Ontology based information integration using Logic Programming

Gergely Lukácsy and Péter Szeredi

Budapest University of Technology and Economics
Department of Computer Science and Information Theory
1117 Budapest, Magyar tudósok körútja 2., Hungary
Phone: +36 1 463-2585 Fax: +36 1 463-3157
`{lukacsy,szeredi}@cs.bme.hu`
Keywords: descripton logic, CWA, information integration, logic programming

Abstract. We present an information integration system called SINTAGMA which supports the semantic integration of heterogeneous information sources using a meta data driven approach. The main idea of SINTAGMA is to build a so called Model Warehouse, containing several layers of integrated models connected by mappings. At the bottom of this hierarchy there are the models representing the actual information sources. Higher level models represent virtual databases, which can be queried, as the mappings provide a precise description of how to populate these virtual sources using the concrete ones.
This paper focuses on a recent development in SINTAGMA allowing the information expert to use Description Logic based ontologies in the development of high abstraction level conceptual models. Querying these models is performed using the Closed World Assumption as we argue that traditional Open World DL reasoning is less appropriate in the context of database oriented information integration environments.
The implementation of SINTAGMA uses constraints and logic programming, for example, the complex queries are translated into Prolog goals. This allows us to provide functionalities not supported by other integration frameworks.

1 Introduction

This paper presents the Description Logic modelling capabilities of the SINTAGMA Enterprise Information Integration system. SINTAGMA is based on the SILK tool-set, developed within the EU FP5 project SILK (System Integration via Logic & Knowledge) [2]. SILK is a data centered, monolithic information integration system supporting semi-automatic integration on relational and semi-structured sources.

The SINTAGMA system extends the original framework in several directions. As opposed to the monolithic SILK structure, SINTAGMA is built from loosely coupled distributed components. The functionality has become richer as, among others, the system now deals with Web Services as information sources.

The present paper is about a recent extension of the system which allows the integration expert to use Description Logic models in the integration process.

The paper is structured as follows. In Section 2 we give a general introduction to the SINTAGMA system, describing the main components, the SILan modelling language, and the query execution mechanism. In the next section we discuss the description logic extension of SILan: we introduce the syntactic constructs and the modelling methodology. In Section 4 we present the execution mechanism used when querying Description Logic models. In Section 5 we examine related work. Finally, we conclude with a summary of our results.

The examples we use in the upcoming discussions are part of an integration scenario. This scenario represents a world where we attempt to integrate various information sources about writers, painters and their work (i.e. books, paintings, etc.) and present this information in the form of abstract views.

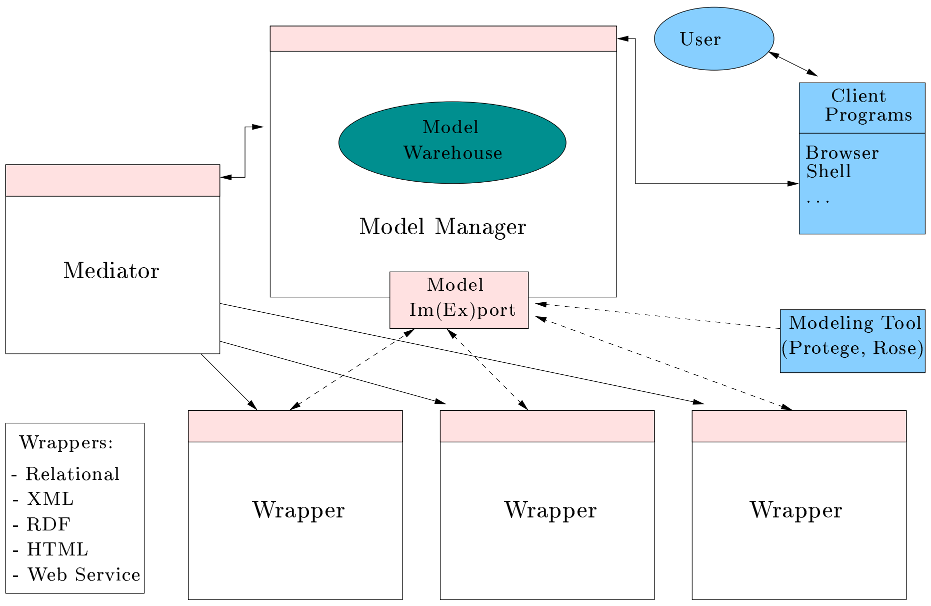

Fig. 1. The architecture of the SINTAGMA system

2 System Architecture

The overall architecture of the system can be seen in Figure 1. The main idea of our approach is to collect and *manage meta-information* on the sources to be integrated. These pieces of information are stored in the *Model Warehouse* of the system, in the form of UML-like models [8], constraints and mappings. This way we can represent structural as well as non-structural information, such as class invariants, implications, etc. The Model Warehouse resides in and is handled by the *Model Manager* component.

