A Deductive Data Model for Representing and Querying Semistructured Data

Fosca Giannotti[1], Giuseppe Manco[1], Dino Pedreschi[2]

[1]CNUCE Institute of CNR
Via S. Maria 36, 56125 Pisa, Italy
e-mail: F.Giannotti@cnuce.cnr.it, G.Manco@guest.cnuce.cnr.it

[2]Dipartimento di Informatica, Univ. Pisa
Corso Italia 40, 56125 Pisa, Italy
e-mail: pedre@di.unipi.it

Abstract

We show how the mechanisms of an active/deductive object-oriented model are capable of modeling the information available on the World Wide Web, the paradigmatic example of a large, distributed collection of poorly structured, heterogeneous information, consisting of documents connected by hypertext links. We exhibit the basic features of the object model, including the schema definition language, the query language with multiple roles, the basic update operations, and a form of active rules, and a compilation of the mentioned features into a fragment of $\mathcal{LDL}$++, which is essentially Datalog extended with non-determinism and a form of stratified negation. The proposed compilation has a twofold aim. On one side, it should provide an abstract implementation level, where the object model is declaratively reconstructed and its semantics clarified. On the other side, the proposed compilation should form the basis of a realistic implementation, as $\mathcal{LDL}$++ can be efficiently executed, and supports real side effects.

1 Introduction

This paper is aimed at showing the suitability of an integrated active/deductive/object-oriented data model—ADOOD in short—to deal with semistructured data. In particular, we show how the mechanisms of such an integrated model are capable of modeling the information available on the World Wide Web, the paradigmatic example of a large, distributed collection of poorly structured, heterogeneous information, consisting of documents connected by hypertext links. Finding documents of interest on the Web is clearly connected with the ability of browsing, searching and querying semistructured, hierarchical information. Effective means for querying the Web are not supported by the current technology, as both database and information retrieval systems revealed largely inadequate to this purpose. This deficiency has motivated a rising interest on this subject—see e.g., [18, 15, 9, 3].

We consider in this paper ADOOD, a small language embodying the essential aspects of an active/deductive object data model. We show how this language is appropriate to manage and query semistructured data, and capable to address crucial aspects such as path expressions, and semantic integration of eclectic objects. We argue that the achieved high degree of expressiveness and flexibility is made possible by the integration of the object-oriented model with deductive and active rules in a coherent overall framework.

The static component of the ADOOD model, i.e., its data definition and query language, has many similarities with other integrated DOOD models, such as *F-logic* [12], but differs substantially in that it provides a simple yet powerful way of deriving new objects with a structure different from that specified in the schema, by means of the deductive query language. As an example, an object can be associated with multiple roles, as well as with attributes not previously defined in the schema. The proposed ADOOD model has also a dynamic component, i.e. the basic update and object-migration operations, inspired by the analogous features of the Fibonacci language [5], and comprises a simple form of active rules, or *condition-action* rules.

The semantics of the proposed ADOOD model is given by means of a *compilation* of this language into an extension of Datalog^{1s} with non-deterministism, which corresponds to a fragment of the logical data language $\mathcal{LDL}$++ [6]. More precisely, the target language is an extension of deductive databases with

- non determinism, used among others to realize object identifiers, in a way similar to [22], and the non-deterministic semantics of active rules, and
- a form of stratification, used to realize updates, object-migration and, in general, actions against the database.

The code obtained as a result of the compilation can be understood in declarative terms, thus providing a formal interpretation of the source object-oriented program. But more importantly, such a code can be executed by the ordinary deductive, fixpoint-based computation integrated with efficient support for updates and active rules by means of side effects. In other words, this compilation is low-level enough to form the basis of a realistic (prototype) implementation on top of $\mathcal{LDL}$++. A major advantage of this opportunity is the fact that the architecture of the abstract machine supporting deductive databases is left unaltered and, as a consequence, the available optimization techniques, such as magic sets, are directly applicable.

The details of the compilation of the ADOOD model are limited in this paper to what is absolutely needed to understand the language and its application to the management of semistructured information. We refer the reader to [10] for the complete semantics of the ADOOD model.

Related Work

The very basic idea of a model for semistructured data management is given in the *object exchange model* proposal (OEM, [17, 2, 1]). There, data is represented by a 4-

tuple $\langle Label, Type, Value, OID \rangle$. Within such a schema, in fact, we can represent both atomic objects (whose value is of a simple type) and relations between objects, describing a sort of hierarchy. Notice that, altough such a representation is closely related to relational representation, the presence of object identifiers introduces a substantial difference between the two models.

