# PICSEL and Xyleme: two illustrative information integration agents

Marie-Christine Rousset and Chantal Reynaud
University of Paris Sud - CNRS (L.R.I) & INRIA (Futurs)
LRI, Building 490, University of Paris-Sud, F-91405, Orsay Cedex, France
mcr@lri.fr, cr@lri.fr

**Abstract**

An information integration agent provides a uniform query interface to a collection of distributed and heterogeneous information sources, giving users or other agents the illusion that they interrogate a centralized and homogeneous information system. In this chapter we focus on integration information agents that follow a mediator approach. A mediator is based on the specification of a single *mediated schema* describing a domain of interest, and on a set of *source descriptions* expressing how the content of each source available to the system is related to the domain of interest. These source descriptions, also called *mappings* because they model the correspondence between the mediated schema and the schemas of the data sources, play a central role in the query answering process. We present two recent information integration agents, namely PICSEL and Xyleme, which are illustrative of two radically different choices concerning the expressivity of the mediated schema. [1]

## 1 Introduction

The explosion of the number of information sources that are available on-line has raised the need for building information integration agents. An information integration agent provides a uniform query interface to a collection of distributed and heterogeneous information sources, giving users the illusion that they interrogate a centralized and homogeneous information system. Given a query expressing the information looked for by a user, an information integration agent takes in charge, in the place of the user, the tasks of finding the relevant information sources, interacting with each of these sources to get the useful data, and combining them to compute the appropriate final answer.

In the recent years, considerable research has been done about information integration both for structured and semistructured data sources, and several information integration systems have been implemented (e.g., [28, 25, 18, 12, 2,

[1] This chapter is a survey and a summary of two publications [18, 13].

32, 33, 5]). Most of them follow a declarative approach based on the specification of a single *mediated schema* describing a domain of interest, and on a set of *source descriptions* expressing how the content of each source available to the system is related to the domain of interest. These source descriptions, also called *mappings* because they model the correspondence between the mediated schema and the schemas of the data sources, play a central role in the query answering process. The users queries are posed in terms of the mediated schema, and are translated by the data integration system using the source descriptions into a set of queries that can be executed against the data sources.

The general architecture of an information integration agent following a mediator approach is given in Figure 1.

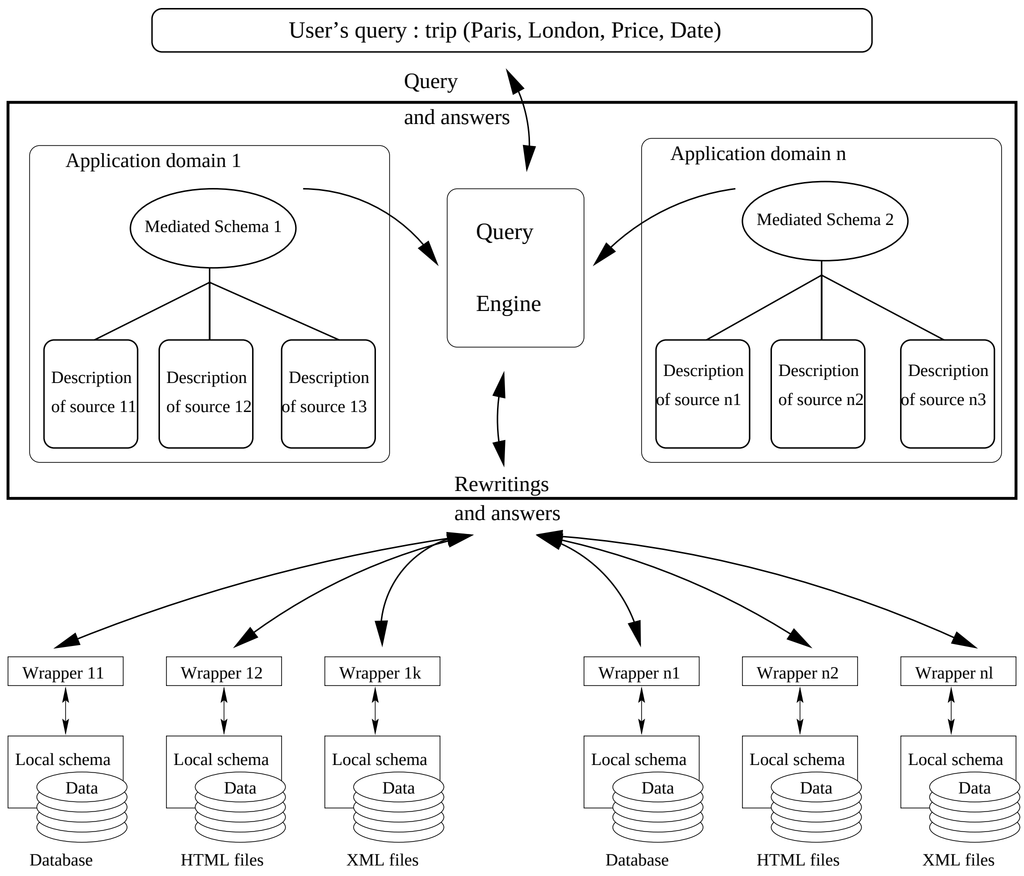


Figure 1: Architecture of a mediator-based information integration agent

The existing mediator-based information integration agents can be distinguished according to (1) the languages used for modeling the mediated schema and the source descriptions, (2) the type of mappings between the mediated schema and the schemas of the sources.

**Global As Views versus Local As Views:** Concerning the last point, information integration agents can be related to two main approaches for modelling inter-schemas correspondence: *Global As Views (GAV)* and *Local As Views (LAV)* . The *GAV* approach has been the first one to be proposed and comes from the Federated Databases world. The mediated schema is defined in function of the schemas of the sources to integrate, i.e., each relation of the mediated schema is defined as a view on the relations of the sources schemas. The advantage of this approach is the simplicity of query reformulation which simply consists of replacing each atom of the query by its definition in terms of the relations of the sources schemas. Its drawback is its lack of flexibility with respect to the addition or deletion of sources to the mediator: adding (or deleting) a source to the mediator may affect the definitions of all the relations of the mediated schema. The *LAV* approach is dual and has opposite advantages and drawbacks. It consists of describing the contents of the sources in function of the mediated schema. In such an approach, adding a new source is quite straighforward because each source is defined independently of each other. It simply requires to add some formulas to model the content of the new source as queries over the mediated schema. The price to pay for this flexiblity is the difficulty of the query answering processing. Since both the users queries and the views describing the sources contents are expressed in terms of the mediated schema, the reformulation of the users queries in terms of the source relations cannot be done as a simple query unfolding as for the *GAV* approach. This requires a more complex process of *rewriting queries using views* (see [19] and [8] for more details on the problem of answering queries using extensions of views).

**Relational versus object-oriented mediated schema:** The most representative information integration systems of the relational approach are: *Razor* ([15]), *Internet Softbot* ([14]), *Infomaster* ([16]) et *Information Manifold* ([25]). They all follow a *LAV* approach. The *Razor* and *Internet Softbot* systems use DATALOG (without recursion) for modeling the mediated schema, the views describing the sources contents and the users queries. *Infomaster* and *Information Manifold* are based on extensions of DATALOG. *Infomaster* exploits integrity constraints in addition of DATALOG rules. *Information Manifold* extends DATALOG by allowing that some predicates used in the rules are concepts defined by using description logics constructors. This hybrid formalism is in fact a precursor of the CARIN language considered in this chapter. The *HERMES* ([30]) system is a system of federated databases that can be seen as a mediator following a *GAV* approach and based on a Prolog-like query language with annotations for handling uncertainty and time.