We use the term *mediation* to refer to the process of querying SINTAGMA models. Mediation decomposes complex integrated queries to simple queries answerable by individual information sources and, having obtained data from these, composes the results into an integrated form. Mediation is the task of the *Mediator* component.

Access to heterogeneous information sources is supported by *wrappers*. Wrappers hide the syntactic differences between the sources of different kinds, by presenting them to upper layers uniformly, as UML models. These models (called *interface models*) are entered into the Model Warehouse automatically. In the following we give a brief introduction to the main components.

2.1 The Model Manager

The Model Manager is responsible for managing the *Model Warehouse (MW)* and providing integration support, such as model comparison and verification (not covered in this paper). Here we focus on the role of the Model Warehouse.

The content of the MW is given in the language called *SILan* which is based on UML [8] and Description Logics [12]. The syntax of SILAN resembles IDL, the Interface Description Language of CORBA [16]. We demonstrate the knowledge representation facilities of SINTAGMA by a simple SILan example showing the relevant features of the meta-data repository (Figure 2).

```
model Art {
    class Artist:BuiltIns::DLAny {
        attribute String name;
        attribute Integer birthDate;
        constraint self.creation.date > 1900;
    };

    class Work:BuiltIns::DLAny {
        attribute String title;
        attribute String author;
        attribute Integer date;
        attribute String type;
        primary key title;
    };

    association hasWork {
       connection Artist as creator;
       connection Work as creation;
    }; };
```

Fig. 2. SILan representation of the model `Art`

The example describes the model `Art` containing two classes, `Artist` and `Work`. It also contains an association between an artist and her works. We will explain the details of this example within the discussion below.

Semantics of SILan models The central elements of SILan models are classes and associations, since these are the carriers of information. A class denotes a set of entities called the *instances* of the class. Similarly, an n-ary association denotes a set of n-ary tuples of class instances called *links*.

Classes can have *attributes* which are defined as functions mapping the class to a subset of values allowed by the type of the attribute. Classes can inherit from other classes. The instances of the descendant class are all instances of the ancestor class. In our example both `Artist` and `Work` inherit from the built-in class `DLAny` (cf. lines 2 and 8). See Section 3.3 for more details.

Associations have *connections*, an n-ary association has n connections. In an association some of the connections can be named, providing intuitive navigation. For example, the connections of association `hasWork` corresponding to classes `Artist` and `Work` are called `creator` and `creation` respectively (lines 17–18).

Classes are allowed to have a primary key, composed of one or more attributes. This specifies that the given subset of the attributes uniquely identifies an instance of the class. In our example, as a gross simplification, attribute `title` serves as a key in class `Work`, i.e. there cannot be two works (books, for example) with the same title.

Finally, invariants can be specified for classes and associations using the object constraint extension of UML, the OCL language [7]. Invariants give statements about instances of classes (and links of associations) that hold for each of them. The constraint in the declaration of `Artist` (line 5) is an invariant stating that the publication date of each work of an artist is greater than 1900[1]. The identifier `self` refers to an arbitrary instance of the context, in this case the class `Artist`. Then two *navigation* steps follow. In the first step we navigate through the association `hasWork` to an *arbitrary* work of the artist and in the second step from the work to its publication date and state that this is always greater than 1900.

In addition to the object oriented modelling paradigm, the SILan language also supports constructs from the Description Logic (DL) world [12]. This, recently added new feature of SINTAGMA is discussed in Section 3.

Abstractions For mediation, we need mappings between the different sources and the integrated model. These mappings are called *abstractions* because they often provide a more abstract view of the notions present in the lower level models. An example abstraction called `w0` can be seen in Figure 3.

[1] This may be so because the given information source is known to be dealing with works of art of 20th century or later.

This abstraction populates the class `Work` in the model `Art` (line 3) using classes `Product` and `Description`, both from the model `Interface`[2]. This means that the abstraction specifies how to create a "virtual" instance of class `Work`, given that the other two classes are already populated (e.g. they correspond to real information sources). In lines 1–3 the identifiers `m0`, `m1` and `m2` are declared, and these will be used throughout the abstraction specification to denote instances of the appropriate classes.

```
 1 abstraction w0 (m0: Interface::Product,
 2                 m1: Interface::Description
 3              -> m2: Art::Work) {
 4
 5    constraint
 6      m1.id = m0.id and
 7      m1.code = 84
 8    implies
 9      m2.title = m0.title and
10      m2.author = m0.author and
11      m2.date = m0.publication_date and
12      m2.type = m1.description and
13      m2.DL_ID = m0.title;
14 };
```

Fig. 3. SILan representation of the abstraction populating class `Work`

The abstraction describes that given an instance of class `Product` called `m0` and an instance of class `Description` called `m1`, for which the conditions in lines 6–7 hold, there exists an instance `m2` of class `Work` with attribute values specified by lines 9–13[3]. Note that line 6 specifies that the `id` attributes of the two instances have to be the same, and thus corresponds to a relational *join* operation. In our integration scenario `Product` and `Description` actually correspond to real-world Oracle tables containing books and paintings. The task of abstraction `w0` is to convert the records of these tables into instances of class `Work`.