Many declarative models have been introduced for querying and restructuring the World-Wide Web [13, 15, 1, 14]. As a paradigmatic example, take the *WebLog* language, that has many similarities with the ADOOD model. In the *WebLog* language, a first class status to hyperlinks is provided. This novelty has two main advantages. First, the user can specify partial information on the traversals to be done to answer a query. Second, the query processor can exploit this information to query directly on the Web in an efficient way. The WebLog can be considered as a proper subset of the ADOOD model (which can be seen as an extension of F-logic [12]). Actually, the conceptual model proposed by the object-oriented paradigm combined with deductive and higher-order features is adequate enough for the representing and querying semistructured data. Moreover, we show here how the formal semantics of the ADOOD model can be expressed in terms of Datalog++ [10], that covers both static and dynamic aspects of the model.

Many proposals in the literature enhance deductive databases to support the object-oriented paradigm, but we found no reference framework, which models both static and dynamic aspects in a uniform way. A first class of proposals tackle only static (structural) aspects. This is the case, for example, of LOGIN [4], or of its extension F-logic [12]. As far as the dynamic aspects are concerned, there are various proposals, which provide a procedural or declarative semantics, as in the case of [16, 7, 8]. In either cases, static and dynamic aspects are dealt with in separate frameworks. On the contrary, we argue in [10] that Datalog++ provides a flexible uniform framework, and moreover enables us to handle sophisticated dynamic features, such as role-dynamics and object-migration, widely recognized as key issues in the object-oriented paradigm [5, 19, 21]. We argue that the achieved high degree of expressiveness and flexibility is made possible by the integration of the object-oriented model with deductive and active rules in a coherent overall framework.

In [3, 1], a model for Web-computability is presented. Such a model can be seen, to a greater extent, a model for semistructured-information computability. It is worth noting how many of the queries that are web-computable in the sense of [3](i.e., computable and eventually computable queries), are expressible in the ADOOD model. Moreover, the combined use of negation and recursive definitions allow to express queries that are not web-computable[1].

[1] such a result can be seen as a direct consequence of the fact that the semantics of the ADOOD model is expressed by means of Datalog++

2 The ADOOD Data Model—a brief overview

We provide in this section a brief overview of the main mechanisms of the ADOOD model. We use basic Web entities, such as HTML documents and hyperlinks, as an example to illustrate the modeling capabilities of ADOOD.

A class can be specified in three different, non mutually exclusive, ways:

1. by explicit definition—i.e., by listing its attributes, as in the following definition of the classes `document` and `hRef`, which define the basic Web entities:

$$\texttt{document}[\texttt{title} \Rightarrow \texttt{string},\ \ \texttt{text} \Rightarrow \texttt{string}]$$
$$\texttt{hRef}[\texttt{source} \Rightarrow \texttt{document},\ \ \texttt{label} \Rightarrow \texttt{string},\ \ \texttt{destination} \Rightarrow \texttt{webObj}]$$

2. by specialization from other classes—for instance, the class `document` can be seen as a subclass of `webObj` class, defining common properties for any web objects—Java applets, image files, quicktime movies, etc.:

$$\texttt{document}\ \ \textbf{isa}\ \ \texttt{webObj}$$

3. by derivation from other classes by means of conditions—for instance, the following rule qualifies as an object of the `document` class any file with `read` permission which can be parsed as an HTML file, whose URL can be determined from the local path of the file and the address of the local host, and used as its unique *oid*:

$$\begin{aligned}\texttt{U} : \texttt{document}[\texttt{title} \rightarrow \texttt{T},\ \ \texttt{text} \rightarrow \texttt{X}] \leftarrow\ & \\ & \texttt{F} : \texttt{file}[\texttt{content} \rightarrow \texttt{X}, \texttt{path} \rightarrow \texttt{P}, \texttt{host} \rightarrow \texttt{H}, \texttt{mode} \rightarrow \texttt{read}] \\ & \wedge\ \texttt{html\_parse}(\texttt{X}, \texttt{T}), \\ & \wedge\ \texttt{oid\_url}((\texttt{P}, \texttt{F}, \texttt{H}), \texttt{U}).\end{aligned}$$

By means of these mechanisms, a schema can be defined, which, even in the case of poorly structured data, gathers the little common structural information about Web objects—documents and references in our example.