The most representative information integration systems of the object-oriented approach are: *TSIMMIS* ([28]), *SIMS* ([4, 3]) *OBSERVER* ([27]) and *MOMIS* ([5]). *TSIMMIS* is based on the object-oriented language OEM for describing the mediated schema and the views, and on the OEM-QL query language. It follows a *GAV* approach. The *SIMS* and *OBSERVER* systems use a description logic for modeling the mediated schema, the views and the queries. They follow a *LAV* approach since the content of the source is described in function of the concepts described in the mediated schema. *SIMS* uses LOOM ([26]) while *OBSERVER* is based on CLASSIC ([7]). In those two systems, the problem of rewriting queries using views is handled as a planning problem and the completeness of the rewriting algorithms is not addressed. The *MOMIS* ([5]) system is based on the use of a very expressive description logic (ODL-I3) for describing the schemas of the sources to integrate. It follows a *GAV* approach since the mediated schema is inferred from the schemas of the sources.

Orthogonally to the *LAV* versus *GAV* and relational versus object-oriented distinctions, another criteria for distinguishing the information integration agents is the expressive power chosen for modeling the mediated schema. In this chapter, we present two recent information integration agents, namely PICSEL and Xyleme, which are illustrative of two radically different choices concerning the expressivity of the mediated schema.

In PICSEL[18], it has been chosen to offer a formalism combining the expressive power of rules and classes for designing a rich mediated schema, thus enabling a fine-grained description of the contents of the sources. Our aim was to build information integration agents related to application domains for which there exists many information sources containing closely related data. In such a setting, it is of primary importance to model the fine-grained differences between contents of sources if we want to be able to answer precise queries in an efficient way. PICSEL has been used for building a mediator in the tourism domain in collaboration with France Telecom R&D and the web travel agent Degriftour [2]. This travel agent makes available on-line three databases that contain different types of touristic products (flights, tours, stays in different places), each of them having its own specificities. For example, the *BonjourFrance* database offers a large variety of touristic products but all of them are located in France. The so-called *Degriftour* database offers flights, stays and tours for a lot of destinations all over the world. Its specificity however is that all its offers correspond to a departure date which is within the next two weeks. As a counterpart, the corresponding prices are specially interesting. The *Reductour* database provides rather similar products but with a less strong constraint on the departure date: it just has to be within the next eight months. Other differences exist between the contents of those three databases. For instance, we might know that *BonjourFrance* can provide rooms in Bed&Breakfast in addition to hotel rooms, while the only housing places that are proposed in the two other sources are hotels, located exclusively out of France for *Degriftour*, and located exclusively in Europe for *Reductour*.

[2]see http://www.degriftour.fr/

In Xyleme [32, 33, 1], the choice of a simple tree structure for mediated schemas has been guided by the goal of providing a very wide-area integration of XML sources that could scale up to the Web. The system architecture and the design choices have been motivated by "web search engine"-like performance requirements, i.e. supporting many simultaneous queries over a Web-scale XML repository. Xyleme is based on a simple data model with data trees and tree types, and a simple query language based on tree queries with boolean conditions. The main components of the data model are a mediated schema modeled by an abstract tree type, as a view of a set of tree types associated with actual data trees, called concrete tree types, and a mapping expressing the connection between the mediated schema and the concrete tree types. The simplicity of the mapping relation (correspondences between tree paths) eases automatic mapping generation and distributed storage. The query language is intended to enable end-users to express simple Query-by-Example tree queries over the mediated schema. Xyleme optimizes such tree queries to get very efficient response time.

This chapter is organized as follows. In section 2, we describe the way the expressive power of the CARIN [21] language is exploited in the PICSEL information integration agent, while maintaining the decidability of query rewriting using views and the tractability of query answering. In section 3, after describing the tree-based data model of Xyleme, we present a method for semi-automatically generating the mapping relation. For setting up semantic integration in Xyleme, the main difficulty comes from the very large number of data sources handled by Xyleme. As a consequence, the number of concrete data trees that have to be mapped to the abstract tree type makes impossible that the mapping relation is manually defined by some database administrator. We have therefore studied how to generate the mapping relation as automatically as possible, by combining syntactic, semantic and structural criteria. A prototype has been implemented and evaluated in the cultural domain. Finally, in Section 4, we draw some conclusions and we outline directions of future work about information integration in the setting of the Semantic Web.

## 2 PICSEL

In PICSEL, CARIN is used to represent both the domain of application and the contents of information sources that are available and relevant to that domain.

### 2.1 Syntax and semantics of CARIN

A CARIN knowledge base (KB) contains two components: a set $\mathcal{R}$ of rules, and a set $\mathcal{T}$ of Description Logics statements. Description logics statements are definition or inclusion statements about concepts and roles in the domain. Concepts and roles are unary and binary relations that can also appear in the antecedents of the rules. Relations that do not appear in the description logics statements are called *ordinary relations*. Ordinary relations can be of any arity.

**Description Logics component in CARIN:** The DL component of a CARIN knowledge base in PICSEL contains *concept definitions* and some kinds of *concept inclusions*, using the $\mathcal{ALN}$ DL. $\mathcal{ALN}$ contains the DL constructors of conjunction ($C_1 \sqcap C_2$), value restriction ($\forall R.C$), number restrictions (($\geq n\, R$), ($\leq n\, R$)), and negation ($\neg A$) restricted to basic concepts only.

- Concept definitions are of the form $CN := ConceptExpression$, where $CN$ is a concept name and $ConceptExpression$ is a concept expression. We assume that a concept name appears in the left hand side of at most one concept definition. *Atomic concepts* are those which do not appear in any left hand side of a definition. A concept name $CN$ depends on a concept name $CN'$ if $CN'$ appears in the definition of $CN$. A set of concept definitions is said *acyclic* if there is no cycle in the concept names dependency relation. In the setting of PICSEL, we consider only acyclic concept definitions. We can *unfold* an acyclic set of definitions by iteratively substituting every concept name with its definition.

- The concept inclusions that are allowed in the PICSEL setting are of the form:
  - $A \sqsubseteq ConceptExpression$, where $A$ is a atomic concept,
  - or $A_1 \sqcap A_2 \sqsubseteq \perp$, where $A_1$ and $A_2$ are atomic concepts.

**Rule component in CARIN:** The rule component $\mathcal{R}$ of a CARIN knowledge base contains a set of rules that are logical sentences of the form:

$$\forall \bar{X}[p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow q(\bar{Y})]$$

where $\bar{X}_1, \ldots, \bar{X}_n, \bar{Y}$ are tuples of variables (included in $\bar{X}$) or constants. We require that the rules are safe, i.e., a variable that appears in $\bar{Y}$ must also appear in $\bar{X}_1 \cup \ldots \cup \bar{X}_n$. As a shortcut, in the following, the variable quantification will be omitted. The relations $p_1, \ldots, p_n$ may be either concept names or expressions, role names, or ordinary relations that do not appear in $\mathcal{T}$. The relation $q$ must be an ordinary relation.

The *base* relations are those which do not appear in any consequent of rules. In particular, any concept or role appearing in the rules are base relations. We call a base atom an atom $p(\bar{X})$ where $p$ is a base relation. We call *concept-atom* an atom $p(X)$ where $p$ is a concept name or expression, and *role-atom* an atom $r(X, Y)$ where $r$ is a role name.

An ordinary relation $p$ is said to *depend* on an ordinary relation $q$ if $q$ appears in the antecedent of a Horn rule whose consequent is $p$. A set of rules is said to be *recursive* if there is a cycle in the dependency relation among ordinary relations. In the setting of PICSEL, we consider non recursive rules.