We note that other abstractions can also populate class `Work`. In this case the set of instances of `Work` will be the union of the instances produced by the appropriate abstractions. Note that if a new information source is added, we only have to specify a new abstraction corresponding to this source, while the existing abstractions can be left unchanged.

Notice that the abstraction in Figure 3 takes the form of an implication describing how the given sources can contribute to populating the high level class `Art::Work`. This is characteristic of the Local as View integration approach [4].

[2] In SILan double colons (::) separate the model name from the name of its constituent (class, association, etc.).

[3] Attribute `DL_ID` has a special role as explained in Section 3.3.

2.2 The Wrappers

Wrappers provide a common interface for accessing various information source types, such as relational and object-oriented databases, semi-structured sources (e.g. XML or RDF), as well as Web-services.

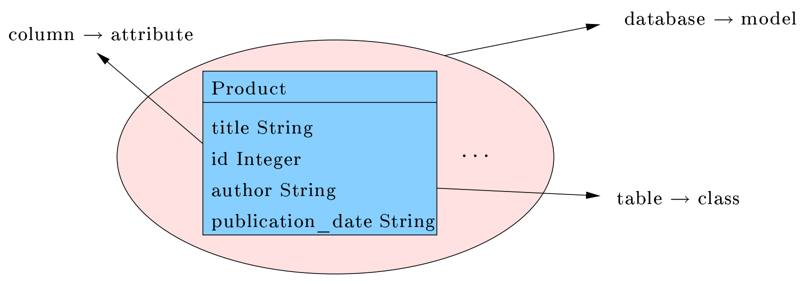

Fig. 4. Modelling relational sources in SILan

A wrapper has two main tasks. First, it extracts meta-data from the information source and delivers these to the Model Manager in the form of SILan models. For example, in case of relational sources, databases correspond to models, tables to classes, columns to attributes, as shown in Figure 4 (cf. class `Product` in abstraction `w0` presented in Figure 3).

The other principal task of a wrapper is to transform queries, formulated in terms of this interface model, into the format required by the underlying information source, and thus allow for running queries on the sources.

2.3 The Mediator

The Mediator [1] supports queries on high level model elements by decomposing them into interface model specific questions. This is performed by creating a query plan satisfying the data flow requirements of the sources. During the execution of this query plan the data transformations described in the abstractions are carried out.

Whenever we query a model element in SINTAGMA, the Model Manager basically provides the following two kinds of information to the Mediator:

1. the query goal itself, i.e. a Prolog term representing what to query;
2. set of mediator rules, using which the mediator can decompose the complex query into primitive ones (i.e. queries that refer only to interface models).

For example, let us consider the query shown below involving class `Work`.

```
query RecentWork
  select *
  from w: Art::Work
  where w.date > 2000;
```

This query is looking for recent works, namely those instances of the class `Art::Work` that were created after 2000[4]. In this case, the query goal is similar to the following simple Prolog expression:

```
:- 'Work:class:220'(DT, [A, B, C, D, E], DA), C > 2000.          (1)
```

Here, the first Prolog goal corresponds to an instance of `Art::Work`. The variables in this term will be instantiated during query execution. The predicate name `'Work:class:220'` is composed from three parts: the kind of the model element (`class`) and its unique internal identifier (220), preceded by the unqualified—and thus non-unique—SILan name (`Work`), provided for readability. Model elements are often referred to by *handles* of form *Kind(Id)*, e.g. `class(220)`. The above predicate name in fact represents the *static type* of the instances queried for.

The first argument of the goal is the dynamic type of the instance, i.e. the handle of the class which, in case of inheritance, can be different from the static type. The second argument contains the values of the static attributes, in this case we have five such variables (cf. declaration of class `Work` in Figure 2), e.g. `C` denotes the value of the attribute `date`. The third and last argument of the query term carries the values of the dynamic attributes. These represent the additional attributes (not known at query time) of the instance if it happens to belong to a *descendant* class of `Art::Work`. Note that in the current implementation of SINTAGMA, as a simplification, the third query argument contains the list of both static and dynamic attributes.

The second part of the query goal corresponds to a simple arithmetic OCL constraint, which uses variable `C` representing the `date` attribute of the work in question.

The mediator rules representing the abstraction `w0` shown in Figure 3 take the following form:

```
'Product:class:190'(_,[A,B,C,D],_),
'Description:class:191'(_,[84,E,B],_) --->
            'Work:class:220'(class(220),[D,A,C,D,E],[D,A,C,D,E])
```

The specific rule above describes how to create an instance of the class `Work` whenever we have two appropriate instances of classes `Product` and `Description` available. If there were more abstractions, the Mediator would get more rules as there would be more than one possible way to populate the given class.