Objects can be viewed as members of different classes: we equivalently say that an object may be viewed in specific *roles*. Referencing to objects in specific roles is obtained by means of atoms of the form `oid : class[attr_val_list]`, whose declarative reading is: the object identified by `oid`, viewed in the role `class`, has attribute values (properties) as specified in `attr_val_list`. The derivation rule (r_1) is based on this syntax. The property list `attr_val_list` may contain either pairs of the form [`attr` → `val`], where `attr` is defined in the schema, or pairs of the form [`mthd` @ `val`], where `mthd` is a new derived attribute, or a *method*, not defined in the schema. For example, a `document` object with URL `U` and an associated image can be seen as:

$$\texttt{U} : \texttt{document}[\texttt{img}\ @\ \text{‘}\texttt{\$HOME/public\_html/image.gif}\text{’}]$$

Both incomplete and derived information is allowed, in the sense that attribute lists may be partially specified in rule heads, as well as new attributes (methods) can be introduced. The following rule (r_2) defines the `img` method, not included in the original definition of class `document`, representing the set of images (`.gif` files) associated with a document. Set-valued attributes are handled by means of the set-grouping `< >` construct of $\mathcal{LDL}$++ using some convenient abbreviations of ordinary string operations needed to deal with file paths.

$$\begin{array}{l} \mathtt{U : document[img} \mathbin{@} < \text{`}\mathtt{\$HOME/public\_html/'IMG\_NAME`.gif'} >] \leftarrow \\ \qquad \mathtt{H : hRef[source} \rightarrow \mathtt{U, destination} \rightarrow \mathtt{W]} \\ \qquad \wedge\ \mathtt{W : file[path} \rightarrow \text{`}\mathtt{\$HOME/public\_html'}, \\ \qquad\qquad \wedge\ name \rightarrow \mathtt{IMG\_NAME`.gif'].} \end{array} \qquad (r_2)$$

Here, string concatenation is denoted by direct juxtaposition, to deal with file paths. Set-valued attributes, such as `img`, are handled by means of the set-grouping `< >` construct of $\mathcal{LDL}$++. Rules such as (r_1) and (r_2) are a flexible mechanism to define virtual objects, views and derived (sub)classes. It is worth noting that the creation of virtual objects is subject to the constraint of the *oid* uniqueness, so that no new objects are created with *oid*'s already in use.

The dynamic component of the ADOOD model consists of a repertoire of five basic update operations, or *actions*, called `new`, `delete`, `modify`, `extend` and `drop`, where the last two support *migration* of objects from/to different roles. A simple form of active rules is then provided, consisting of production rules which perform actions when triggered by conditions. For instance, the following active rule represents a dynamic integrity constraint, which detects dangling references in documents, and takes an action to repair the inconsistency—the link is redirected to a warning document.

$$\begin{array}{ll} \mathtt{on} & \mathtt{U : document[\,]}\ \wedge \\ & \mathtt{H : hRef[source} \rightarrow \mathtt{U, destination} \rightarrow \mathtt{U_1]}\ \wedge \\ & \neg\, \mathtt{U_1 : webObject[\,]}\ \wedge \\ & \mathtt{O : document[title} \rightarrow \text{`}\mathtt{warning.html'}] \\ \mathtt{do} & \mathtt{modify(H, hRef[destination} \rightarrow \mathtt{O])}. \end{array}$$

2.1 Compiling ADOOD to Datalog^{1s}

The semantics and the implementation of the ADOOD model are given by means of a translation to *Datalog*^{1s} augmented with non-determinism and XY-stratified negation. This target language is based on the notion of a *stage argument*, a natural number which is used to model a database state. Stage arguments are used in a disciplined manner, namely either in X-rules, where the stage is left unaltered in both head and bodies, or in Y-rules, where the stage is incremented by 1 from the body to the head. Negation is XY-stratified in the sense that it can be used in Y-rules on smaller stages only. This restriction leads to a natural interpretation of negation, as well as to the possibility of efficiently executing programs.

Clearly, the stage plays a role in modeling the dynamic aspects of the ADOOD model. In the compilation of the static part of the model, X-rules only are used.

In the $Datalog^{1s}$ representation, classes are represented by predicates which specify the pertinent attributes augmented with two extra arguments, denoting the stage and the *oid*. Therefore, we associate with every object with identifier *oid* belonging to a class p with n attributes, a set of n atoms $attr(j, oid, p_{class}, f_i(x))$, for $i = 1, \ldots, n$, where j is the so-called stage argument, f_i is the i-th attribute of the class p, and x is its value.