Since the rules are safe, without loss of generality, we can assume that in every rule, we have the disequality $X \neq Y$ for every pair of distinct variables appearing in the rule. For clarity, we omit these atoms in our examples and algorithms.

In addition, we allow *constraints* of the form:

$$p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow \perp.$$

**Semantics of** C**ARIN KBs:** The semantics of CARIN KBs is the standard model-based semantics of first-order logic. An interpretation $I$ contains a non-empty domain $\mathcal{O}^I$. It assigns to every constant $a$ an object $\alpha^I(a) \in \mathcal{O}^I$, and a relation of arity $n$ over the domain $\mathcal{O}^I$ to every relation of arity $n$. In particular, it assigns a unary relation $C^I$ to every concept in $\mathcal{T}$, and a binary relation $R^I$ over $\mathcal{O}^I \times \mathcal{O}^I$ to every role $R$ in $\mathcal{T}$. The extensions of concept and role descriptions are given by the following equations: ($\sharp\{S\}$ denotes the cardinality of a set $S$):

$(C \sqcap D)^I = C^I \cap D^I$,
$(\neg A)^I = \mathcal{O}^I \setminus A^I$,
$(\forall R.C)^I = \{d \in \mathcal{O}^I \mid \forall e : (d,e) \in R^I \rightarrow e \in C^I\}$
$(\geq n\, R)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \geq n\}$
$(\leq n\, R)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \leq n\}$

An interpretation $I$ is a model of a knowledge base $(\mathcal{T}, \mathcal{R})$ if it is a model of each of its components $\mathcal{T}$ and $\mathcal{R}$ . An interpretation $I$ is a model of $\mathcal{T}$ if $A^I \subseteq D^I$ for every inclusion $A \sqsubseteq D$ in $\mathcal{T}$, $CN^I = D^I$ for every concept definition $CN := D$. If $CE$ and $DE$ are two concept expressions, we say that *$CE$ is subsumed by $DE$* (modulo $\mathcal{T}$ ), denoted $CE \preceq DE$ ($CE \preceq_{\mathcal{T}} DE$). if $CE^I \subseteq DE^I$ in every interpretation $I$ (model of $\mathcal{T}$ ). A concept expression $CE$ is satisfiable (modulo $\mathcal{T}$ ) iff there exists an interpretation $I$ (model of $\mathcal{T}$ ) such that $CE^I$ is not empty.

An interpretation $I$ is a model of a rule $r : p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow q(\bar{Y})$ if, whenever $\alpha$ is a mapping from the variables [3] of $r$ to the domain $\mathcal{O}^I$, such that $\alpha(\bar{X}_i) \in p_i^I$ for every atom of the antecedent of $r$, then $\alpha(\bar{Y}) \in q^I$.

An interpretation $I$ is a model of a constraint $c : p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow \perp$ if there does not exist a mapping $\alpha$ from the variables of $c$ to the domain $\mathcal{O}^I$ such that $\alpha(\bar{X}_i) \in p_i^I$ for every atom of the antecedent of $c$.

## 2.2 Modelling the mediated schema

Given a basic vocabulary denoting names of base relations that are meaningful for the application domain (e.g., tourism), CARIN is used for defining new relations that are significant for the tourism domain and that can be defined over the base relations. In the setting of CARIN, there are two ways of defining complex relations: by rules, or by concept expressions. It is illustrated by the following example.

[3] Distinct variables are mapped to distinct elements

**Example 1** Let us suppose that our basic vocabulary contains atomic concepts such as $HousingPlace$, $Breakfast$, $CollectiveBuilding$, $PrivateBuilding$, which respectively denote the set of housing places and varied sets of services or buildings, and roles such as $AssRoom$, $AssMeal$ and $AssBuilding$, which denote binary relations between housing places and their associated buildings, room and meal services. We can define the two new concepts $Hotel$ and $Bed\&Breakfast$ by the following DL definitions:

$$Hotel := HousingPlace \sqcap (\geq 5\, AssRoom) \sqcap (\forall AssBuilding.CollectiveBuilding)$$

$$\begin{aligned} Bed\&Breakfast := {} & HousingPlace \sqcap (\geq 1\, AssRoom) \sqcap (\geq 1\, AssMeal) \\ & \sqcap (\forall AssMeal.Breakfast) \sqcap (\forall AssBuilding.PrivateBuilding) \end{aligned}$$

Rules can be used to define new n-ary relations. For instance, the following rule defines the notion of flights combined with stays as a 4-ary relation $FlightStay$. A stay combined with a flight is characterized by a departure city (denoted by the first variable $Dcity$ in the consequent of the rule), an arrival city (denoted by the variable $Acity$), a departure date ($Ddate$), a return date ($Rdate$). The possible combinations of a flight to and back a given destination with a stay at that place obey some constraints that are expressed by the conditions in the antecedent of the rule.

$$\begin{aligned} & Stay(S) \wedge Flight(V) \wedge Assoc(S,H) \wedge Located(H, ACity) \\ & \wedge ArrivalCity(V, Acity) \wedge DepartureCity(V, Dcity) \wedge DepartureDate(V, Ddate) \\ & \wedge ReturnDate(V, Rdate) \wedge BeginningDate(S, Ddate) \wedge EndDate(S, Rdate) \\ & \quad \Rightarrow FlightStay(Dcity, Acity, Ddate, Rdate) \end{aligned}$$

### 2.3 Modelling the contents of the information sources

Our approach for describing the information sources has been guided by the necessity of trading off expressive power against decidadibility of query answering. More precisely, each information source $\mathcal{S}$ is characterized by a set of source relations $\mathcal{V}_S$ (also called views), and described by a CARIN knowledge base which contains:

1. a set $\mathcal{I}_S$ of rules $v(\bar{X}) \Rightarrow p(\bar{X})$ that indicates which kind of data can be found in the source $\mathcal{S}$. The $p$'s are domain relations and there are as many source relations $v$'s in $\mathcal{V}_S$ (and as many implications in $\mathcal{I}_S$) as domain relations whose instances can be found in the source $\mathcal{S}$,

2. a set $\mathcal{C}_S$ of constraints on the instances of the source relations. We allow two types of constraints. *Terminological constraints* are of the form $v \sqsubseteq C$, where $C$ is a concept expression. *Integrity constraints* are of the form $l_1(\bar{X}_1) \wedge \ldots \wedge l_n(\bar{X}_n) \Rightarrow \bot$ where the $l_i$'s are source relations or negation of source relations, such that there is atmost one negation in each constraint.

A *source atom* (also called *view atom*) is an atom of the form $v(\bar{X})$ where $v$ is a source relation.

We encapsulate the terminological constraints within the *definition of views* associated with such constraints.

**Definition 1 (View definition)** *Let $v$ be a view in $\mathcal{V}$ appearing in the description $\mathcal{I}_\mathcal{S} \cup \mathcal{C}_\mathcal{S}$ of a source $\mathcal{S}$, and such that $v(X) \Rightarrow p(X) \in \mathcal{I}_\mathcal{S}$. The* definition of the view $v$, *denoted $def(v)$, is:*

- *$def(v) = p \sqcap C$ if $v \sqsubseteq C \in \mathcal{C}_\mathcal{S}$,*
- *$def(v) = p$ if $v$ is not associated to any terminological constraint.*

As an example, we can have the following (partial, for this illustration) descriptions of the three information sources that we have previously mentioned (i.e., $BonjourFrance$, $Degriftour$ and $Reductour$).

