Associations may be freely created thus defining new relationships between the concepts. Therefore, in the framework of such a model, one cannot be sure that a separate association completely describes the relationships between the involved concepts; indeed, the schema may contain other associations or new associations may be created later that give a different description of the same concepts. As a result, the semantics of each association is defined externally with respect to the schema; that is, the semantics must be known to the user when he or she formulates a query. This reasoning suggests that the schema structure can be simplified by placing the entire semantics of the AD into concepts and depriving the associations of their independent contents so as to make each association to completely describe the relationships between the concepts for which it is created. On the one hand, this goal imposes certain requirements for the schema structure must be satisfied (see below). On the other hand, once this goal is attained, we can do the following. (a) Make it easier for non-IT experts to understand the schema because the associations will give a complete description of the relationships between the concepts. When analyzing each association, the user can be sure that the complete description of the relationships between the concepts is analyzed and that the schema cannot contain another association that gives a different description of the relationships between the same concepts; moreover, no such associations may be created later. (b) Simplify the query language on this schema (see Section 2).

By way of elaborating this idea, the author proposed the semantically complete model (SCM) [33–40]. The main elements of the SCM are the concepts of the application domain, associations of these concepts, and the constraints formulated in the built-in constraint language. The main difference of the SCM from other conceptual models is that each association of a schema is constructed on a *set* of concepts (rather than on a multiset or on a sequence); moreover, in the framework of the schema under consideration, no other association exists that is constructed on the same set of concepts or on its proper subset. As a result, each association completely describes the *semantics* of the relationships between the concepts in the set. It is precisely this prop-

erty, which is called the semantic completeness, that is reflected in the name of the model.

This property has several important consequences. The first consequence is that the work with the model can be performed exclusively in terms of the application domain concepts. This makes the work somewhat simpler and makes it easier for non-IT experts to understand the schema because it is difficult for such people to realize the existence of alternative relationships between the entities. Another consequence is the possibility to develop a conceptual query language that allows one to formulate queries using only the concepts of the application domain rather than the names of associations, which is impossible in the framework of other available conceptual query languages. Such a language was proposed in [34, 37, 38]; it is called the semantically complete query language (SCQL) because it is based on the SCM and essentially uses the semantic completeness property. This language has some properties that simplify its use without detriment to its expressive power, which will be demonstrated in Section 3. The queries in this language have a simple and transparent structure understandable for non-IT experts. It is well known that the simplicity of the query structure is a key factor in query languages used in the SDIS.

The application of the SCM and SCQL as a basis of the SDIS makes it possible to extend their advantages to the integration system and ultimately facilitate the semantic integration process and the system operation. Such a system is called the concept space [35, 38] because, due to the semantic completeness property, the work with this system is performed exclusively in terms of the application domain concepts without using the names of associations. In the proposed system, the first method for schema consolidation is used; i.e., the local schemata are a part of the unified global schema, which reduces the cost of the design and integration of the local schemata. This system assumes that the structures of the implementation of the local information systems are flexibly mapped to the global schema (here and in what follows, intermediate function of the local schemata can be neglected because they are a part of the global schema) without imposing any constraints on the implementation of the systems being integrated.

The system of semantic data integration proposed in this paper is designed to perform the following tasks:

- provide the possibility for the customers or end users to create full-functional user forms for the work with distributed and heterogeneous data without engaging IT experts;
- ensure an automatic integration of new information systems at the moment when their local schemata are published;
- provide mechanism for the semantic navigation and filtration in a distributed and heterogeneous environment without the need for programming;
- ensure that the data access interface is preserved when the structure of the data storage that does not con-

cern the semantics (for example, when the query execution is optimized) is modified.

In the following sections, we briefly describe the semantically complete model, the query language and their properties. The architecture and the properties of the proposed SDIS are discussed in Section 3. In Section 4, we explain how the proposed system performs the tasks listed above. In conclusion, an operational prototype of the system is described, open problems are discussed, and the results are summarized.

2. SEMANTICALLY COMPLETE MODEL AND ITS PROPERTIES

It was mentioned above that the semantically complete model (SCM) is based on three types of elements: the concepts of the application domain, the associations between concepts, and the constraints formulated in the built-in constraint language. The SCM has some important differences from the existing conceptual modeling techniques. The first difference is that each association is constructed on sets of concepts rather than on multi-sets or sequences. As a result, a concept cannot appear in the same association several times, which simplifies the understanding of the SCM schema by non-IT experts and determines all the properties characteristic of the SCQL (see the next Section).

Another difference is an extension of the first one: in the framework of the SCM schema, no two associations may exist that are constructed on the sets of concepts s_1 and s_2 such that $s_1 \subseteq s_2$. This property has the most profound consequences both for the model itself and for the SCQL. It is called the semantic completeness property because any association of the schema completely describes the semantics of a relationship between the relevant concepts. Thus, the SCM does not include mechanisms for describing alternative associations, which violate the semantic completeness.

The semantic completeness is a strict constraint; however, it does not restrict the expressive power of the model because any schema containing alternative associations can be reduced to an equivalent schema without alternative associations by replacing them with some additional concepts and nonalternative associations [37]. We demonstrate this fact using the following example. Consider two relationships between a man and a town: a birth place and a residence. These relationships can be modeled in any conceptual model except for the SCM using the two following associations: a *person* was born in a *town* and a *person* lives in a *town* (the concepts, or type of objects, underlying the associations are typed in italics). Since the SCM does not allow to have both associations in the same schema, one must reduce it to a semantically complete form. For this purpose, we introduce two additional concepts: birth town and residence town. Now, replace the associations described above by the following associations: a *person* was born in a *birth town*, a *person* lives in a

residence town, the *birth town* is a *town*, and the *residence town* is a *town*. Here, the relationships between the concepts *town*, *birth town*, and *residence town* are one-to-one constraints (they are also called equivalence or inheritance relationships). Thus, we transformed the initial schema to a semantically complete schema by introducing new concepts that are equivalently related to the existing concepts.

In contrast to the other conceptual modeling techniques, the SCM does not include mechanisms that take into account the implementation details of the target system. This makes it possible to abstract oneself from implementation details when working with the SCM schema by means of the SCQL, which is especially important for SDIS. For example, consider the most advanced modern conceptual modeling technique called the object–role technique [18–20]. Even this technique assumes that the object types (an analog of concepts in the SCM) are divided into abstract, which are not stored explicitly, and concrete, which are stored explicitly. If this categorization is used in queries on the schema in the framework of the SDIS, then any modification of the schema implementation in this respect requires that the queries be also modified even if the modification of the implementation did not modify the semantics of the schema.

The conceptual modeling techniques not belonging to the ORM class make the developers design the conceptual schemata that incorporate even more implementation details. For example, the ER model [16, 17] subdivides the concepts of the application domain into entities and attributes; this subdivision is more characteristic of data storage (in the form of tables) than of the conceptual representation of the application domain. Since the decisions concerning the data storage are made on the basis of the implementation efficiency, the attributes may be transformed into entities and conversely in the process of the system evolution. Therefore, if a data integration system is based on the ER model, these transformations inevitably lead to the necessity of modification of the queries developed earlier. This fact is one reason for which the ER model evolved in the direction of greater conceptualization; as a result, the ORM with its variants and later the SCM were developed.