The representation of an object by means of an extensible collection of single attributes facilitates the operation of adding structuring information by means of deductive rules.

An ISA relation between two classes, q **isa** p, is modeled in the following way. For each attribute f in the superclass p the following rule is generated if there is no overriding:

$$attr(J, Oid, p, f(X)) \leftarrow attr(J, Oid, q, f(X)).$$

which basically states that each object of the subclass is also an object of the superclass[2]

The proposed translation scheme is based on an abstract object store where objects are represented as instances of the most specialized class they dynamically belong to. In other words, each object is completely specified by its *most specialized version* (*msv* in short) which contains all attributes currently (i.e., at each stage) available for the object. To this purpose, we use the relation $msv(j, oid, q(x))$, which denotes that, at stage j, the tuple x in class q is the most specialized version of object *oid*. The role of *msv* is to activate the deduction process which populates the classes in the whole hierarchy. For each predicate q_{class} and for each attribute f of class q the following clause is defined:

$$attr(J, Oid, q_{class}, f(X)) \leftarrow \quad msv(J, Oid, q_{class}(Y)).$$

where Y is the tuple of variables corresponding to the attributes of the object Oid in class q, X is the variable corresponding to the attribute f in the tuple Y.

To see how deductive rules for deriving views or methods are compiled, we consider for simplicity rules where all atoms are either of the form $oid : p[f \rightarrow t]$ or of the form $oid : p[f@t]$. Intuitively, each new attribute definition is propagated downwards to the subclasses, provided that the pertinent object belong to such subclasses—in a way which recalls the treatment of `self` in o-o languages. Conversely, the conclusions of derivation rules are only automatically propagated upwards, by means of the rules for the **isa** hierarchy. It is needed, however, to check that virtual objects are created only if not already existing: this is accomplished by an admissible use of negation. For example, rule (r_2) above is translated as follows:

[2] Though the approach we follow here is based on static schema definition, a light modification of the above rule can help handling dynamic schema definition. We can, in fact, add in the body of the rule the facts regarding updatable information regarding inheritance relation and overriding information, and generalize the rule for all objects.

$$\mathtt{attr}(\mathtt{J},\mathtt{U},\mathtt{document},\mathtt{img}(<\text{`}\mathtt{\$HOME/public\_html/}\text{'}\mathtt{IMG\_NAME}\text{`}\mathtt{.gif}\text{'}>)) \leftarrow$$
$$\mathtt{attr}(\mathtt{J},\mathtt{U},\mathtt{document},\_),$$
$$\mathtt{attr}(\mathtt{J},\mathtt{H},\mathtt{hRef},\mathtt{source}(\mathtt{U})),$$
$$\mathtt{attr}(\mathtt{J},\mathtt{H},\mathtt{hRef},\mathtt{destination}(\mathtt{W})),$$
$$\mathtt{attr}(\mathtt{J},\mathtt{W},\mathtt{file},\mathtt{path}(\text{`}\mathtt{\$HOME/public\_html}\text{'})),$$
$$\mathtt{attr}(\mathtt{J},\mathtt{W},\mathtt{file},\mathtt{name}(\mathtt{IMG\_NAME}\text{`}\mathtt{.gif}\text{'})).$$

In order to modify the current state of the database (i.e., to add, delete or modify objects), we operate on the stage argument, in a way similar to that used in [23, 20]. The effect of the basic update operations is *virtual*, in the sense that the real update will take place only if the update is in the action part of an active rule enabled for execution. Therefore, the update operations record their potential effects in a *virtual update* relation *vupd*:

$$vupd(j, oid, new\_msv, old\_msv)$$

denoting the fact that, at stage j, the old *msv* of object *oid* is modified as specified in *new_msv*. As a consequence, the translation of the five update operations `new`, `extend`, `drop`, `delete` and `modify` is assigned by means of (deductive) rules which derive entries for the *vupd* relation. The translation of an update action A occurring in an active rule `on` Q `do` A is parametric w.r.t. the precondition Q. For example, the translation of an active rule

$$\texttt{on } Q \texttt{ do modify}(oid, q(t_{new}))$$

is is composed by the following rules, one for each subclass r of q (possibly $r = q$):

$$vupd(s(J), oid, r(t'_{new}), r(t_{old})) \leftarrow Q, msv(J, oid, r(t_{old})).$$

where the tuple t'_{new} is obtained from t_{new} by completing the missing attributes as in tuple t_{old}.