**Example 2** The rules in the description of the three sources say that we can find instances of housing places and flights in all of them (together with instances of associated properties like their location, their departure cities and dates ...). The constraints contained in each description enable distinguishing between them: for instance, the housing places and flights that can be found in $BonjourFrance$ are restricted to be located in France; the housing places that can be found in $Reductour$ are necessarily hotels. Some constraints serve to express that instances of some binary relations that can be found in a given source are exclusively related to some unary relations that can be found in the same source: for instance, the constraint $v^2_{BF}(X,Y) \wedge \neg v^1_{BF}(X) \Rightarrow \bot$ in $Descr(BonjourFrance)$ expresses that the locations that can be found in the source $BonjourFrance$ (denoted by the source relation $v^2_{BF}$ and the implication $v^2_{BF}(X,Y) \Rightarrow Located(X,Y)$) are only locations that are related to the housing places that can be found in the same source (denoted by the source relation $v^1_{BF}$ and the implication $v^1_{BF}(X) \Rightarrow HousingPlace(X)$).

| $Descr(BonjourFrance) : v^1_{BF}, \ldots, v^n_{BF}$ | |
|---|---|
| mappings: $\mathcal{I}_{BonjourFrance}$ | constraints: $\mathcal{C}_{BonjourFrance}$ |
| $v^1_{BF}(X) \Rightarrow HousingPlace(X)$<br>$v^2_{BF}(X,Y) \Rightarrow Located(X,Y)$<br>$v^3_{BF}(X) \Rightarrow Flight(X)$<br>$v^4_{BF}(X,Y) \Rightarrow ArrivalCity(X,Y)$<br>... | $v^1_{BF} \sqsubseteq (\forall Located.France)$<br>$v^2_{BF}(X,Y) \wedge \neg v^1_{BF}(X) \Rightarrow \bot$<br>$v^3_{BF} \sqsubseteq (\forall ArrivalCity.France)$<br>$v^4_{BF}(X,Y) \wedge \neg v^3_{BF}(X) \Rightarrow \bot$<br>... |

| $Descr(Degriftour) : v^1_D, \ldots, v^n_D$ | |
|---|---|
| mappings: $\mathcal{I}_D$ | constraints: $\mathcal{C}_D$ |
| $v^1_D(X) \Rightarrow Hotel(X)$<br>$v^2_D(X,Y) \Rightarrow Located(X,Y)$<br>$v^3_D(X) \Rightarrow Flight(X)$<br>$v^4_D(X,Y) \Rightarrow ArrivalCity(X,Y)$<br>... | $v^1_D \sqsubseteq (\forall Located.\neg France)$<br><br>$v^3_D \sqsubseteq (\forall DepartureDate.NextTwoWeeks)$<br>$v^4_D(X,Y) \wedge \neg v^3_D(X) \Rightarrow \bot$<br>... |

| $Descr(Reductour) : v_R^1, \ldots, v_R^n$ | |
|---|---|
| mappings: $\mathcal{I}_R$ | constraints: $\mathcal{C}_R$ |
| $v_R^1(X) \Rightarrow Hotel(X)$ | $v_R^1 \sqsubseteq (\forall Located.Europe)$ |
| $v_R^2(X,Y) \Rightarrow Located(X,Y)$ | |
| $v_R^3(X) \Rightarrow Flight(X)$ | $v_R^3 \sqsubseteq (\forall DepartureDate.NextEightMonths)$ |
| $v_R^4(X,Y) \Rightarrow ArrivalCity(X,Y)$ | $v_R^4(X,Y) \wedge \neg v_R^3(X) \Rightarrow \perp$ |
| ... | ... |

Here are some view definitions that we get from this modelling:
$def(v_{BF}^1) = HousingPlace \sqcap \forall Located.France$
$def(v_D^1) = Hotel \sqcap \forall Located.\neg France$
$def(v_R^1) = Hotel \sqcap \forall Located.Europe.$

### 2.4 Query processing in PICSEL

The queries that we consider are conjunctive queries of the form $Q(\bar{X}) : p_1(\bar{X}_1, \bar{Y}_1) \wedge \ldots \wedge p_m(\bar{X}_m, \bar{Y}_m)$, where the $p_j$'s are domain relations, some of which may be concept expressions or role names. The variables of $\bar{X} = \bar{X}_1 \cup \ldots \cup \bar{X}_m$ are called the *distinguished* variables of the query: they represent the variables of the query which the user is interested in knowing the instances when he asks the query. The variables that are not distinguished (denoted by $\bar{Y} = \bar{Y}_1 \cup \ldots \cup \bar{Y}_m$) are called *existential* variables of the query: they are existentially quantified and their use is to constrain the distinguished variables. For example, by the query $Q(X) : Hotel(X) \wedge Located(X,Y) \wedge France(Y)$, the user specifies that the answers he is interested in, are all the possible instances of $X$ that are hotels, and for which there exists instances of $Y$ (their locations) that are in France.

Answering queries in PICSEL resorts to find conjunctions of source atoms (called *rewritings*) which, together with the mediated schema and the descriptions of the sources, entail the initial query.

**Definition 2 (Rewriting)** *Let $Q(\bar{X}) : p_1(\bar{X}_1, \bar{Y}_1) \wedge \ldots \wedge p_m(\bar{X}_m, \bar{Y}_m)$ be a query expressed in term of domain relations, let $Q_{\mathcal{V}}(\bar{X}, \bar{Z})$ be a conjunction of view atoms. $Q_{\mathcal{V}}(\bar{X}, \bar{Z})$ is a rewriting of $Q(\bar{X})$ iff :*
*- $\{Q_{\mathcal{V}}(\bar{X}, \bar{Z})\} \cup \mathcal{R} \cup \mathcal{T} \cup \mathcal{I}_{\mathcal{V}} \cup \mathcal{C}_{\mathcal{V}}$ is satisfiable, and*
*- $\{Q_{\mathcal{V}}(\bar{X}, \bar{Z})\} \cup \mathcal{R} \cup \mathcal{T} \cup \mathcal{I}_{\mathcal{V}} \cup \mathcal{C}_{\mathcal{V}} \models \exists \bar{Y}\, p_1(\bar{a}_1, \bar{Y}_1) \wedge \ldots \wedge p_m(\bar{a}_m, \bar{Y}_m)$*

**Example 3** Coming back to Example 2, let us consider again the query:
$Q(X) : Hotel(X) \wedge Located(X,Y) \wedge France(Y)$.

The query $Q_{\mathcal{V}}(X) : v_R^1(X) \wedge v_{BF}^1(X) \wedge v_R^2(X,Y)$ is a rewriting of $Q(X)$ because, according to the definitions of $v_R^1$ and of $v_{BF}^1$, $v_R^1(X)$ entails $(Hotel \sqcap \forall Located.Europe)(X)$ and $v_{BF}^1(X)$ entails $(HousingPlace \sqcap \forall Located.France)(X)$. Then, $v_R^1(X) \wedge v_{BF}^1(X)$ entails in particular $Hotel(X) \wedge (\forall Located.France)(X)$. Therefore, according to the semantics of the $\forall$ constructor, $v_R^1(X) \wedge v_{BF}^1(X) \wedge v_R^2(X,Y)$ entails that there exists $Y$ such that $Hotel(X) \wedge Located(X,Y) \wedge France(Y)$ is true, and thus $Q_{\mathcal{V}}$ is a rewriting of $Q$.

Note that this rewriting shows that answering the query $Q$ using the available sources requires to integrate the information of the two sources *Reductour* and

*BonjourFrance*: hotels and their location are found in the *Reductour* source, but we have to intersect those data with housing places stored in the *BonjourFrance* source to be sure to obtain hotels that are located in France.