Note that the mediator rules are also used to describe inheritance between model elements. In such a case the dynamic type of the model element on the right hand side of the rule is a variable (as opposed to the constant `class(220)` above). This variable is the same as the dynamic type of the model element on the left hand side. The dynamic attributes are propagated similarly.

[4] We could have created a class named `RecentWork` and populated it by an appropriate abstraction. Then, instead of formulating a SILan query, we could have simply directly asked for the instances of this class. The question whether to use a query or an abstraction is a modelling decision.

Finally, let us mention that an n-ary association is implemented as an n-ary relation, each argument of which is a ternary structure corresponding to a class instance, similar to the one appearing in (1). For example, a query goal for the association `hasWork` (cf. Figure 2) has the following form:

```
:- 'hasWork:association:227'(
       'Artist:class:218'(DT1,[A,B,C],DA1),                     (2)
       'Work:class:220'(DT2,[D,E,F,G,H],DA2)
    ).
```

3 DL modelling in SINTAGMA

Let us now introduce the new DL modelling capabilities of the SINTAGMA system. First we discuss why we need Description Logic models during the integration process and we provide an introductory example. Then we present the DL constructs supported by our system and discuss the restrictions we place on their usage. Finally, we summarise the tasks of the integration expert when using DL elements during integration.

3.1 Introduction

In the Model Warehouse we handle models of different kinds. We distinguish between *application* and *conceptual models.* The application models represent existing or virtual information sources and because of this they are fairly elaborate and precise. Conceptual models, however, represent mental models of user groups, therefore they are vaguer than the application models.

We argue that to construct such models it is more appropriate to use some kind of ontological formalism instead of the relatively rigid object oriented paradigm. Accordingly, we extended our modelling language to incorporate several description logic constructs, in addition to the UML-like ones described earlier. In the envisioned scenario, the high-level models of the users are formulated in description logic and via appropriate definitions they are connected to lower-level models. Mediation for a conceptual model works in the same way as for any other model: the query is decomposed, following the abstractions, until we reach the interface models (in general, through some further intermediate models) which can be queried directly.

Before going into the details, we show an example to illustrate the way how DL descriptions are represented in SILan (note that `Writer` and `Painter` are both descendants of class `Artist`, but otherwise they are normal UML classes.

```
model Conceptual {
  class WriterAndPainter {};
  constraint equivalent {                                       (3)
        WriterAndPainter,
        Unified::Writer and Unified::Painter};
};
```

Here we define the class `WriterAndPainter` by providing a SILan constraint. This constraint can be placed anywhere in the Model Warehouse: in the example above we simply put it in the same model that declares the class `WriterAnd Painter` itself. The constraint actually corresponds to a DL *concept definition axiom*: $\mathsf{WriterAndPainter} \equiv \mathsf{Writer} \sqcap \mathsf{Painter}$. Namely, it states that the instances of class `WriterAndPainter` are those (and only those) who belong to the unnamed class containing the individuals who are both writers and painters. Thus, DL concepts are defined using the Global as View approach [4], as opposed to when populating high-level classes using abstractions (cf. Section 2.1).

Note that the class `WriterAndPainter` could be created without DL support. However, in that case the integration expert would have to go through a much more elaborate process of (1) creating the high level class `WriterAndPainter`, specifying all its attributes and (2) populating it with an appropriate abstraction involving a join. Now, with DL support, she simply formulates a very short and intuitive DL axiom. We argue that this is easier for the expert to do, and it also makes the content of the Model Warehouse more readable to others.

3.2 DL elements in SILan

From the DL point of view, SINTAGMA supports acyclic Description Logic TBoxes containing concept definition axioms formulated in the extension of the $\mathcal{ALCN}(\mathbf{D})$ language (see more below about the extension). Only single atomic concepts, so called *named symbols* can appear on the left hand side of the axioms, such as `WriterAndPainter` in example (3). The remaining atomic concepts, not appearing on the left hand side are called *base symbols*. Such a TBox is definitorial, i.e. the meaning of the base symbols unambiguously defines the meaning of the named symbols. The base symbols, in our case, correspond to normal SINTAGMA classes and associations, e.g. `Writer` and `Painter` in the example (3). The ABox is a set of concept and role assertions, as determined by the instances of the classes which correspond to the base symbols participating in the TBox.

The DL concept constructors supported by SINTAGMA and their SILan equivalents are summarised in Table 1. Note that this table actually describes the possible concept formats on the right hand side of a definition axiom, assuming that we have *expanded* the TBox[5].