Among all enabled instances of all rules, one is selected non-deterministically, and the corresponding action executed (i.e., a *rupd*–real update–atom is derived from one of the *vupd* rules), by means of the non-deterministic `choice` construct of $\mathcal{LDL}$++ [11]. The next step is to record the real update on the object store, i.e., on the *msv* relation. As a consequence of the assumption that each object has a unique *msv*, we can model updates by simply updating the *msv* relation, as the rules for the management of the hierarchy accomplish the task of reflecting the updates on the whole database. This is done by means of a *frame axiom*, in the style of [20, 23]. Intuitively, the alive objects at the current stage are the newly created (non-*null*) ones, together with those of the previous stage which have not been explicitly changed. The following rule states the above concept.

$$msv(s(J), Oid, New\_msv) \leftarrow rupd(s(J), Oid, New\_msv, \_), New\_msv \neq null.$$
$$msv(s(J), Oid, Old\_msv) \leftarrow msv(J, Oid, Old\_msv), \neg rupd(s(J), Oid, \_, Old\_msv).$$

It should be noted that an efficient execution of this kind of frame rules is feasible, which avoids unnecessary copies of the database, by means of real side effects [10].

3 Semistructured Data Management with the ADOOD

So far, we have introduced a few basic Web objects, by means of classes `webObj`, `document` and `hRef`. A few remarks are in order.

- We made the assumption that the *oid* of any web object coincides with its (unique) URL-address on the Web—see rule (r_1).
- Although not strictly necessary, the class `hRef` has been introduced to achieve a more expressive and efficient Web model. In particular, from the implementation side, it is possible to store locally many references without being compelled to store the associated web objects. A different model has been adopted in most other proposals for modeling semi-structured data— e.g., the *Object Exchange Model*. In our model, the basic notion of OEM can be represented by the ADOOD class definition $\texttt{Label}[\texttt{val} \Rightarrow \texttt{Type}]$ and associated atom $\texttt{OID} : \texttt{Label}[\texttt{val} \rightarrow \texttt{Value}]$.

In the previous section we have seen how the capabilities of the ADOOD model allow schema and definition and restructuring. For instance, consider the DB & LP Web site by Michael Ley, consisting of the file `ley/db/index.html`, located at host `informatik.uni-trier.de`. Then rule r_1 allows the definition of a Web document with URL (oid) U= `http://www.informatik.uni-trier.de/ ley/db/index.html`, with title *Database Systems and Logic Programming*.

Another example of the restructuring capabilities of ADOOD is the following rule, which rebuilds the structure of a Web object on the basis of the references currently available for that object.

$$\begin{array}{l}\texttt{X} : \texttt{webObj}[\texttt{A} \ @ \ \texttt{Y}] \leftarrow \\ \qquad\qquad \texttt{H} : \texttt{hRef}[\texttt{source} \rightarrow \texttt{X},\ \texttt{label} \rightarrow \texttt{A},\ \texttt{destination} \rightarrow \texttt{Y}].\end{array}$$

Observe that by this mechanism Web objects dynamically acquire new attributes, not definable once and for all in the schema. For instance, consider again the DB & LP Web site by Michael Ley, which is the source of a reference with label `PODS` and destination Y. The above rule introduces a derived attribute named `PODS` whose value is Y.

Many Web queries in the *search and browse* model can be naturally expressed in the ADOOD model. We discuss here two important aspects, namely the capability to deal with *path expressions* and *eclectic objects*.

First, we can explore the structure of links by following attribute labels. For example, in order to retrieve the objects reachable from a given object `O` through a path labeled `a.b.c`, we can use the following simple method

$$\begin{array}{l}\texttt{O} : \texttt{webObj}[\texttt{a.b.c.reachable} \ @ \ \texttt{Z}] \leftarrow \\ \qquad\qquad \texttt{O} : \texttt{webObj}[\texttt{a} \ @ \ \texttt{X}] \wedge \texttt{X} : \texttt{webObj}[\texttt{b} \ @ \ \texttt{Y}] \wedge \texttt{Y} : \texttt{webObj}[\texttt{c} \ @ \ \texttt{Z}].\end{array}$$

where, for a generic link label `A`, the following method is defined on Web objects:

$$\begin{array}{l}\texttt{X}:\texttt{webObj}[\texttt{A} \ @ \ \texttt{Y}] \leftarrow \\ \qquad\qquad \texttt{H}:\texttt{hRef}[\texttt{source} \rightarrow \texttt{X},\ \texttt{label} \rightarrow \texttt{A},\ \texttt{destination} \rightarrow \texttt{Y}].\end{array}$$