On the other hand, $Q'_{\mathcal{V}}(X) : v^1_D(X) \wedge v^1_{BF}(X) \wedge v^2_D(X,Y)$ is not a rewriting because it is insatisfiable. It is due to the fact that from the definitions of $v^1_D$ (i.e., $Hotel \sqcap \forall Located.\neg France$) and $v^1_{BF}$ (i.e., $HousingPlace \sqcap \forall Located.France$), it can be entailed that the location of $X$ is on the one hand necessarily out of France, and on the other hand in France. Therefore, $v^1_D(X) \wedge v^1_{BF}(X) \wedge v^2_D(X,Y)$ entails the two contradictory facts $France(Y)$ and $\neg France(Y)$, and thus $Q'_{\mathcal{V}}$ is insatisfiable.

A rewriting of a query $Q(\bar{X})$ can then be viewed as a specialized query that can be directly executed on the sources, and which provides relevant answers to the initial query.

Query processing in PICSEL can be summarized as follows.

1. **Normalization of the query.** We exhaustively apply to the body of the query the normalization rule: $C(o) \wedge D(o) \rightarrow (C \sqcap D)(o)$, where $C(o)$ and $D(o)$ are concept-atoms. Then, the resulting concept expressions are themselves normalized. Every concept expression is put in a normal form of a conjunction of concept expressions of the (*simple*) form: $A$ (atomic concept), $\neg A$, $(\geq\ n\, R)$, $(\leq\ n\, R)$, or of the (*complex*) form: $\forall R_1 \forall R_2 \ldots \forall R_k.D$, where D is simple.

2. **Verification of the satisfiability of the query.** This consists of (a) checking the satisfiability modulo the terminology of each concept expression appearing in the query, and (b) applying a forward-chaining algorithm to the rules until $\perp$ is obtained or no new atom can be obtained. If the query is insatisfiable, the query processing stops and the query is transmitted to a module of query relaxation described in [6].

3. **Query unfolding.** This is an iterative process based on successive steps of unfolding using the rules. Let $p(\bar{X})$ be an ordinary atom. Let $r$ : $p_1(\bar{X}'_1, \bar{Y}'_1) \wedge \ldots \wedge p_k(\bar{X}'_k, \bar{Y}'_k) \Rightarrow p(\bar{X}')$ be a rule of $\mathcal{R}$. Let $\alpha$ be the most general unifier of $p(\bar{X})$ and $p(\bar{X}')$, extended such that every variable $Y'_i$ is assigned to a fresh variable that appears nowhere else. An unfolding of the query using the rule $r$ is a conjunctive query obtained by replacing in the query the atom $p(\bar{X})$ by the conjunction: $p_1(\alpha(\bar{X}'_1), \alpha(\bar{Y}'_1)) \wedge \ldots \wedge p_k(\alpha(\bar{X}'_k), \alpha(\bar{Y}'_k))$.

   The process is iterated on each satisfiable unfolding until getting conjunctive queries in which all the ordinary atoms are base predicates. Since the rules are non recursive, the process terminates.

4. **Computation of terminological approximations.** Depending on the variable binding within the body of each conjunctive query, it is possible that conjunctions of concept-atoms and role-atoms are entailed by a single

concept-atom. It is important to determine such situations when they exist because they may lead to rewritings that could not be found otherwise.

In [17], the conjunctions of concept-atoms and atom-roles that can be entailed by a single concept-atom are characterized by the structure of the binding graph of their variables. The binding graph of a conjunction of unary and binary atoms is defined as follows: its nodes are the variables appearing in the conjunction ; there is an edge from $U_1$ to $U_2$ iff there exists a binary atom $r(U_1, U_2)$ in the conjunction. In particular, it is shown that if a conjunction $cj(X, \bar{Y})$ has a binding graph which is a tree rooted in $X$, we can build its $\mathcal{ALN}$ *concept approximation* which is the most general $\mathcal{ALN}$ concept description $C$ such that $C(X) \models \exists \bar{Y}\, cj(X, \bar{Y})$.

**Example 4** Consider again the query $Q(X) : Hotel(X) \wedge Located(X, Y) \wedge France(Y)$. Its body has a binding graph which is a tree rooted in $X$. Its concept approximation is: $Hotel \sqcap (\geq 1\, Located) \sqcap (\forall Located.France)$. The query $Q$ can then itself be approximated by the atomic query $Q'$ obtained by replacing the body of $Q$ by:
$(Hotel \sqcap (\geq 1\, Located) \sqcap (\forall Located.France))(X)$.
$Q'(X) : (Hotel \sqcap (\geq 1\, Located) \sqcap (\forall Located.France))(X)$. is called a terminological approximation of $Q$.

Fo every satisfiable conjunctive query resulting from rule unfolding, we build their *terminological approximations.* They are obtained by replacing a conjunction $cj(X, \bar{Y})$ of concept-atoms and role-atoms whose binding graph is a tree rooted in $X$ by the single-concept atom $C(X)$ where $C$ is the concept approximation of $cj(X, \bar{Y})$.

For every resulting conjunctive query, we then proceed to the rewriting of each atom of its body separately, and we conjunct the results to get the final rewritings.

5. **Rewriting of each query, atom by atom.**

Rewriting an ordinary atom or of a role-atom is trivial: the rewritings of a role-atom $\mathsf{r}(U_1, U_2)$ (or of an ordinary atom $\mathsf{p}(\bar{U})$) are the view-atoms $v(U_1, U_2)$ (ou $v(\bar{U})$) such that $v \in \mathcal{V}$ and $def(v) = \mathsf{r}$ (or $def(v) = \mathsf{p}$).

Rewriting concept-atom is much more complex. It is done by iterating the application of rewriting rules according to a strategy which guarantees termination of the rewriting process. Those rewriting rules are founded the following rules of entailment.

- Rewriting of a concept-atom of the form $(\sqcap_{i=1}^{n} \mathsf{C}_i)(U_0)$ : the rewritings of this type of atom are obtained by conjuncting the rewritings of the concept-atoms $\mathsf{C}_i(U_0)$ .

$$(1)\ \bigwedge_{i=1}^{n} \mathsf{C}_i(U_0) \models (\bigsqcap_{i=1}^{n} \mathsf{C}_i)(U_0).$$

- Rewriting of a concept-atom of the form $\mathsf{C}(U_0)$ (where $\mathsf{C}$ is simple i.e., not a conjunction) is based on the following entailment rule where $\mathsf{D} \preceq_{\mathcal{T}} \mathsf{C}$ :

$$(2)\ (\forall \mathsf{r}_1 \ \cdots (\forall \mathsf{r}_n\ \mathsf{D}))(U_n) \wedge \bigwedge_{i=1}^{n} \mathsf{r}_i(U_{n+1-i}, U_{n-i}) \models \mathsf{C}(U_0),$$

  This is a termination case for the rewriting algorithm.

- Rewriting of a concept-atom of the form $(\geq n\ \mathsf{r})(U_0)$ is based on the following entailment rule:

$$(3)\ \bigwedge_{i=1}^{n} \mathsf{r}(U_0, U_i) \wedge \bigwedge_{i=1}^{n} \bigwedge_{\substack{j = i+1 \\ i \neq j}}^{n} U_i \neq U_j \models (\geq n\ \mathsf{r})(U_0).$$

  The rewriting process is iterated on the role-atoms of the left hand side of the rule.