The SCM uses mainly the text notation. This notation consists of association sentences and constraint sentences. Each association and each notation of the schema is represented by a single sentence in the text notation. An association sentence is a sentence in a natural language that relates the concepts underlying this association. By way of example, consider the project management application domain. In the framework of this AD, there are associations between the concepts *Person* and *Task*. In the SCM text notation, this association can be written as *Person performs Tasks*

In the association sentence, the concepts of the application domain are capitalized. Several forms of the

same word or word combination may be used to refer to the same concept. In the following example, the concept *Task* appears in different association sentences in singular or in plural: *Person performs Tasks* *Task* is a part of a *Project* → *Employee* is a *Person*≡

The most fundamental constraints are specified in the association sentences. These constraints are as follows: the binary functional dependence, the binary equivalence, and the mandatory constraint. All three types of constraints are illustrated in the example above. An arrow at the end of an association sentence indicates the binary functional dependence; in the example above, this means that a task can be part of only one project. An equivalence sign at the end of an association sentence indicates the binary equivalence; i.e., an employee is a single person, and conversely. If a concept is underscored, then this concept is mandatory in the corresponding association. The mandatory constraint is used twice in the example above meaning that each task is a part of at least one project and each employee is a person. The composition of all the constraints in the example above is reduced to the two following statements: each task is a part of exactly one project, each employee is exactly one person and each person can be only one employee.

More complicated constraints are formulated using constraint sentences, which follow the corresponding association sentence. Every constraint sentence begins with an indentation and is enclosed in square brackets as is shown in the following example. *Person has a Skill at a certain skill Level* [(*Person*, *Skill*) → *skill Level*] This constraint indicates that each pair of the concept instances *Person* and *Skill* is assigned a single skill level (more precisely, a single instance of the concept *skill Level*). This constraint is an example of a static instance of a template specified in the association context (*Person*, *Skill*, *skill Level*). To explain this thesis, we introduce the concepts of a static constraint, a template instance, and a constraint context.

A constraint is said to be static if it is formulated exclusively for a current state of the schema. By the state of a schema, we mean the set of states of all its associations, while the state of an association is the set of instances contained in it. An instance of an association is the set of instances of its concepts. One can cite a distant analogy between an instance of an association and a row in a table: both are the set values (instances) of concepts in the first case and of table fields in the second case.

In the framework of an integration system, only one state of the schema is the current state at each moment in time. As time goes on, the state of the schema changes. Each change step is performed as a result of completing a transaction. At the moment of completion, we may assume that the integration system can access two states of the schema: the preceding and the next one. A constraint formulated with regard for both the preceding and the next states of the schema is called

dynamic. The dynamic constraints in the SCMs make it possible to model the dynamic behavior of the schema contents at the conceptual level without taking into account the implementation details.

The implementation of static constraints usually requires decisions concerning the storage structure and the data access paths (methods); the implementation of dynamic constraints requires algorithms that dramatically depend on these decisions. For example, the static constraint *each task is a part of only one project* implies the following decisions concerning the implementation in the framework of a relational DBMS: (a) there is a table with the primary key [41] corresponding to the concept *Task*; (b) this table must have an indexed field not included in the primary key and corresponding to the concept *Project*.

An example of a dynamic constraint is provided by the description of any finite state machine. Assume that we want each of the tasks registered in the system to have a code consisting of a fixed number of characters; moreover, we want the code of each task to be as different from the codes of the other tasks as possible. First of all, we introduce the concept *Code* into the system: A Task has a Code

Task @@ Code≡

Assume that there is a criterion of code similarity. Then, the dynamic constraint can be formulated as follows: newly added tasks must have codes that are the least similar to the codes of the existing tasks. Since several tasks can be created within the same transaction, this constraint can be more accurately formulated as follows: the total value of the pairwise code similarity criterion for the existing and newly created tasks must be minimum.

Structurally, the formulation of dynamic constraints is the same as the formulation of static constraints. The syntax is as follows: if a SCQL query that is a part of a dynamic constraint involves the next state of the schema, then it is preceded by the symbol “^” (see a discussion of SCQL queries for constraint formulation below).

The static constraints are more fundamental from the viewpoint of the description of the essence of an application domain. They play a key role in the implementation of the final schema. However, there is a wide class of application domains in which the dynamic perspective plays as important role as the static one. For example, this class includes all the application domains that are modeled using finite state machines. The dynamic constraints are most useful in such domains.

It must be noted that the SCM is the first conceptual model used in designing information systems that makes it possible to formulate dynamic constraints. Neither ORM, nor ER, nor other conceptual modeling techniques provide such mechanisms.

In the SCM, a constraint can be specified in two different ways: as an instance of a template or as a propo-

sition formulated using the SCQL predicate calculus, which is the predicate calculus extended by the SCQL. A constraint template is a proposition having a fixed structure with a certain set of parameters. One of these parameters is always an SCQL query that represents the context of the constraint; the result of this query must satisfy the template proposition. Examples of templates are the constraint templates of the functional dependence, equivalence, and mandatory constraint.

For example, the parameters of the functional dependence are as follows: (a) the set of determining concepts, (b) the set of defined concepts, (c) the SCQL query that is to be constrained.

The proposition of each template is formed as follows: in the framework of the result of the given SCQL query, every combination of the instances of the determining concepts is assigned a single combination of the instances of the concepts being defined. A particular instance of the given template is specified by a particular SCQL query, and particular sets of the determining concepts and the concepts being defined.

A constraint sentence formulated using a template has the following syntax in the SCM: [*<SCQL query>*: *<template instance>*]. For example, the constraint *a person can have only one skill level for each skill* is formulated using the functional dependence template as follows:

[(Person, Skill, Skill level):

(Person, Skill) → Skill level]

Here, (Person, Skill, Skill level) is the SCQL query that retrieves the contents of the corresponding association (see the description of SCQL below). Retrieval of an association contents is the simplest variant of a query. If a constraint sentence immediately follows the corresponding association sentence, then the SCQL query may be omitted; for example,

@A Person has a Skill at a certain Skill level

[(Person, Skill) → Skill level]

The definition of all possible constraint templates in the SCM is beyond the scope of this paper. A formalization of some constraint templates in the SCM can be found in [33].

Consider another method of constraint formalization, which is based on the SCQL predicate calculus. In the framework of this extension of the predicate calculus, the concepts of the application domain are considered as sets (the sets of admissible instances of concepts), and the results of SCQL queries are considered as relations defined on the concepts of the application domain. This predicate calculus admits the following abridgements:

(a) A variable defined on a certain concept has the same name as the concept itself (maybe extended in parentheses by the specification of its role in the proposition). (b) The use of the variable *A(role)* corresponding to the concept *A* without explicit declaration is

equivalent to declaring this variable to be an instance of this concept: $A(\text{role}) \in A$. (c) The use of an SCQL query as a predicate is equivalent to the claim that this query returns at least one instance. (d) The use of a concept variable without the existential quantifier is equivalent to declaring this variable with the existential quantifier \exists placed after the other nearest quantifier (up the tree if the proposition is represented by a tree).