It should be noted that the above rule employs a limited form of higher-order programming, which is supported by the ADOOD model. A greater flexibility is achieved using recursive (deductive) rules. For instance, the set of all objects reachable from a given object `O` can be retrieved by the following query:

$$\texttt{O}:\texttt{webObj}[\texttt{reachable} \ @ \ \texttt{Y}] \leftarrow \texttt{H}:\texttt{hRef}[\texttt{source} \rightarrow \texttt{O}, \texttt{destination} \rightarrow \texttt{Y}].$$

$$\begin{array}{l}\texttt{O}:\texttt{webObj}[\texttt{reachable} \ @ \ \texttt{Z}] \leftarrow \texttt{H}:\texttt{hRef}[\texttt{source} \rightarrow \texttt{O}, \texttt{destination} \rightarrow \texttt{Y}] \\ \qquad\qquad\qquad\qquad \wedge\ \texttt{Y}:\texttt{webObj}[\texttt{reachable} \ @ \ \texttt{Z}].\end{array}$$

Recursion can be combined with the higher-order programming feature, thus obtaining a web-reachability query generalized to follow arbitrary links as follows:

$$\texttt{O}:\texttt{webObj}[*.\texttt{reachable} \ @ \ \texttt{Y}] \leftarrow \texttt{O}:\texttt{webObj}[\_ \ @ \ \texttt{Y}].$$

$$\begin{array}{l}\texttt{O}:\texttt{webObj}[*.\texttt{reachable} \ @ \ \texttt{Z}] \leftarrow \texttt{O}:\texttt{webObj}[\_ \ @ \ \texttt{Y}] \\ \qquad\qquad\qquad\qquad \wedge\ \texttt{Y}:\texttt{webObj}[*.\texttt{reachable} \ @ \ \texttt{Z}].\end{array}$$

Thus, the objects reachable from `O` along a path expressions of the form $\texttt{a}.*.\texttt{b}$, can be concisely expressed by the query:

$$\begin{array}{l}\texttt{O}:\texttt{webObj}[\texttt{a}.*.\texttt{b}.\texttt{reachable} \ @ \ \texttt{Z}] \leftarrow \\ \qquad\qquad\qquad \texttt{O}:\texttt{webObj}[\texttt{a} \ @ \ \texttt{X}] \\ \qquad\qquad\qquad \wedge\ \texttt{X}:\texttt{webObj}[*.\texttt{reachable} \ @ \ \texttt{Y}] \\ \qquad\qquad\qquad \wedge\ \texttt{Y}:\texttt{webObj}[\texttt{b} \ @ \ \texttt{Z}].\end{array}$$

A systematic treatment of path expression is achievable, by generalizing from the above examples.

The use of negation allows us to define generic queries. For example, the following query retrieves all objects which are not referenced by any other object:

$$\texttt{noref}(\texttt{X}) \leftarrow \neg\ \texttt{connected}(\texttt{X}).$$

$$\texttt{connected}(\texttt{X}) \leftarrow \texttt{Y}:\texttt{webObj}[*.\texttt{reachable} \ @ \ \texttt{X}].$$

Observe that the above query is not Web-computable, in the terminology of [3], in that it requires the exploration of the entire Web *before* returning an answer. This example points out the high expressiveness of ADOOD as a querying and restructuring model for the Web. Also, the example points out the need of identifying fragments of the query language which express computable queries only, by means of a disciplined use of negation.

As a further application of the ADOOD model, we briefly address semantic integration, and the ability to deal with eclectic objects. To this purpose, we use as an example the case of bibliographic BibTeX files. Consider a class `bibTeXEntry` of objects corresponding to BibTeX entries. New such objects are automatically generated, whenever a file is parsed as a collection of BibTeX entries, by means of an active rule like the following, which takes care of BibTeX entries of type `inproceedings`:

$$
\begin{array}{ll}
\texttt{on} & \texttt{F}:\texttt{file}[\texttt{content} \rightarrow \texttt{C}] \ \wedge \\
& \texttt{bibTeXParse}(\texttt{C},\texttt{Bs}) \ \wedge \\
& \texttt{member}((\texttt{proceedings},\_),\texttt{Bs}) \\
\texttt{do} & \texttt{new}(\texttt{I},\texttt{inproceedings}[\texttt{refFile} \rightarrow \texttt{F}])
\end{array}
$$