- Rewriting of a concept-atom of the form $(\forall \mathsf{r}\ \mathsf{C})(U_0)$ is based on the following entailment rule:

$$(4)\ (\leq n\ \mathsf{r})(U_0) \wedge \bigwedge_{i=1}^{n} (\mathsf{r}(U_0, U_i) \wedge \mathsf{C}(U_i)) \wedge \bigwedge_{i=1}^{n} \bigwedge_{\substack{j = i+1 \\ i \neq j}}^{n} U_i \neq U_j \models (\forall \mathsf{r}\ \mathsf{C})(U_0).$$

  The rewriting process is iterated on the role-atoms and on the concept-atoms of the left hand side of the rule.

**Example 5** Consider the atomic query:

$Q'(X) : (Hotel \sqcap (\geq 1\, Located) \sqcap (\forall Located.France))(X)$
which is the terminological approximation of the conjunctive query:

$Q(X) : Hotel(X) \wedge Located(X, Y) \wedge France(Y)$.

Consider the views described in Example 2. Based on Rule (1), rewriting $(Hotel \sqcap (\geq 1\, Located) \sqcap (\forall Located.France))(X)$ results in the rewritings of $Hotel(X)$, $(\geq 1\, Located)(X)$ and $(\forall Located.France)(X)$.
Based on Rule (2), the rewriting of $Hotel(X)$ is terminal and we get:
$Rewritings(Hotel(X)) = \{v_D^1(X), v_R^1(X)\}$.
Based on Rule (3), we obtain the following rewritings of $(\geq 1\, Located)(X)$:
$Rewritings((\geq 1\, Located)(X)) = \{v_{BF}^2(X, Y), v_R^2(X, Y), v_D^2(X, Y)\}$.
Finally, Rule (2) is used for getting the rewriting of $(\forall Located.France)(X)$:
$Rewritings((\forall Located.France)(X)) = \{v_{BF}^1(X)\}$.

There are 6 ways of conjunctively combining the rewritings of the three atoms $Hotel(X)$, $(\geq 1\, Located)(X)$ and $(\forall Located.France)(X)$:

- $v_D^1(X) \wedge v_{BF}^2(X, Y) \wedge v_{BF}^1(X)$: not satisfiable.

- $v_D^1(X) \wedge v_R^2(X,Y) \wedge v_{BF}^1(X)$: not satisfiable.
- $v_D^1(X) \wedge v_D^2(X,Y) \wedge v_{BF}^1(X)$: not satisfiable.
- $v_R^1(X) \wedge v_{BF}^2(X,Y) \wedge v_{BF}^1(X)$: satisfiable.
- $v_R^1(X) \wedge v_R^2(X,Y) \wedge v_{BF}^1(X)$: not satisfiable.
- $v_R^1(X) \wedge v_D^2(X,Y) \wedge v_{BF}^1(X)$: satisfiable.

The rewritings of $Q'(X)$, and thus of $Q(X)$, are those which are satisfiable:

$Q_{\mathcal{V}}^1(X) : v_R^1(X) \wedge v_{BF}^2(X,Y) \wedge v_{BF}^1(X)$

$Q_{\mathcal{V}}^2(X) : v_R^1(X) \wedge v_D^2(X,Y) \wedge v_{BF}^1(X)$.

# 3 Xyleme

The Xyleme project deals with data integration when data sources are XML documents. All XML documents are stored in a repository. In this way, the system is efficient even when several data issued from distributed sources must be combined to answer queries. A semantic module plays the role of interface between end-users and XML documents which, by definition, come from several sources and are semantically heterogeneous. Figure 2 presents the overall architecture of the system.

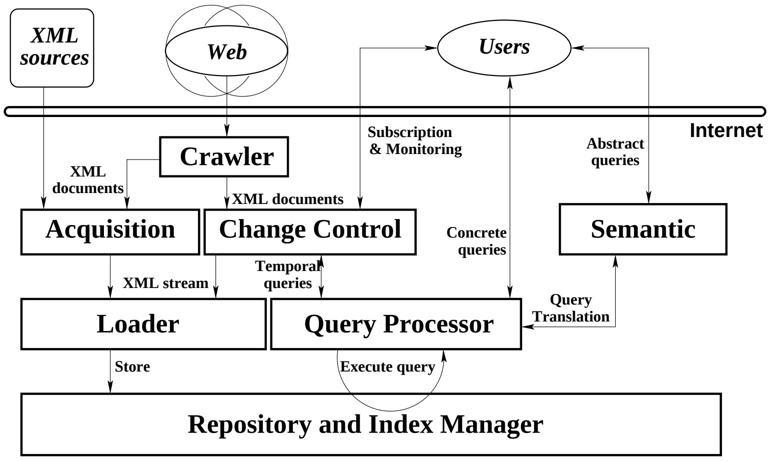


Figure 2: Xyleme architecture

- The Repository and Index Manager module is the lowest level in Xyleme. It enables to store and index XML documents.
- The Acquisition and Crawler module inspects the Web and collects all the XML documents, which are loaded in the repository by the Loader module.

- The Change Control module is responsible of specific functionalities, such as monitoring of document changes, version management and subscription of temporal queries.
- The Semantic module provides a homogeneous integrated and mediated schema on the heterogeneous XML documents stored in the repository.
- The Query Processor module enables to query the XML repository as a database. In particular, it translates a query in terms of the semantic layer into another one computable on the stored documents.

A general presentation of Xyleme is given in [33]. In this chapter we focus on the semantic module and the translation of abstract queries. A presentation of the other modules can be found in [20, 22, 24].

## 3.1 A uniform tree-based data model

The XML documents that are stored in the repository are instances of a DTD that defines their structure. From a data integration viewpoint, the data coming from differents sources are the XML documents, and the DTDs are the schemas of the data sources. For example, here is the partial definition of the DTD `MyDTD`:

```
<!ELEMENT MyDTD (Film*, Painting*)>
<!ELEMENT Film (Casting, Character*,Title)>
<!ELEMENT Casting (Actor*)>
<!ELEMENT Painting (Title, Author, Museum?)>
<!ELEMENT Actor (#PCDATA)>
............
```

In our model, we abstract XML documents as being *data trees*. Our modelling simplifies real XML documents in several ways. For example, the model does not distinguish between attributes and elements. We consider unordered trees. We use also a simplified version of DTDs, which we call *tree type*, where the multiplicity of an element is not considered.

Data trees and tree types are labelled unordered trees, where the labels correspond to the names of the elements and the attributes of the XML documents.

A data tree $d$ is an instance of a tree type $t$ if there exists an homomorphism from $d$ onto $t$. For example, Figure 3 gives an example of a data tree. It is an instance of the tree type given in Figure 4.

The aim of the semantic module is twofold: (i) to provide an homogeneous mediated schema over the data trees, in order to allow a natural expression of queries for the user without having to be aware of actual tree types that can have heterogeneous structures and labels; (ii) to define the connection between this semantic module and the tree types describing the actual data trees, in order to make possible query processing.