For example, the constraint equivalent to the last example above is formulated in the form of the SCQL predicate calculus as follows (a complete notation):

$$[\forall \text{person} \in \text{Person} \forall \text{skill} \in \text{Skill} \exists_{0..1} \text{skill level} \in \text{Skill level} \{(\text{person}, \text{skill}, \text{skill level}) \in (\text{Person}, \text{Skill}, \text{Skill level})\}]$$

In the abridged notation described above, the same proposition is as follows:

$$[\forall \text{Person}(1) \forall \text{Skill} \exists_{0..1} \text{Skill level} \{(\text{Person}(1), \text{Skill}, \text{Skill level})\}]$$

The extension of a name by a role is illustrated above by the digit one in the parentheses after the concept Person. In this context, this extension is fairly useless and can be omitted. Such an extension is useful when the same concept is used in a proposition several times. In that case, the concepts that have different roles are considered to be different variables. In other respects, the SCQL predicate calculus has the principles of proposition construction as the ordinary predicate calculus. The fundamental principles of the SCQL predicate calculus were proposed in [33, 36].

Thus, the SCM is a conceptual model that enables one to model application domains with complex constraints including dynamic ones and the constraints formulated using the extended predicate calculus. The SCQL query language, which will be described below, is a part of the SCM.

The constraints in the SCM, as in other modeling techniques, represent finer specific features of the application domain than the concepts and associations between them. The more powerful the constraint language of a model is, the greater number of features can be represented in the schemata of the model. The presence of the SCQL predicate calculus in the SCM makes it possible to construct a model of an arbitrary application domain with any desired degree of accuracy without specifying the implementation details. In the framework of the SDIS, the constraints in the SCM can be used for the following purposes: (a) to help users get an in-depth understanding of the structure of the application domain; (b) to control the implementation of the information systems development using the requirements represented in the form of an SCM schema; (c) to check whether a particular implementation of an information system meets the requirements represented in the form of an SCM schema.

Note that the text notation of the SCM is fairly close to the natural language, which makes it possible for non-IT experts to use it. There is also a graphical notation in the SCM, which is immaterial for this paper and is not described here for this reason. By way of example, which will be used throughout this paper, we consider the following schema of an application domain concerning project management. The text notation is as follows:

Person performs Tasks

Person has a Skill at a certain skill Level

$[(\text{Person}, \text{Skill}) \rightarrow \text{skill Level}]$

Person has an Age \rightarrow Person can have a Phone

\rightarrow **Employee is a Person**

\equiv Task is a part of a Project

\rightarrow Person can belong to a Project Team

In the next section, this example will be used to illustrate SCQL queries. The bold type is used in the association sentences to describe the execution context of SCQL queries, which is thoroughly described in the next section.

3. SEMANTICALLY COMPLETE QUERY LANGUAGE AND ITS PROPERTIES

The semantically complete query language (SCQL) is used for two purposes: (a) to formulate SCM constraints (see the preceding section); (b) to interact with the semantic data integration system (SDIS) based on the SCM.

The first task assumes only data retrieval; the second task assumes data retrieval, data modification, and modification of the schema itself. In this paper, we focus on data retrieval because it plays the key role in the *semantic* integration. The modification of the data and of the schema using the SCQL is the subject of a separate study.

The SCQL queries are constructed on the basis of the concepts of the application domain. They have a simple structure and a meaningful signature of the result (see below); as applied to the SDIS, this facilitates the process of the system creation and use. As in any other query language, the expression of any SCQL query (below it is referred to simply as expression) is a tree of operations. The operations in the leaves of this tree are called leaf operations; the other operations are called intermediate. Any operation is an instance of an operation type. The SCQL has the following types of leaf operations: association selection and logical selection. It also has the following types of intermediate operations: composition, union, difference, and transformation. Below, we consider the semantics and the syntax of all these operations in detail.

The association selection is the selection of all the instances of an association that are present in the cur-

rent and in the next state of the schema (see the definition of these states above). Due to the semantic completeness property, the SCM and SCQL do not use the proper names of associations, and the associations are referenced using the sets of concepts on which the corresponding associations are constructed. Therefore, the set of concepts is the only parameter of the association selection. It is assumed that the association selection refers to the association whose set of concepts is specified as the selection parameter.

The association selection is specified by listing some concepts of the application domain with the coma as a separator; for example, (Person, Skill, Skill level). The order of the concepts in the list is immaterial; the selection above can be equivalently reformulated as (Skill, Skill level, Person).

The concepts specified in a selection parameter can be extended by specifying the role that they play in the expression; the role is given in parentheses after the concept name. Such extended concepts are called role concepts. The concept with the empty role name is a degenerate case of the role concept. This concept is assumed to be different from all other role concepts of the same concept for which a role is specified. For example, (Person (Project(of the Person))) is the association selection whose result includes the role concept Person with the empty role name and the role concept Project(of the Person). It makes sense to use role concepts if the same concept is used in an expression in several different roles. In this case, different role concepts are considered as semantically different but referring to the same concept; for example, Project(of a Person) and Project(of a Task) are different role concepts of the same concept *Project*.

An association selection can be also represented as a set of role concepts under the condition that no concept can appear in this set several times in different roles. Because the sets of role concepts are used instead of the proper names of associations, SCQL queries can be formulated using only the concepts of the application domain. An association selection is the only method in the SCQL for accessing the current state of the schema (in the framework of dynamic constraints, this is also true for the next state). All other types of operations are used to transform the data obtained using association selections.

The next SCQL operation is the composition. There are two types of compositions—the inner and the outer ones. The inner composition is a particular case of the outer composition, and it adheres to simpler principles. For this reason, we first describe the inner composition. The inner composition is the mathematical superposition of the results returned by several subqueries (in relational algebra, an analog of the inner composition is the natural join [42]). The superposition is performed when the role concepts of the subqueries are identical. For example, consider the subqueries in the form of association selections (Person, Task) and (Task,

Project). Then, their inner composition has the signature consisting of the concepts Person, Task, and Project. Here and in what follows, the signature of an operation including the composition is interpreted as the set of role concepts that describe the structure of this operation result.

The execution of the inner composition can be also described in terms of three relational algebra operations—Cartesian product, selection, and projection—which are not explicitly available in the SCQL. From this point of view, the execution of the inner composition proceeds in three steps: (a) finding the Cartesian product of the subqueries results; (b) selection of the instances in the Cartesian product that contain identical instances of the same role concepts; (c) projecting the result in such a way that each role concept is represented in the projection only once.

The general notation of the composition is the list of the subqueries being composed separated by comas and enclosed in parentheses regardless of the order. For example, ((Person, Task), (Task, Project)) is the inner composition of two association selections. This notation does not impose any restrictions on the arity of the subqueries' signature. There are also two other special forms of notation for the composition in the SCQL: path expression and star expression. Both notations are used only for compositions of selections of binary associations.

A path expression is a chain of role concepts separated by dashes. Each dash denotes one selection of the association constructed on the role concepts specified on two sides of the dash. Thus, a chain as a whole is the inner composition of all the association selections assumed for each of the dashes. For example, (Person-Task-Project) is an equivalent notation of the composition described above.