The only non-classical DL element in Table 1 is the concrete domain restriction (the last line in the table). Such a restriction specifies a subset of instances of the base concept A for which the given OCL constraint holds. This is a generalisation of the idea of concrete domains in the Description Logics world. Below we show an example of a concrete SILan restriction describing those works whose type (i.e. the value of the attribute `type`) is "`painting`".

```
{class constraint Art::Work satisfies self.type="painting"}
```

The reason we allow only concept *definition* axioms is that we aim to use DL concepts to describe executable high-level views of information sources. In this

[5] The expanded version of an acyclic TBox is another TBox where every named symbol on the right hand side of the axioms is substituted by its definition.

Name	**Syntax**	**SILan equivalent**
Base concept	A	UML class
Atomic role	R	UML association
Top	$\top$	`DLAny`
Bottom	$\perp$	`DLEmpty`
Negation	$\neg C$	`not C`
Intersection	$C \sqcap D$	`C and D`
Union	$C \sqcup D$	`C or D`
Value restrictions	$\forall R.C$	`slot constraint R all values C`
Existential restrictions	$\exists R.C$	`slot constraint R some value C`
Number restrictions	$\bowtie nR$	`slot constraint R cardinality i..j`
Concrete restriction	—	`class constraint A satisfies OCL`

Table 1. (extended) DL concept constructors supported in SILan

sense a DL concept is actually a syntactic variant of a SILan query or a SILan class populated by an abstraction.

Note that this also implies that we use the Closed World Assumption (CWA) in DL query execution. We argue that this is appropriate because of the following three reasons. First, CWA automatically ensures that our DL constructs are semantically compatible with other constructs in the SINTAGMA system. Second, we argue that the Open World Assumption(OWA) is applicable when we have only partial knowledge and would like to determine the consequences of this knowledge, true in every universe in which the axioms of this partial knowledge hold. In contrast with this, in the context of information integration, our users would like to consider a single universe, in which a base concept or a role denotes *exactly* those individuals (or pairs of individuals) which are present in the corresponding database. To illustrate this issue, let us consider the following example: the concept of novice painter is defined to contain painters having at most 5 paintings (for example, being a novice painter may be a precondition for a government grant). To model this situation, the integration expert creates the DL axiom shown below.

$$\mathsf{NovicePainter} \equiv \mathsf{Painter} \sqcap (\leqslant 5\,\mathsf{hasPainting})$$

However, querying this concept, using OWA, will provide no results in general as an open world reasoner would return an individual only if it is *provable* that it has no more than 5 paintings. Practically, this is not what the information expert wants.

The third reason why we decided to use the closed world assumption is the fact that we have huge amounts of data in the underlying databases. Traditional, tableau based DL reasoners do not cope well with large ABoxes [10]. Resolution based DL proving techniques [13] do much better, but they are either still not fast or not expressive enough [15]. By using CWA we can implement DL queries using the well researched, efficient database technology.

3.3 Modeling methodology and tasks of the integration expert

The integration expert is responsible for creating the DL axioms. Although these are represented in SILan within the SINTAGMA system, the expert can use any available OWL editor to create OWL descriptions. These descriptions then can be loaded by the OWL importer of the SINTAGMA system that basically realises an OWL-SILan translation (cf. the "Model Im(Ex)port" box in Figure 1).

One thing the expert should take care of is to match the names of the base symbols and the corresponding SINTAGMA classes and associations. This is often done in two steps: first the integration expert creates concept definition axioms using the widely accepted terminology of the domain, not paying attention to the names of the model elements in the Model Warehouse. Next, the expert provides additional definition axioms for each base symbol connecting it with the proper model element. For example, we could use names `A` and `B` instead of `Writer` and `Painter` in (3), provided that we have the following axioms:

$$\mathtt{A} \equiv \mathtt{Writer}$$
$$\mathtt{B} \equiv \mathtt{Painter}$$

Another important issue is to decide how to identify the instances of the base concepts, e.g the instances of the class `Writer` and class `Painter`. Without this, it is not possible to determine the instances of class `WriterAndPainter`.

In a traditional DL ABox, an instance has a name that unambiguously identifies it. Unambiguity is guaranteed because DL reasoning systems use the so called unique name assumption, i.e. they assume that two different instance names denote different elements in the domain.

In SINTAGMA, similarly to databases, an instance is identified by the subset of its attribute values, e.g. two writers could be considered to be the same if their names match. In other words, this means that `name` is a key in class `Writer`.

The problem is that such keys are fairly useless when we compare instances of different sources. This is because, in general, we cannot draw any direct conclusion from the relation of the keys belonging to instances from different classes. For example, databases containing employees often use numeric IDs as keys. Having two employees from different companies with the same ID does not mean that we are talking about the same person. Similarly, if the IDs of the employees do not match, they are not necessarily different persons.

What we need is some kind of shared key that uniquely identifies the instances of the classes participating in DL concept definitions. Luckily, the object-oriented paradigm we use in SINTAGMA provides a nice way to have such identifiers.