The semantic module provides a simple description of domains (e.g. culture, tourism) in the form of an *abstract tree type*. An abstract tree type can be

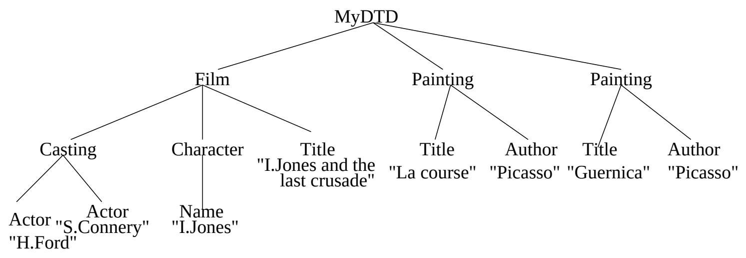


Figure 3: A data tree

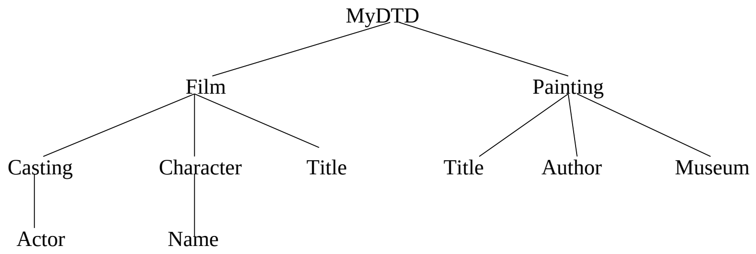


Figure 4: Example of a concrete tree type

considered as an abstract merger of the DTDs associated with the set of XML documents of some domain. Figure 5 shows a fragment of an abstract tree type in the cultural domain. The label of the root is the domain name (`Culture`). The internal nodes are labelled with terms that represent either concepts of the domain (`Art, Architecture, Cinema`) or properties of those concepts (`Name, Title`).

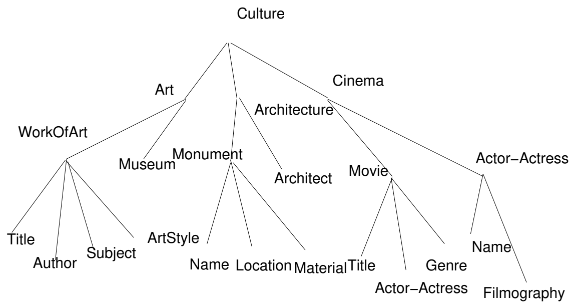


Figure 5: Fragment of an abstract tree type on culture

There are two main reasons for the choice of representing a mediated schema as a tree. First, it must be compatible with a real XML DTD, because it is intended to be the schema of all the XML documents of the domain. Next, the mediated schema must be simple enough, because it is the support of the visual query interface tool, intended to be used by non-programmer users. The links between a node and its sons in the abstract tree type have no strict semantics. They may correspond to a relation of specialisation between concepts, or to a relation of composition, or simply express different viewpoints on a concept. A link also exists between a concept and a property of it. We do not distinguish between those different types of links. In other words, a node labelled with a term $E_2$ being a son of a node labelled with $E_1$ just means that the term $E_2$ has to be interpreted in the scope of meaning of $E_1$. Thus, the different occurrences of a same term do not have the same meaning. This is obvious for terms that represent properties (e.g. `Name` can appear different times, and may represent the name of different entities), but it is also the case for terms representing concepts (e.g. `Author` under `Movie` means director, while `Author` under `Painting` means painter).

Such representations are very simple and structured. They enable end-users to build semantic queries from different viewpoints in an easy and natural way, using both the vocabulary and the structure of the abstract tree type. This provides a homogeneous and simple user interface for querying a very large amount of heterogeneous XML documents stored in the repository. Using this interface, end-users must be able to obtain relevant answers for queries.

For this, Xyleme has to identify the XML documents concerning each query issued against the semantic module. This requires the creation and maintenance of a correspondence between the labels of the abstract tree type and the elements of the concrete tree type that describe the structure of the XML documents. This way, each abstract query can be translated into a set of concrete queries over real documents, by simply replacing the abstract labels of the query with the corresponding concrete labels. In Xyleme, the correspondence is stated by a *mapping relation* between paths of the abstract tree type and paths of the tree types modeling the DTDs of the data stored in the repository.

**Example 6** Here are some examples of meaningful mapping elements between the abstract tree type given in Figure 5 and the tree type given in Figure 4:

```
Culture/Cinema/Movie <-> MyDTD/Film
Culture/Cinema/Movie/ActorActress <-> MyDTD/Film/Casting/Actor
.......................
```

Because of the number of concrete data trees that have to be mapped to the abstract tree type, the mapping relation cannot be manually defined by some database administrator. Some automatic help must be provided. In Section 3.3, we describe a semi-automatic method for generating the mapping relation that has been implemented in Xyleme and experimented. By accessing the abstract tree type through a user-friendly interface, end-users query a single tree structure summarizing many DTDs. The query language is also tree-based.

The query tree on the left in Figure 6 models a query in the cultural domain asking the titles of all the films in which the actor Sean Connery is acting. The query tree on the right in Figure 6 asks the description of all impressionist works of art. These examples illustrate the basic method for building abstract queries: the user marks in the abstract tree type the nodes to be included in the result (selected nodes are marked with a `S`, e.g. `Title, WorkOfArt`) and the nodes constrained with conditions (conditional nodes are marked with a `C`, e.g. `Actor-Actress, ArtStyle`). The query itself is given by a tree query pattern built from the abstract tree type.

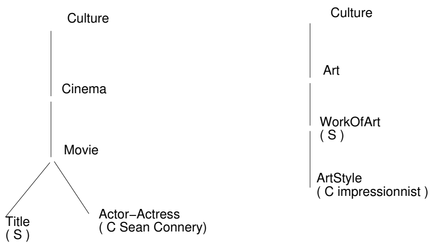


Figure 6: Example of user queries

The Xyleme query language is a subset of *XQuery*, the W3C query language for XML data [31]. Unlike XQuery, the Xyleme query language does not allow joins or document transformation. Moreover, the language ignores ordering and random access through an ordinal number to the descendants of a tree node, features that exist in *XPath* [10], which is a part of XQuery. On the other hand, the Xyleme query language is more powerful than XPath, because it enables to extract any part of a subtree, while XPath can only extract a full subtree, identified by its access path. Notice that even if the abstract tree queries do not contain joins, the translation into concrete tree queries may produce joins based on links between concrete documents (see [11]). Notice also that the last version of the Xyleme query language, not presented here, was enriched with additional features such as joins (on values and on links) and simple document transformation.

## 3.2 Query processing

The users queries are defined relatively to the mediated schema and not relatively to the actual tree types corresponding to the data stored in the repository. Therefore they cannot be directly evaluated against that data. The evaluation of an abstract query is then a two step process. First, the abstract tree query is translated into concrete tree queries by using the mapping relation. Second, each concrete tree query has to be evaluated on the database. The translation process is described in detail in [11, 13]. It relies on building trees consistent with the mapping relation. In this chapter, we focus on illustrating by example the semantics of the queries and the query translation process.

Consider the following database schema composed of an abstract tree type and a concrete tree type as given in Figure 7 with the following mapping relation:

```
Movie <-> Film          Movie/ActorActress <-> Film/Casting
Movie/ActorActress/Name <-> Film/Character
Movie/ActorActress/Name <-> Film/Casting/Actor
```

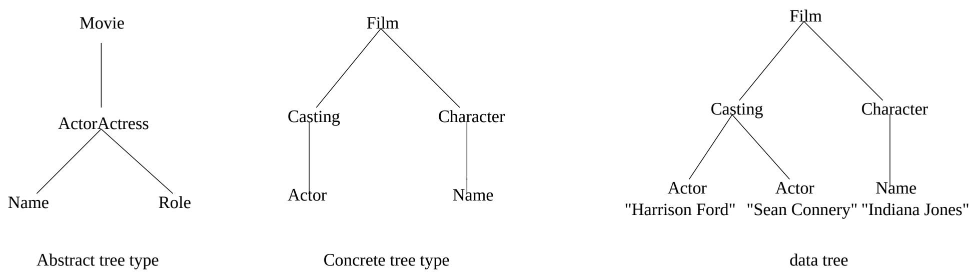


Figure 7: An abstract tree type, a concrete tree type and a data tree

```
Movie/ActorActress/Role <-> Film/Character/Name
```

An instance of the database is also given in Figure 7 with only one data tree.