A star expression is a star-like structure consisting of the selections of binary associations with a single central role concept. The format of the star expressions is as follows: (the central role concept-[peripheral role concepts separated by comas]). For example, (Person-[Task, Project]) is a star expression equivalent to the composition described above. It makes sense to use a star expression when the number of peripheral concepts is greater than two. The special notations of the composition make the queries clearer. The query developer is free to choose which notation to use based on the query structure and his or her own preferences.

The inner composition is very easy to use due to the following properties: (a) It has a single parameter that specifies the set of the subqueries being composed. (b) It does not require an explicit formulation of the criterion for the subqueries join. (c) The resulting signature is formed in an intuitively clear way as the union of the signatures of the subqueries being composed. (d) The meaning of each element of the signature is preserved because the role concepts of the elements directly correspond to the concepts of the application domain rep-

resented in the SCM schema. (e) In some cases, intuitively clear path and star expressions can be used.

These properties of the inner composition significantly simplify its understanding, make it easier to master the language, and create queries.

The other type of composition is the outer composition. It is a distant analog of the outer join in SQL. Like the inner composition, the outer composition is based on a set of subqueries. The difference is that the subqueries are additionally subdivided into two categories: the inner and the outer ones. The outer composition must meet the following constraints: (a) at least one subquery is inner; (b) the signatures of all the inner subqueries form a connected hypergraph; (c) each outer subquery contains in its signature at least one role concept common with each inner subquery.

The execution of the outer composition proceeds in three steps: (a) inner composition of all the inner subqueries; (b) selection of all adjacent instances (see below) of the outer subqueries for each instance i of the inner composition; (c) partial inner compositions for each instance i within its adjacent instances.

Here and in what follows, two instances of distinct subqueries are said to be adjacent if they contain identical instances for each common role concept. The result of performing an outer composition as a whole is the set of instances of all partial inner compositions. It is clear that, if an outer composition includes only inner subqueries, then it is reduced to an inner composition.

Outer compositions can be represented in the same ways as inner ones. The difference is that the outer subqueries are marked by the symbol + after the subquery. For example, ((Person, Task), (Task, Project)+) is the outer composition of the inner subquery (Person, Task) with the inner composition of the outer subquery (Task, Project)+. The same composition notation of a path and star expressions is written, respectively, as follows: (Person-Task-Project+) and ((Task-[Person]), (Task-[Project]+)).

The logical selection operation is a leaf operation that has the predicate parameter based on role concepts. The result of a logical selection is the set of all combinations of the role concepts that do not contradict the predicate. A logical selection is written as a logical predicate enclosed in parentheses. For example, the result of the logical selection (Age < 30 AND Phone > 20000) consists of all possible combinations of the instances of the concepts Age and Phone in which the age is less than 30 and the phone number is greater than 20000.

Every logical selection must be an inner query of a composition and must be based on the role concepts that are included in the signatures of the other subqueries in the same composition that are not logical selections. In this case, the composition acquires an additional filtering effect because all the instances that contradict the predicates of the nested logical selections are filtered out. For example, the query “select the tasks

performed by people younger than 30 years” is formulated as follows: ((Task-Person-Age), (Age < 30)) in the full notation. For the predicates based on the same role concept, an abridged notation can be used. In this notation, the last query can be reformulated in the form (Task-Person-(Age < 30)).

Thus, SCQL does not have an analog of the SELECT operator of the relational algebra because the same filtration effect is achieved by including a logical selection in the composition. It can be shown that, if a composition includes several logical selections, then it can be transformed to a composition with a single logical selection based on the predicate that is the conjunction of the predicates of the initial logical selections.

The union operation in SCQL is used to combine the results of several subqueries. As the inner composition, the union has no additional parameters except for the subqueries themselves. The union operation is executed in two steps: (a) selection of clusters of instances; (b) composition within these clusters.

The selection of clusters of instances is performed according to the following rules: (a) If two instances of different subqueries are adjacent (see above), then they belong to the same cluster. (b) If two instances of different subqueries can be assigned to different clusters without violating rule (a), then they belong to different clusters.

The composition within a cluster is the inner composition performed taking into account only the instances of the subqueries that belong to this cluster. The result of the union operation as a whole is the set of all the instances obtained at step (b).

The signature of a union is formed as the union of the signatures of the subqueries. This is an advantage of SCQL over the similar operation in other query languages that often require that the subqueries be reduced to a common signature before the union. It follows from the union description that, if the signatures of all the subqueries are identical, then the result is the set of all distinct instances of the subqueries. In this case, the union operation is close to the union used in the relational algebra. If the signatures of all the subqueries are different, then the closest analog in SQL is the complete outer join.

The union is written by listing the queries being combined in parentheses separated by the key word *union*. By way of example, consider the following query: for each person, select the projects in which he performs tasks or to which he is assigned, and specify all those tasks. This union is written as ((Person-Task-Project) union (Project, Person)). This example illustrates the simplicity of the union operation in SCQL. Indeed, the subqueries need not be reduced to a common signature, nor the role concepts in the subqueries are required to have the same order. Moreover, the resulting signature is semantically clear because it is based on the concepts of the application domain. For this reason, I believe that the union operation in SCQL

is simpler to understand and use than similar operations in other query languages.

The next operation in SCQL is the difference. It is always constructed of two subqueries and is used to filter the results of the first subquery by the inverted result of the second subquery. The result is the set of instances of the first subquery that have no adjacent instances in the result of the second subquery. The difference is the only operation in SCQL for which the order of the subqueries is important.

It follows from the definition of the difference that its signature coincides with the signature of the first subquery. The advantage of the SCQL difference over its analogs in other query languages is that it does not require that the signatures of the subqueries be identical and the role concepts in them have the same order; moreover, the signature of the difference is based on the concepts of the application domain. The notation of the difference operation is as follows: (<the first subquery> minus <the second subquery>). For example, the query *get the phone numbers of all the people that have no high skill level in any skill* is formulated as follows: ((Person, Phone) minus (Person, Skill, (Skill level = "High"))).

The last intermediate operation in SCQL is the transformation. It is built from a single subquery and is used to transform its signature. The resulting signature of the transformation operation is explicitly specified as its parameter; it consists of the role concepts of three types: calculated, aggregated, and copied concepts. The copied role concepts are the role concepts of the query that are directly moved to the result without any changes. The calculated role concepts are not directly present in the subquery signature but are calculated using a non-aggregating function. The aggregated role concepts are not contained in the subquery signature but are calculated using an aggregating function. The resulting signature of the transformation operation cannot consist only of aggregated role concepts: it must contain at least one copied or calculated role concept.

The transformation operation is performed in four steps: (a) Each instance of the subquery is extended by the values of the calculated role concepts using the corresponding functions. (b) The extended results are projected by the copied and calculated role concepts without duplication of instances. (c) Each instance of projection (b) is assigned a group of the initial instances of the subquery that were projected to the given instance of the projection at the preceding step. (d) Each instance of projection (b) is extended by the values of the aggregated role concepts calculated using the corresponding aggregating functions over group (b) that corresponds to the given instance of the projection.

The set of all extended instances of the projection obtained at step (d) is the result of the transformation. Thus, the SCQL transformation operation performs all the transformations of the subquery that are related to the controlled change of its signature. In other query

languages, analogs of this operation are the projection, grouping, and calculation of aggregating and non-aggregating functions.