Here, `inproceedings` is a subclass of `bibTeXEntry`. Newly created entries are associated with the file they were retrieved from. Each `bibTeXEntry` object is formed by a series of properties deriving from the parsing process:

$$
\begin{array}{l}
\texttt{I}:\texttt{inproceedings}[\texttt{author} \ @ \ \texttt{A},\texttt{title} \ @ \ \texttt{T},\texttt{editor} \ @ \ \texttt{E}] \leftarrow \\
\qquad \texttt{F}:\texttt{file}[\texttt{content} \rightarrow \texttt{C}] \\
\qquad \ \wedge \ \texttt{bibTeXParse}(\texttt{C},\texttt{Bs}) \\
\qquad \ \wedge \ \texttt{member}((\texttt{proceedings},[\texttt{A},\texttt{T},\texttt{E}]),\texttt{Bs}) \\
\qquad \ \wedge \ \texttt{I}:\texttt{inproceedings}[\texttt{refFile} \rightarrow \texttt{F}].
\end{array}
$$

Observe that the properties of an entry are derived using methods, obtaining therefore maximum flexibility (different types of entries have different formats), but preserving uniqueness of oid's. Finally, we can derive the subclass of BibTeX files, by viewing as such any file which can be parsed into a collection of BibTeX entries which are associated with it:

$$
\begin{array}{l}
\texttt{F}:\texttt{bibTeXFile}[\texttt{bibTeXEntries} \rightarrow \ < \texttt{B} >] \leftarrow \\
\qquad \texttt{F}:\texttt{file}[\texttt{content} \rightarrow \texttt{C}] \\
\qquad \ \wedge \ \texttt{bibTeXParse}(\texttt{C},\_) \\
\qquad \ \wedge \ \texttt{B}:\texttt{bibTeXEntry}[\texttt{refFile} \rightarrow \texttt{F}].
\end{array}
$$

A `bibTeXFile` object is therefore eclectic in the proper sense that it is at the same time both an ordinary file object with an unstructured content, and a structured BibTeX file consisting of a collection of structured entries.

It is worth noting that we make no assumptions about the format of the `file` object and the `bibTeXParse` procedure: in principle, `biBTeXParse` can be defined in a different way for each (heterogeneous) file format we want to deal with. For instance, the `file` object can be an ASCII file, an MS-Word document, or a table in some relational DBMS—in this latter case `bibTeXParse` is probably implemented as an SQL query. In fact, the architecture of a mediated system for ADOOD is envisaged (in fig. 1), capable of dealing with heterogeneous sources of information. A collection of *wrapper* modules provides the translation of external data items into the format of the ADOOD model, as well as support for (intelligent) caching and retrieval of data. Several such wrappers may be concurrently employed, which provide interfaces to SQL servers, to Web search engines, to index servers, and so on. The current prototype implementation of the abstract machine of ADOOD on top of the $\mathcal{LDL}$++ system provides means to integrate wrappers to heterogeneous data sources, much in the spirit of the desired open architecture.

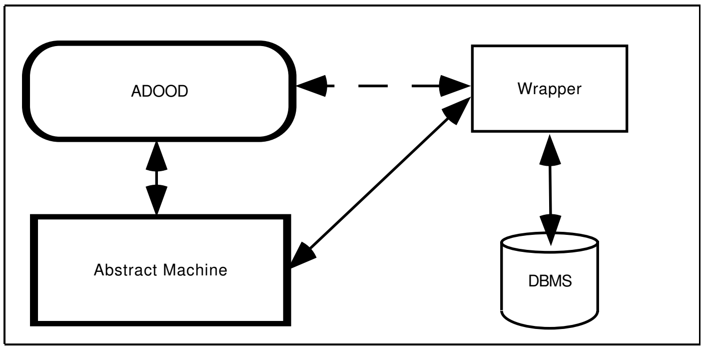

Figure 1: A mediated system architecture for ADOOD.

4 Conclusions

We have discussed in this extended abstract the adequacy of the ADOOD model for the management of semistructured data. We believe that a combined active, deductive and object-oriented data model offers a high degree of expressiveness, which is apparently suitable to tackle the problem. Clearly, such a combined data model is expected to be simple enough to be accepted as a viable tool. To this purpose, more experience has to be acquired by using the ADOOD model. Moreover, further research on certain aspects of the ADOOD model is needed, including: the higher-order mechanisms, needed to support generalized path expressions and schema discovery, the full exploitation of the query capabilities of $\mathcal{LDL}$++ (such as set-grouping and aggregates), and the optimization techniques needed to support efficient execution of updates and actions.