We have mentioned earlier that in SINTAGMA the notion of DL concept is a syntactic variant of SINTAGMA class. This also means that the result of a DL query is an ordinary instance that has to belong to some class(es). For example, when we are looking for the instances that are in both classes `Writer` and `Painter` we are actually interested in an *artist* instance belonging to these classes simultaneously. This is true in general: whatever DL concept constructs we use to describe a DL concept the result must belong to some class that is a common ancestor of the classes involved.

Instead of asking the integration expert to define such common ancestor classes in an ad hoc way, we introduce the built-in class `DLAny`. This class corresponds to the DL concept top ($\top$) and it has only one attribute called `DL_ID` which is a key. We require that all the classes participating in DL concept definitions are the descendants of `DLAny`[6] (cf. lines 2 and 8 of Figure 2). Because of the properties of generalisation attribute `DL_ID` will be a key in all of the descendant classes, i.e. it will exactly serve as the global identifier we were looking for.

Now, the task of the integration expert is to assign appropriate values to the `DL_ID` attributes: she needs to extend the existing abstractions populating the base symbols (classes) to also consider the attribute `DL_ID`. By appropriate values we mean that the `DL_ID`s of two instances should match if these instances are the same, and should differ otherwise. An example for this can be seen in Figure 5 populating the class `Writer`.

```
 1 abstraction ap (m0: Interface::Member ->
 2                 m1: Unified::Writer) {
 3
 4    constraint let n = m0.fname.concat(" ").concat(m0.lname) in
 5      m1.name = n and
 6      m1.birthDate = m0.date and
 7      m1.id = m0.iwa and
 8      m1.style = m0.style and
 9      m1.DL_ID = n;
10 };
```

Fig. 5. Populating the `DL_ID` attribute of a base concept

This abstraction populates the class `Writer` from an interface class called `Member` (lines 1–2). Let us assume that the members of this association have some kind of a unique identifier, such as the membership number of an imaginary "International Writer Association" (IWA), present in the underlying database. It may be worth bringing this key to the class `Writer` (line 7) as it makes possible to find writers efficiently if they happen to be IWA members. However, the unique identifier from the DL point of view has to be different: in fact it is the concatenation of the first and last name of the writer (line 4 and 9).

This is because the class `Writer` can also be populated from other sources where the IWA number makes no sense. Furthermore, we may want class `Writer` to be a descendant of class `Artist`, together with some other classes, such as

[6] Note that this is a necessary condition. As for any concept C, $C \sqsubseteq \top$ holds, any DL instance has to belong to the class corresponding to $\top$, i.e. to `DLAny`.

`Painter`. This requires a key that can be computed from all the underlying sources, such as the name of the artist[7].

To summarise, the integration expert has to perform the following tasks when DL modelling is used during the integration process:

1. declare DL classes and for each provide corresponding definition axioms;
2. ensure that each base concept appearing in the definition axioms is:
   (a) inherited from class `DLAny`,
   (b) populated properly, i.e. its `DL_ID` attribute is filled appropriately.

4 Querying DL models in SINTAGMA

Now we turn our attention to querying DL concepts in SINTAGMA. As described in Section 2.3 our task is to create a *query goal* and a set of *mediator rules*. When we query a DL class, we only generate mediator rules for the base symbols. As these are ordinary classes and associations, this process is exactly the same as the one we use for cases without any DL construct involved.

Recall that a SINTAGMA instance is characterised by three properties, as exemplified by (1) on page 7: its dynamic type `DT`, its static attributes `SA` and its dynamic attributes `DA`. A DL class has only a single static attribute, the `DL_ID` key. However, in contrast with an object oriented query, a DL query may return an answer that has multiple dynamic types. For example, when we enumerate the class `WriterAndPainter` we get instances that belong to both classes `Writer` and `Painter`. Accordingly, an answer to a DL query takes the form of a pair `(ID, DTAs)`, where `ID` is the value of the `DL_ID`, while `DTAs` = `[`$\mathtt{DT_1}$`-`$\mathtt{DA_1}$`,`$\mathtt{DT_2}$`-`$\mathtt{DA_2}$`,...]` is the list of the dynamic types of the answer, each paired with the corresponding dynamic attribute list.

The algorithm specifying what goals to create from a DL concept description is summarised in Figure 6. Here we describe a function Φ_C which, given an arbitrary concept C, returns the corresponding query goal with two arguments, `ID` and `DTAs`. We define this function case by case.

If we have a base class, we simply create a query term representing the instances of the class, similar to the one in goal (1). If we have the intersection of two concepts C and D, we recursively transform concepts C and D and put them in a Prolog conjunction. The union of classes is similar: we create a Prolog disjunction. Negation $\neg C$ is implemented by enumerating the `DLAny` class, and removing those instances which belong to C. The expensive `DLAny` enumeration can be avoided when the negation appears in a conjunction (which is normally the case).

The more interesting cases involve associations. Here R denotes the association itself, while R^D and R^R denote the classes that are the domain and the range of association R, respectively. Recall that a binary association is represented by a binary relation with ternary structures as arguments as in (2).