There are two equivalent notations for the transformation operation: `SELECT <role concepts> FROM <subquery>` or `<subquery>.<role concepts>`. The user is free to choose any of them. For example, the query *get the average age of people in the team of each project* is formulated as `SELECT Person, AVG(Age) FROM` or as `(Project-Person-Age).(Person, AVG(Age))`. Here, `AVG` is the aggregating function of taking the average. The complete list of the SCQL functions and the description of their syntax are beyond the scope of this paper.

The transformation operation has some advantages over the similar operations in other query languages. The first advantage is that it is parameterized only by the resulting signature and by the subquery; the other languages require that the additional parameter specifying the grouping method be explicitly provided. The second advantage is that the signature of the transformation remains semantic; i.e., it is based on the concepts of the application domain. This simplifies the understanding and the use of the transformation operation.

We have considered all operations of the SCQL. Using these operations, practically any query on the SCM schema can be formulated. Another essential part of the SCQL is the context mechanism, which makes it possible to deal with indirectly related concepts as if they were related directly. A context determines the possible ways of shortening SCQL queries. The context that is in effect at the current moment is called the current context.

Any context is a set of associations of the SCM schema. Any set of associations, including a context, can be considered as a hypergraph constructed from concepts, which are used as the hypergraph nodes, and associations, which serve as its edges. Consider the connected subgraphs of the hypergraph that corresponds to a certain context. The interrelation of the concepts in such a subgraph can be obtained in the same way as if these concepts were directly related by an association, i.e., by listing the desired concepts. For example, if a context includes the associations (Employee, Person) and (Person, Phone), then the relation between the concepts Employee and Phone can be obtained using the query (Employee, Phone) without the need to explicitly formulate the composition (Employee-Person-Phone).

It is seen from this example that the indirect relation query is a special case of the association selection operation (see above). Taking into account the context mechanism, the association selection is executed as follows:

- If the specified set of concepts corresponds to an SCM schema association, then the query returns all the instances of this association as its result.

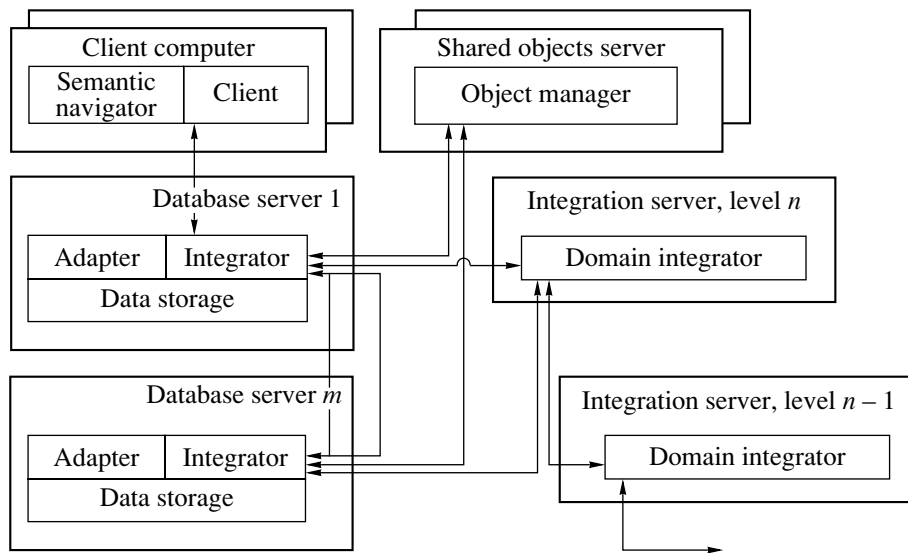


Fig. 1. Architecture of the semantic data integration system proposed.

- If the specified set of concepts belongs to a connected subgraph of the hypergraph corresponding to the current context, then – all the alternative paths that connect the specified concepts one with another are found; – the associations belonging any of the alternative paths are combined into a unified set of associations; – the inner composition of the resulting set of associations is constructed and projected by the desired concepts; – all the instances of this projection are returned as the query result.

- Otherwise, an error is reported: no relationship between the specified concepts can be established either directly or indirectly.

The associations that are always included in the current context are marked by bold type in the SCM text notation (see the example in Section 2). Other associations can be included into the current context by a user or automatically according to various rules. Recall the example considered in Section 2. If the current context, in addition to the associations indicated in that example, also includes the association (Person, Project), then the average age of the participants of each project can be obtained using the query (SELECT Project, AVG(Age) FROM (Project-Age)) or using the equivalent query (Project, Age).(Project-AVG(Age)). Both queries can be reduced to the equivalent query (Project-AVG(Age)).

The use of the context mechanism makes it possible to make queries much simpler and to modify their semantics without modifying their structure. For example, if we replace the association (Person, Project) in the context with associations (Person, Task) and (Task, Project), then the same query (Project-AVG(Age)) acquires a completely different meaning: for each project, get the average age of the people who perform its tasks.

If it is required to preserve the semantics of a query independently of the context, then the query must be formulated using the association selections that refer to the associations that are included in the schema. For example, the preceding query must be reformulated as (Project-Task-Person-Age).(Project, AVG(Age)) in this case. The current context is not involved in the execution of this query because all the relevant relationships between the concepts are direct. Thus, the SCQL combined with the context mechanism makes it possible to considerably simplify the structure of the conceptual queries, which justifies the choice of the SCM-SCQL as the basis for the semantic data integration system proposed in this paper.

4. ARCHITECTURE OF THE SEMANTIC DATA INTEGRATION SYSTEM

The semantic data integration system proposed in this paper (it is referred to as the system in the following consideration) uses the SCM-SCQL as the data access interface, which makes it possible to endow this system with some unique functional capabilities. The main software components of this system are the data storage, adapter, integrator, domain integrator, object manager, client, and the semantic navigator (see Fig. 1).

The information systems to be integrated reside on so-called data servers and provide their data to the integration system for shared access. The shared data storages may be managed by various DBMSs including the most popular relational DBMSs Oracle, MS SQL, MySQL and others. Along with the data storage, an adaptor and an integrator must reside on the same server. The adaptor maps the contents of the storage to the part of the SCM schema that it publishes. Each adaptor publishes its own part of associations and constraints of the global schema while using a unified (for

all the adapters) set of concepts. The mapping of the storage to the global schema is sufficiently flexible to ensure that two different but semantically equivalent storage structures can be mapped to the same SCM schema. This enables us to preserve the data access interface when the storage structure is modified without changing the semantics, for example, when the query execution or the information storage details are optimized. The principles of such a mapping are considered elsewhere (see, e.g., [38, 40]).

The integrator implements the protocols used to exchange data between the server and the external software components. The integrator interacts with the object managers, which can reside both on the data server and on the dedicated servers, to obtain data concerning the structure of users' forms and other shared data. Each integrator is connected to one of the domain integrators, which form a hierarchical structure. The domain integrators supply each integrator with information concerning the structure of the global schema published by other integrators.

The user's computer runs two components: the semantic navigator and the client. The client interacts with the navigator to obtain information concerning the structure of the global schema, the schema modification, execution of SCQL queries (including data modification), to obtain information concerning the structure of the user forms, published components, and so on. The semantic integrator uses the client to supply the structure and the contents of the global schema to the user.