References

[1] S. Abiteboul, D. Quass, L.McHugh, J. Widom, and J.L. Wiener. The Lorel Query Language for Semistructured Data. Technical report, Department of Computer Science, Stanford University, 1996. Available at `ftp://db.stanford.edu//pub/papers/lorel96.ps`.

[2] Serge Abiteboul. Querying Semi-Structured Data. In *Proceedings of the International Conference on Database Theory (ICDT97)*, pages 1–17, 1997.

[3] Serge Abiteboul and Victor Vianu. Queries and Computation on the Web. In *Proceedings of the International Conference on Database Theory (ICDT97)*, pages 262–275, 1997.

[4] H. Aït Kaci and R. Nasr. LOGIN: A Logic Programming Language with Built-in Inheritance. *Journal of Logic Programming*, 3(3):185–215, 1986.

[5] A. Albano, R. Bergamini, G. Ghelli, and R. Orsini. An Object Data Model with Roles. In *Proceedings of the 19th International Conference on Very Large Data Bases*, 1993.

[6] N. Arni, K. Ong, S. Tsur, and C. Zaniolo. $\mathcal{LDL}$++: A Second Generation Deductive Databases Systems. Technical report, MCC Corporation, 1993.

[7] E. Bertino, G. Guerrini, and D. Montesi. Deductive Object Databases. In *Proocedings of ECOOP'95*, 1995.

[8] A. J. Bonner and M. Kifer. Transaction Logic Programming. Technical Report CSRI-270, Computer System Research Institute, University of Toronto, December 1993.

[9] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding Structure to Unstructured Data. In *Proceedings of the International Conference on Database Theory (ICDT97)*, pages 336–350, 1997.

[10] F. Giannotti, G. Manco, M. Nanni, and D. Pedreschi. Datalog++: A basis for Active Object-Oriented Databases. Technical report, Department of Computer Science university of Pisa, may 1997.

[11] F. Giannotti, D. Pedreschi, D. Saccà, and C. Zaniolo. Non-Determinism in Deductive Databases. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases (DOOD91)*, Lecture Notes in Computer Science, pages 129–146. Springer-Verlag, Berlin, 1991.

[12] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Programming. *Journal of ACM*, 42(4):741–843, July 1995.

[13] L.V.S. Lakshman, F. Sadri, and I.N. Symramanian. A Declarative Language for Querying and Restructuring the Web. In *Proceedings of the POst-ICDE IEEE Workshop on research Issues in Data Engineering (RIDE-NDS'96)*, February 1996.

[14] S. W. Loke, A. Davison, and L. Sterling. Lightweight deductive databases on the world-wide web. In *Proceedings of the 1st Workshop on "Logic Programming Tools for INTERNET Applications"*, September 1996.

[15] A.O. Mendelzon, G.A. Mihaila, and T. Milo. Querying the World Wide Web. In *Proceedings of the Conference on Parallel and Distributed Information Systems*, 1996. Available from `http://www.cs.toronto.edu/ georgem/WebSQL.html`.

[16] D. Montesi. *A Model for Updates and Transactions in Deductive Databases.* PhD thesis, Dipartimento di Informatica Università di Pisa, 1993.

[17] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange across Heterogeneous Information Sources. In *Proceedings of the International Conference on data Engineering*, pages 251–260, 1995.

[18] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying Semistructured Heterogeneous Information. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases (DOOD95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 319–344, 1995.

[19] J. Su. Dynamic Constraints and Object Migration. In *Proceedings of the 17th International Conference on Very Large Data Bases*, 1991.

[20] V. S. Subrahmanian and C. Zaniolo. Relating Stable Models and AI Planning Domains. In *Proceeding of the International Conference on Logic Programming*, 1995.

[21] R. Wieringa, W. de Jonge, and P. Spruit. Using Dynamic Classes and Role Classes to Model Object Migration. *Theory and Practice of Object Systems*, 1(1):173–196, 1995.

[22] C. Zaniolo. Object Identity and Inheritance in Deductive Databases - An Evolutionary Approach. In *Int. Conf. on Deductive and Object-Oriented Databases, DOOD'89*, 1989.

[23] C. Zaniolo. Active Database Rules with Transaction Conscious Stable Model Semantics. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases (DOOD95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 55–72, 1995.