[7] This is also a simplification. More realistically, the key could be the name together with the birth date.

$$\begin{aligned}
\Phi_A(\texttt{ID, DTAs}) &= A(\texttt{DT, [ID|\_], DA), DTAs = [DT-DA]} \\
\Phi_{C \sqcap D}(\texttt{ID, DTAs}) &= \Phi_C(\texttt{ID, DTAs1}), \Phi_D(\texttt{ID, DTAs2}), \texttt{DTAs = DTAs1} \oplus \texttt{DTAs2}, \\
&\quad \text{where } \oplus \text{ denotes the (compile time) concatenation of lists} \\
\Phi_{C \sqcup D}(\texttt{ID, DTAs}) &= (\Phi_C(\texttt{ID, DTAs})\ \texttt{;}\ \Phi_D(\texttt{ID, DTAs})) \\
\Phi_{\neg C}(\texttt{ID, DTAs}) &= \texttt{DLAny(ID, DTAs), \textbackslash+ } \Phi_C(\texttt{ID, \_}) \\
\Phi_{\exists R.C}(\texttt{ID, DTAs}) &= R(R^D(\texttt{DT, [ID|\_], DA), } R^R(\texttt{\_, [ID2|\_], \_)),} \\
&\quad \Phi_C(\texttt{ID2, \_), DTAs = [DT-DA]} \\
\Phi_{\forall R.C}(\texttt{ID, DTAs}) &= R^D(\texttt{DT, [ID|\_], DA), DTAs = [DT-DA],} \\
&\quad \texttt{\textbackslash+ (}R(R^D(\texttt{DT, [ID|\_], DA), } R^R(\texttt{\_, [ID2|\_], \_)),} \\
&\quad \texttt{\textbackslash+ } \Phi_C(\texttt{ID2, \_))} \\
\Phi_{\bowtie nR}(\texttt{ID, DTAs}) &= \texttt{aggregate([DT, ID, DA], [S=cnt(0)],} \\
&\qquad\qquad R(R^D(\texttt{DT, [ID|\_], DA), } R^R(\texttt{\_, \_, \_))),} \\
&\quad \texttt{condition}_{\bowtie}\texttt{(n, S), DTAs = [DT-DA]}
\end{aligned}$$

Fig. 6. Transforming DL constructs into query goals

The existential restriction $\exists R.C$ is simply transformed to a query of the association R and the concept C. The goal corresponding to a value restriction $\forall R.C$ first enumerates the domain of R and then uses double negation to ensure that the given instance has no R-values which do not belong to C. Finally, a number restriction ($\bowtie nR$) is transformed into a goal which uses a `bagof`-like Prolog predicate `aggregate/3` to enumerate the instances in the domain of R together with the number of R-values connected to them, and then simply applies the appropriate arithmetic comparison.

A concrete restriction involving a base concept A and an OCL constraint O is transformed in a straightforward way into the query goal as shown below[8]:

$$\Phi_A(\texttt{ID, DTAs), DTAs = [DT-DA], } \Psi_O(\texttt{ID, DA})$$

In all formulas so far, `ID` denotes the value of the attribute `DL_ID` containing the unique name of the DL instances (see Section 3.3). Here we make use of the fact that these attributes are always placed first in the static attribute list of an instance. To illustrate the general algorithm, two example transformations are shown in Figure 7. The second example involves an association `hasPainting` and a class `Modern` (representing, say, contemporary pieces of art).

[8] Here, Ψ_O`(ID, DA)` is the Prolog translation of the OCL constraint O. This is an “old” feature, implemented before the introduction of DL extension into SINTAGMA.

```
Class to query: WriterAndPainter
DL definition:  Writer ⊓ Painter
Query goal:     'Writer:class:234'(DT1,[ID|_],DA1),
                'Painter:class:236'(DT2,[ID|_],DA2),
                DTAs = [DT1-DA1,DT2-DA2]

Class to query: ModernPainterWriter
DL definition:  Writer ⊓ ∃hasPainting.Modern
Query goal:     'Writer:class:234'(DT1,[ID|_],DA1),
                'hasPainting:association:142'(
                     'Artist:class:218'(DT2,[ID|_],DA2),
                     'Work:class:220(_,[ID2|_],_)),
                'Modern:class:237'(_,[ID2|_],_),
                DTAs = [DT1-DA1,DT2-DA2]
```

Fig. 7. Transformation examples

5 Related work

The two main approaches in information integration are the Local as View (LAV) and the Global as View (GAV) [4]. In the former, sources are defined in terms of the global schema, while in the latter, the global schema is defined in terms of the sources (similarly to the classical views in database systems). Information Manifold [14] is a good example for a LAV system. Examples for the GAV approach include the Stanford-IBM integration system TSIMMIS [6].