Consider the principles of the system operation using a typical scenario as an example. We assume that all the relevant servers are fine-tuned, and the parts of the global schema that map the structure and the contents of each data storage are published via the adapters. The main steps of the system operation in the usage (not tuning) mode are as follows:

- The user examines the SCM schema and formulates an SCQL query (which may modify data and the SCM schema).
- The semantic navigator sends the query to the client, and the client sends it to the integrator.
- The integrator requests from its domain integrator the information concerning the distribution of associations over the data servers to find out which other integrators must take part in the query execution (if this query is a distributed one).
 - If the domain integrator has no relevant information, it requests it from the higher level domain integrator (this operation is repeated recursively).
- The integrator separates the query into parts and sends it to the remote integrators, while the locally executed part is sent to the local adaptor, for execution. If the query is a data selection query, then the integrator collects the results of the distributed query and returns the combined result to the client; if the query is a data

modification one, then the integrator uses the transaction manager to manage the distributed transaction.

- The adaptor executes the local SCQL query by mapping it to a query on the DBMS (for example, to a SQL query) that manages the particular data storage.
- The remote adapter executes its part of the query using its adapter and returns the result to the calling integrator or uses the transaction manager to declare the changes a part of the global transaction.
 - The semantic navigator visualizes the result of the query execution.
 - The user initiates the completion or the rollback of the global transaction: the client sends the command to the integrator, and the integrator uses the transaction manager to complete or roll back the global transaction.

The information concerning the change of the global schema (the set of concepts, associations, and constraints) is sent to the corresponding domain integrators within the same transaction. This ensures the integrity of the global schema at any time from the viewpoint of any client connected to various data servers of the system; this also solves the problem of automatic integration of any new information system at the moment of the publication of each part of the global schema.

The implementation of such a large-scale system involves details that cannot be considered in a single paper. In this paper, we focus on the issues concerning the semantic navigator because its functionality is crucial for our purposes. The semantic navigator can function in two modes: work with the global schema structure and inspection of user forms. In the first mode, the user can browse through the published elements (concepts, associations, constraints, and the structure of the user forms), navigate through the related elements, change, delete, and add elements, modify SCQL queries and save them for further use. In the first mode, one can use the published forms to solve application problems and navigate through those forms. Prior to presenting the details of the system operation, we describe the structure of user forms.

Each form has a unique name, which can be used to request its structure from the object manager. The form's structure is a set of interrelated interface elements, which are visualized using special software components; the software components are also published by object managers. The visualization components satisfy the following well-known principles of designing component-oriented architectures:

- (a) Each component has its own structure of properties; the user of a component instance may modify these properties to adjust the component to his needs.
- (b) The components form a hierarchy of property inheritance: the descendant component inherits all the properties of the parent component.

In addition, all the visual components of the system under consideration have the following general properties:

(a) an SCQL query that is visualized by the components; (b) a filter (a predicate) constructed from role concepts that is used to filter the query result before its visualization; (c) a set of input and output links between the interface elements; (d) a location of the component on the form; (e) a position of the cursor and selections of rows and columns in the query result.

An interface element is a named instance of a component declared in the framework of a user form. The element name is used to enable the elements reference each other within the form. For each interface element, the initial values of its properties are defined, including its position on the form that the element takes when the form is loaded. Any of these parameters, including the SCQL query, the filter, and the links between the elements, may be changed by the user while working with the form. Thus, the form can be adjusted to the user's needs at run time; the modified variant of the form can be saved for local use or published under a new name for the shared use.

A link between the interface elements is used to transfer data between them; this link is a non-visual component. The general properties of the links are as follows:

(a) an interface element (data source); (b) an interface element (data handler); (c) the signature of the data being transferred (a set of role concepts).

The data are transferred from the source element to the handler element in accordance with the given signature. The data to be transferred between the elements and the method for their processing are defined by the link type and by its additional properties.

Consider the most important master–detail link. The data transferred over this link serve for the addition filtration of the contents of the handler element. Since both the signature of the SCQL query being filtered and the signature of the link are constructed from the concepts of the application domain, there is no need for the formulation of additional predicates: the filtration is performed on the basis of the equality of the values of all the coinciding role concepts (as in the inner SCQL composition),

The data transferred over the master–detail link are formed by one of the following methods: (a) The values of the role concepts listed in the link signature are extracted from the current row of the table being visualized; this method corresponds to the conventional meaning of the master–detail link. (b) The combinations of values of the role concepts listed in the link signature are extracted from the rows selected by the user. (c) The combinations of values of the role concepts are extracted from the columns selected by the user; the link signature is automatically modified. (d) The combinations of values of the cells of the role concepts are extracted from the cells selected by the user; the link signature is automatically modified.

Such a structure of the user forms enables the end users to create new forms (for working with distributed

and heterogeneous data) without the help from IT experts; these forms have the following functionality: (a) browsing and modification of data based on the results of SCQL queries represented by various components (grids, trees, diagrams, etc.); (b) master–detail links and other links between the interface elements; (c) filtration of the data visualized by the interface elements; (d) tuning the location and other properties of the interface elements.

It should be noted that the tools that are presently available make it possible to develop forms with a similar functionality only with the help of professional programmers using high-level programming languages.

The possibility of semantic navigation between forms is an important capability of the proposed navigator. The semantic navigation consists of three phases: (a) the user selects some combinations of values of the role concepts; (b) the system finds the semantically related forms to which the user can navigate in the context of the chosen role concepts; (c) the user indicates one of the forms, which is then loaded and tuned for the selected combinations of the values of the role concepts.

The execution of the first phase—selection of some combinations of values of the role concepts—completely depends on the visualizing component. For example, the following selection modes are available for grids: (a) all values of the current row; (b) combinations of all values of the selected rows; (c) combinations of all values of the selected columns; (d) combinations of the values of the selected cells.

The second phase—finding the semantically related forms—is performed by the navigator using the object manager; it can proceed in several modes: (a) all forms for which the signature of the main interface element (see below) contains at least one selected concept (or role concept); (b) all forms for which the signature of the main interface element contains all the selected concepts (or role concepts); (c) all forms for which the signature of each interface element contains at least one selected concept (or role concept); (a) all forms for which the signature of each interface element contains all the selected concepts (or role concepts).

Here, the main interface element is the element that has no input master–detail links (there may be several main elements in a form).

In the last phase of the semantic navigation the user chooses a form from the list of semantically related forms and instructs the system to load it in the context of the selected combinations of values of the role concepts. The context form loading means that, upon the form initialization, the system modifies the current filter of the interface element that was used as the basis for determining this form as a semantically related. The modification of the filter consists of its conjunction with an additional predicate that represents the equality of the common role concepts of the signature of the interface element and the role concepts selected by the

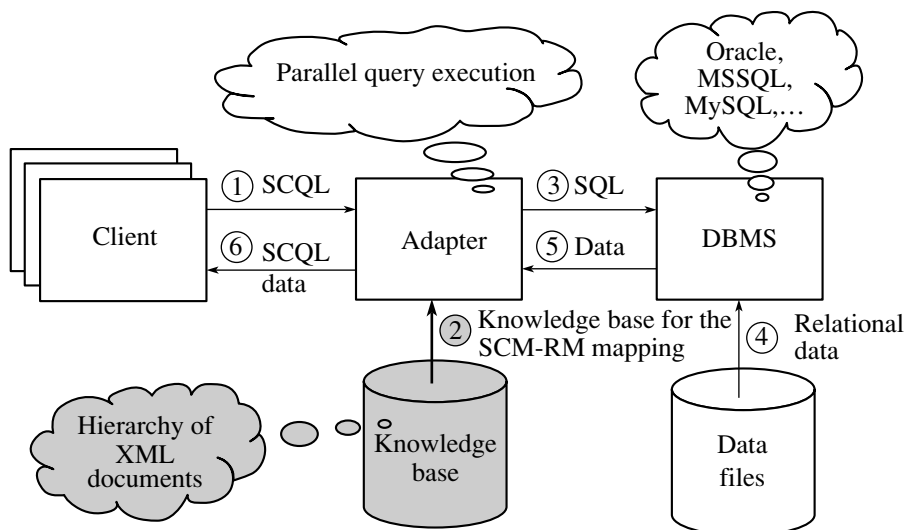


Fig. 2. Architecture of the prototype: client/server technology based on the SCM-SCQL.

user. As a result, this interface element will display only the instances of the corresponding SCQL query that completely contain one of the combinations of the role concepts selected by the user (this is similar to the inner SCQL composition); i.e., the user is presented the information in the context of the selected combinations of the role concept values.

Thus, the proposed architecture of the system provides semantic navigation and filtration mechanisms in a distributed heterogeneous environment without the need for programming. To my knowledge, the proposed system of semantic data integration has no full-functional analogs.

5. CONCLUSIONS

The application of the SCM and SCQL as a basis for a semantic data integration system facilitates the development, integration, and understanding of the global schema; it also speeds up and facilitates the development of an integrated system with rich functional capabilities without the need for programming. A high degree of facilitation is a result of the semantic completeness of the SCM, which also implies the following properties: (a) the possibility to work with the schema exclusively in terms of the application domain; (b) stable semantics of associations, which remains intact when new associations are added; (c) the SCQL query language with some special properties.

The special properties of SCQL that simplify the formulation and examination of queries are as follows: (a) The queries are formulated using the concepts of the application domain and without using the proper names of associations. (b) The queries have a semantic signature constructed in terms of the concepts of the application domain; the signature is constructed according to simple and clear rules. (c) The composition operation

does not require an explicit join criterion and is performed according to intuitively clear rules. (d) The path and star expressions significantly facilitate the notation of the composition operation in the particular case of binary associations. (e) The transformation operation does not require an explicit grouping criterion. (f) With the help of the context mechanism, the relationship between indirectly interrelated concepts can be requested by simply listing the relevant concepts. (g) The semantics of a query can be changed by modifying the current context without modifying the query structure and so on.

The architecture of the semantic data integration system has the following advantages: (a) It enables end users to create fully functional user forms for the work with distributed heterogeneous data without resorting to IT experts. (b) It makes it possible to automatically integrate information systems once their local schemata are published. (c) It supports semantic navigation and filtration mechanisms in a distributed heterogeneous environment without the need for programming. (d) It makes it possible to improve the implementation of an information system without changing the data access interface.

The author has developed a prototype that demonstrates the possibility of a practical implementation of a semantic data integration system based on the SCM-SCQL. The prototype is available at site [40]. This prototype is developed in the framework of the client-server technology based on the SCM-SCQL. Its architecture is illustrated in Fig. 2, and it has an independent practical value.

This technology is designed to develop client-server systems in which the server functions as an adapter that publishes the contents of the data storage in the form of an SCM schema; the server also executes SCQL queries. In contrast to the semantic data integra-

tion system, this technology does not include other software components and, therefore, it does not allow one to integrate several adapters into a unified information space. In the framework of the client-server technology, the client directly interacts with the adapter. A typical scenario of the work with a client-server system developed on the basis of this technology is as follows: – an application forms an SCQL query and sends it to the adapter through the client; – the adapter translates this query into a SQL query using the knowledge base that describes the mapping of the relational schema into the SCM schema and sends the SQL query to the relational DBMS for execution; – the RDBMS executes the SQL query and returns the result to the adapter; – the adapter transforms the result to the form of the initial SCQL query and returns it to the client; – the application receives the data from the client and processes them.

The knowledge base is a hierarchy of XML documents. Each document describes the SCM schema for a certain application domain and its mapping into a relational schema. The document nesting is used to organize a hierarchy of application domains in the part-whole framework. In future, it is supposed to develop adapters for other (not relational) types of DBMSs.

A full practical implementation of the proposed architecture of the semantic data integration system requires additional studies in the following directions: – mapping of the SCM schema to other schemata in addition to the relational ones; – mapping of the SCQL queries to other query languages in addition to SQL; – mapping of the SCQL queries to SQL queries with simultaneous run-time optimization; – support of enumeration type domains for concepts; – extensions of the SCQL for working with meta-information; – principles of data access management in the framework of the semantic integration system; – detailed architecture of each software component, i.e., of the integrator, domain integrator, object manager, and semantic navigator.