In SINTAGMA we apply a hybrid approach, i.e. we use both LAV and GAV. When using abstractions to populate high-level classes we employ the LAV, while in case of DL class definitions we use the GAV approach.

There are several completed and ongoing research projects in the area of using description logic-based approaches for both Enterprise Application Integration (EAI) and Enterprise Information Integration (EII) as well.

The generic EAI research stresses the importance of the Service Oriented Architecture, and the provision of new capabilities within the framework of Semantic Web Services. Examples for such research projects include DIP [11] and INFRAWEBS [9]. These projects aim at the semantic integration of Web Services, in most cases using Description Logic based ontologies and Semantic Web technologies. Here, however, DL is used mostly for service discovery and design-time workflow validation, but not during query execution.

On the other hand, several logic-based EII tools use Description Logics and take a similar approach as we did in SINTAGMA. That is, they create a DL model as a view over the information sources to be integrated. The basic framework of this solution is described e.g. in [5,3]. The fundamental difference compared to our approach is that these applications deal with the classical Open World Assumption, therefore their task can be viewed as an ABox instance retrieval task. However, as already discussed in Section 3.2, one problem with this is that the ABox is distributed among the underlying heterogenous databases which therefore can be extremely big. We argue that existing DL reasoners are not usable when this amount of data and complex DL queries are involved.

6 Conclusions

In this paper we presented the DL extension of the information integration system SINTAGMA. This extension allows the information expert to use Description Logic based ontologies in the development of high abstraction level conceptual models. Querying these models is performed using the Closed World Assumption over the underlying information sources.

We have presented the main components of the SINTAGMA system: the Model Manager which is responsible for the Model Warehouse, the Wrapper, which provides a uniform view over the heterogenous information sources and the Mediator, which decomposes complex high-level queries into primitive ones answerable by the individual information sources.

Next, we have described the DL modelling elements the integration expert can use when building conceptual models and we have also discussed the modelling methodology she has to follow. We have presented the way how DL queries are executed within the SINTAGMA system. Finally, we have illustrated our approach by providing a use case about artists.

We argue that our solution for combining DL and UML modelling in a unified integration framework provides a viable alternative to existing systems. The usage of DL constructs in building high-level conceptual models has substantial benefits, both in terms of modelling efficiency and maintenance.

Acknowledgements

The authors acknowledge the support of the Hungarian NKFP programme for the SINTAGMA project under grant no. 2/052/2004. We would also like to thank all the people participating in this project, and especially Tamás Benkő.

References

1. Liviu Badea and Doina Tilivea. Query Planning for Intelligent Information Integration using Constraint Handling Rules, 2001. IJCAI-2001 Workshop on Modeling and Solving Problems with Constraints.
2. T. Benkő, P. Krauth, and P. Szeredi. A logic based system for application integration. In *Proceedings of the 18th International Conference on Logic Programming, ICLP 2002*. Springer, LNCS, 2002.
3. A. Borgida, M. Lenzerini, and R. Rosati. Description logics for databases. In *Description Logic Handbook*, pages 462–484, 2003.
4. D. Calvanese, D. Lembo, and M. Lenerini. Survey on methods for query rewriting and query answering using views. Tech. report, University of Rome, April 2001.
5. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Principles of Knowledge Representation and Reasoning*, pages 2–13, 1998.

6. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.
7. T. Clark and J. Warmer, editors. *Object Modeling with the OCL: The Rationale behind the Object Constraint Language*, volume 2263 of *LNCS*. Springer, 2002.
8. Martin Fowler and Kendall Scott. *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley, 1997.
9. Vladislava Grigorova. Semantic description of web services and possibilities of BPEL4WS. *Information Theories and Applications*, 13(2):183–187, 2006.
10. V. Haarslev and R. Möller. Optimization techniques for retrieving resources described in owl/rdf documents: First results. In *Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR 2004, Whistler, BC, Canada, June 2-5*, pages 163–173, 2004.
11. M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic business process management: A vision towards using semantic web services for business process management, 2005.
12. Ian Horrocks. Reasoning with expressive description logics: Theory and practice. In *Proc. of the 18th Int. Conf. on Automated Deduction (CADE 2002)*, number 2392 in Lecture Notes in Artificial Intelligence, pages 1–15. Springer, 2002.
13. U. Hustadt, B. Motik, and U. Sattler. Reasoning for description logics around $\mathcal{SHIQ}$ in a resolution framework. Technical Report 3-8-04/04, June 2004.
14. T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In C. Knoblock and A. Levy, editors, *AAAI Spring Symposium ob Information Gathering from Heterogeneous, Distributed Environments*, 1995.
15. Zsolt Nagy, Gergely Lukácsy, and Péter Szeredi. Translating description logic queries to Prolog. In *Proc. of PADL, Springer LNCS 3819*, pages 168–182, 2006.
16. Interface Definition Language. ISO International Standard, number 14750.