REFERENCES

1. Ermolaev, V., Keberle, N., Shapar, V., and Vladimirov, V. Ontology-Driven Sub-Query Extraction for Distributed Autonomous Information Resources in UnIT-IEDI, in *Proc. 3rd Int. Conf. on Information Systems Technology and Its Applications ISTA'2004*, Salt Lake City, *Lect. Notes Inform.*, 2004, vol. P-48, pp. 137–150.
2. Wache, H., Ontology-Based Integration of Information—A Survey of Existing Approaches, in *Proc. of the IJZAI-01 Workshop on Ontologies and Information Sharing*, 2001, pp. 108–118.
3. *RDF Vocabulary Description Language 1.0: RDF Schema*, McBride, B., Ed., W3C Recommendations, 2004; <http://www.w3.org/TR/rdf-schema/>.
4. *OWL Web Ontology Language Reference*, Dean, M. and Schreiber, G., Eds. W3C Recommendations, 2004; <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
5. Bergamaschi, S., Castano, S., Vincini, M., and Benevenuto, D., Semantic Integration of Heterogeneous Information Sources, *Data & Knowledge Eng.*, 2001, vol. 36, no. 3, pp. 215–249.
6. Collins, S.R., Navathe, S., and Mark, L. XML Schema Mappings for Heterogeneous Database Access, *Inf. Software Technol.*, 2002, vol. 44, no. 4, pp. 251–257.
7. Gartner, H., Bergmann, A., and Schmidt, J., Object-Oriented Modeling of Data Sources as a Tool for the Integration of Heterogeneous Geoscientific Information, *Comput. & Geosciences*, 2001, vol. 27, no. 8, pp. 975–985.
8. Jensen, M.R., Moller, T.H., and Pedersen, T.B., Converting XML DTDs to UML Diagrams for Conceptual Data Integration, *Data & Knowledge Eng.*, 2003, vol. 44, no. 3, pp. 322–346.
9. Pontieri, L., Ursino, D., and Zumpano, E., An Approach to Extensional Integration of Data Sources with Heterogeneous Representation Formats, *Data & Knowledge Eng.*, 2003, vol. 45, no. 3, pp. 291–331.
10. Claypool, K.T., and Rundensteiner, E.A., Gangam: A Transformation Modeling Framework, in *8th International Conference on Database Systems for Advanced Applications (DASFAA'03)*, 2003, p. 47.
11. Levy, A.Y., Rajaraman, A., and Ordille, J.J., Querying Heterogeneous Information Sources Using Source Description, in *Proc. of the 22nd International Conference on Very Large Databases (VLDB'96)*, 1996, pp. 251–262.
12. Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J., The TSIMMIS Project: Integration of Heterogeneous Information Sources, in *Proc. of the 10th Meeting of the Information Processing Society of Japan*, 1994, pp. 7–18.
13. Friedman, M., Levy, A., and Millstein, T., Navigational Plans for Data Integration, in *Proc. 10th National Conf. on Artificial Intelligence, (AAAI'99)*, 1999 pp. 67–73.
14. Li, Y., Liu, D. and Zhang, W., A Data Transformation Mapping Based on Schema Mapping, in *Proc. 3rd Int. Conf. on Information Systems Technology and Its Applications ISTA'2004*, Salt Lake City, *Lect. Notes Inform.*, 2004, vol. P-48, pp. 67–79.
15. Xu, L. and Embley, D.W., Combining the Best Global-as-View for Data Integration, in *Proc. 3rd Int. Conf. on Information Systems Technology and Its Applications ISTA'2004*, Salt Lake City, *Lect. Notes Inform.*, 2004, vol. P-48, pp. 123–135.
16. Chen, P.P.S., The Entity-Relationship Model—Towards a Unified View of Data, *ACM Trans. Database Syst.*, 1976, vol. 1, no. 1, pp. 9–36.
17. Chen, P.P.S., A Preliminary Framework for Entity-Relationship Models, in *Entity-Relationship Approach to Information Modeling and Analysis*, Saugus, Calif. 1981.
18. Bronts, G.H.W.M., Brouwer, S.J., Martens, C.L.J., and Proper, H.A., Unifying Object Role Modeling Approach, *Inf. Syst.*, 1995, vol. 20, no. 3, pp. 213–235.
19. Halpin, T.A., *Information Modeling and Relational Databases*, San Francisco: Morgan Kaufman, 2001.
20. Halpin, T.A., *Conceptual Schema and Relational Database Design*, Sydney: Prentice-Hall, 1995, 2nd ed.
21. van Bommel, P., ter Hofstede, A.H.M., and van der Weide, Semantics and Verification of Object-Role Models, *Inf. Syst.*, 1991, vol. 16, no. 5, pp. 471–495.

22. Brouwer, Martens, C.L.J., S.J., Bronts, G.H.W.M., and Proper, H.A., Towards a Unifying Object Role Modeling Approach, in *Proc. First Int. Conf. on Object-Role Modelling (ORM-1)*, Magnetic Island, Australia, 1994, pp. 259–273.
23. Halpin, T.A. and Orłowska, M.E., Fact-Oriented Modeling of Data Analysis, *J. Inf. Syst.*, 1992, vol. 2, no. 2 pp. 1–23.
24. ter Hofstede, A.H.M. and van der Weide, Expressiveness in Conceptual Data Modeling, *Data & Knowledge Eng.*, 1993, vol. 10, no. 1, pp. 65–100.
25. Nijssen, G.M. and Halpin, T.A., *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*, Sydney: Prentice-Hall, 1989.
26. de Troyer, O.M.F., The OO-Binary Relationship Model: A Truly Object Oriented Conceptual Model, *Proc. Third Int. Conf. CaiSE'91 on Advanced Information Systems Eng.*, Trondheim, Norway, 1991, *Lect. Notes Comput. Sci.*, 1991, vol. 498, pp. 561–578.
27. Bakema, G., Zwart, J., and van der Lek, H., Fully Communication Oriented NIAM, in *NIAM-ISDM 1994 Conf. Working Papers*, Albuquerque, 1994, p. L1–35.
28. ter Hofstede, A.H.M., Proper, H.A., and van der Weide, Formal Definition of a Conceptual Language for the Description and Manipulation of Information Models, *Inf. Syst.*, 1993, vol. 18, o. 7, pp. 489–523.
29. Bloesch, A.C. and Halpin, T.A., ConQuer: A Conceptual Query Language, in *Proc. Of ER'96: 15th Int. Conf. On Conceptual Modeling, Lect. Notes Comput. Sci.*, 1996, vol. 1157, pp. 121–133.
30. Bloesch, A.C. and Halpin, T.A., Conceptual Queries Using ConQuer-II, in *Proc. Of ER'97: 16th Int. Conf. On Conceptual Modeling*, 1997, vol. 1157, pp. 121–133.
31. RDQL—A Query Language for RDF, W3C Member Submission, 2004, <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>.
32. Owei, V., Navathe, S.B/, and Rhee, H.-S., An Abbreviated Concept-Based Query Language and Its Exploratory Evaluation, *J. Syst. Software*, 2002, vol. 63, no. 1, pp. 45–67.
33. Ovchinnikov, V.V., Improving the Controllability of Large Conceptual Models, *Inform. Tekhnol.*, 2004, no. 10, pp. 8–14.
34. Ovchinnikov, V.V., Fundamentals of Constructing a Conceptual Query Language on a Semantically Complete Model, in *Trudy Instituta Problem Upravleniya: Upravlenie bol'shimi sistemami (Controlling Large Systems)*, Moscow: Institut Problem Upravleniya, Ross. Akad. Nauk, 2004.
35. Ovchinnikov, V.V., The Technique of Semantically Complete Modeling and Its Application or Designing Distributed Heterogeneous Information Systems, *Inform. Tekhnol.*, 2005, no. 3.
36. Ovchinnikov, V.V., A Conceptual Modeling Technique without Redundant Structural Elements, *J. Conceptual Model.*, 2003, vol. 29, www.inconcept.com/jcm.
37. Ovchinnikov, V.V., A Conceptual Modeling Technique Based on Semantically Complete Model, Its Applications, in *Proc. of the 3rd International Conference on Information Systems Technology and Its Applications ISTA'2004*, Salt Lake City, *Lect. Notes Inform.*, 2004, vol. P-48, pp. 25–38.
38. Ovchinnikov, V.V., Architecture of Heterogeneous Concept Space Managed by non-IT people, *J. Conceptual Model.*, 2004, vol. 33, www.inconcept.com/jcm.
39. Ovchinnikov, V.V., A Concept-Based Query Language Not Using Proper Relation Names, in *Proc. of CAiSE'05 Workshops*, 2005, Vol. 1, pp. 617–628.
40. Ovchinnikov, V.V., SCM Portal, <http://scm.lipetsk.ru>
41. Codd, E.F., Domains, Keys, and Referential Integrity in Relational Databases, *InfoDB*, 1988, vol. 3, no. 1.
42. Codd, E.F., Relational Completeness of Data Base Sublanguages, *Data Base Syst.*, in Courant Computer Science Symposia Series 6, Englewood Cliffs, N.J.: Prentice-Hall, 1972.

SPELL: 1. arity, 2. easer