Figure 8 gives three examples of abstract tree queries that are conform to the abstract tree type of Figure 7.

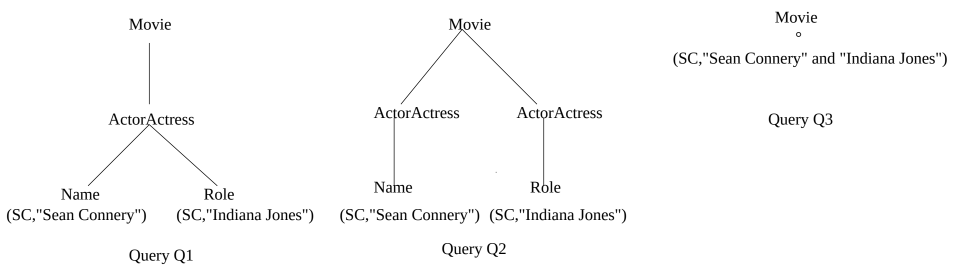


Figure 8: Example of tree queries

The interpretation of the first one (Query $Q_1$) is: find all the data trees rooted at a node labelled by `Movie`, such that there exists a subtree rooted in a node labelled by `ActorActress` which contains two branches, the one rooted in a node labelled by `Name` must contain the string “Sean Connery” and the one rooted in a node labelled by `Role` must contain the string “Indiana Jones”; for those data trees returns the identity of subtrees rooted in the nodes labelled by `Name` and `Role`.

The second one (Query $Q_2$) has a slightly different interpretation: find all the data trees rooted at a node labelled by `Movie` such that there exists one subtree rooted in a node labelled by `ActorActress` having a branch rooted in a node labelled by `Name` which contains the string ”Sean Connery” , and there exists another (possibly distinct) subtree rooted in a node labelled by `ActorActress` having a branch rooted in a node labelled by `Role` which contains the string ”Indiana Jones”.

The last tree query (Query $Q_3$) is an example where we have a non atomic boolean condition `"Sean Connery" and "Indiana Jones"`. This boolean condition expresses that the corresponding data subtree must contain the two strings `"Sean Connery"` and `"Indiana Jones"`. When talking about the interpretation of an abstract query, we must understand that these data trees exist only virtually. As explained in the database model, the real data trees are associated with concrete tree types.

For the translation process, some nodes in the tree query are crucial: they are nodes with conditions, nodes that are selected, or "join" nodes.

**Definition 3 (Necessary nodes in a tree query)** *A node in a tree query is* **necessary** *iff it is the root, a selected node, a conditional node or a node ( called a join node) which has, at least, two distinct descendants that are necessary nodes. For a tree query $Q$, we denote $N(Q)$ the set of its necessary nodes.*

The necessary nodes can be partially ordered by the relation $\prec$. Let $u$ and $v$ be two necessary nodes: $u \prec v$, iff $u$ is a descendant of $v$ and there is no necessary node $w$ such that $u \prec w \prec v$.

The translation of an abstract query $Q$ into a concrete query can be summarized as follows. The concrete query has exactly the same tree structure as the translated abstract query but the nodes of the abstract tree query must be mapped by a function $h$ onto the nodes of the concrete tree query such that the labelling of necessary nodes of the abstract tree query and the concrete tree query is conform to the mapping relation. More precisely: for two necessary nodes $u$ and $v$ in $N(Q)$ such that $u \prec v$, their images by the translation $h$ must satisfy that $path(h(v))$ is a prefix of $path(h(u))$ (denoted $path(h(v)) \lhd path(h(u))$).

An abstract query can have more than one translation, depending on the mapping relation. It may also be possible that an abstract query has no translation., as it is illustrated in the following example.

**Example 7** We are considering the abstract queries given in Figure 8 and the database composed of a unique data tree instance $d$ as presented in Figure 7. Each abstract path to any necessary node of the query must be converted into a concrete path using the mapping relation. For example, if we consider the abstract tree query $Q_1$, its necessary nodes, $u_1$, $u_2$, $u_3$, and $u_4$, respectively labelled by `Movie`, `ActorActress`, `Name` and `Role` , are such that: $u_3 \prec u_2$ and $u_4 \prec u_2$. The corresponding abstract paths leading to the necessary nodes are: `Movie`, `Movie/ActorActress`, `Movie/ActorActress/Name`, and `Movie/ActorActress/Role`. By the mapping relation given in Example **??**, they must be mapped to the concrete paths `Film`, `Film/Casting`, `Film/Casting/Actor` (or `Film/Character`) and `Film/Character/Name` respectively. This suggests a candidate translation based on an homomorphism $h$ such that: $h(u_1)$ is labelled by `Film`, $h(u_2)$ is labelled by `Casting`, $h(u_3)$ is labelled by `Actor` or by `Character`, and $h(u_4)$ is labelled by `Name`. However, in order that candidate translation to be a real translation, since $u_4 \prec u_2$, $path(h(u_2))$, i.e., `Film/Casting`, should be a prefix of $path(h(u_4))$, which must be `Film/Character/Name`. Since `Film/Casting` is not a prefix of `Film/Character/Name`, there is no translation, and thus no answer, for the query $Q_1$.

If we consider now the abstract query $Q_2$, its necessary nodes are $v_1$, $v_2$ and $v_3$, respectively labelled by `Movie`, `Name` and `Role`. The abstract paths leading to those necessary nodes are: `Movie`, `Movie/ActorActress/Name`, and `Movie/ActorActress/Role`. Based on the mapping relation, they can be converted respectively into `Film`, `Film/Casting/Actor` and `Film/Character/Name`. This leads to a translation $Q'_2$ corresponding to an homomorphism $h'$ such that $h'(v_1)$ is labelled by `Film`, $h'(v_2)$ is labelled by `Actor` and $h'(v_3)$ is labelled by `Name`. In this case, $h'$ satisfies the prefix property since: $v_2 \prec v_1$ and $path(h(v_1))$ (i.e., `Film`) is a prefix of $path(h(v_2))$ (i.e., `Film/Casting/Actor` ), and $v_3 \prec v_1$ and $path(h(v_1))$ (i.e., `Film`) is a prefix of $path(h(v_3))$ (i.e., `Film/Character/Name`). The resulting concrete query $Q'_2$ is given in Figure 9.

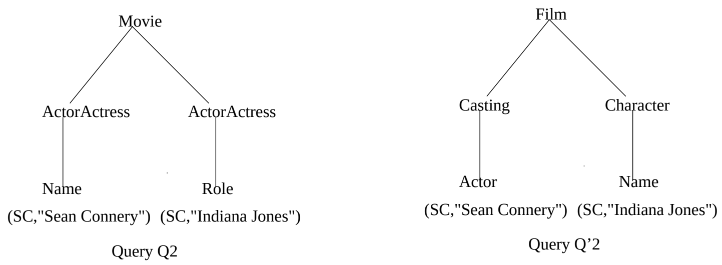


Figure 9: The abstract query $Q_2$ and its translation $Q'_2$

It can then be evaluated against the data tree $d$: there exists a valuation $\sigma_d$, where the node labelled with `Actor` can be associated with the leaf of $d$ labelled by `Actor` and valued by `"Sean Connery"` and the other node labelled by `Name` can be associated with the leaf of $d$ labelled by `Name` and valued by `"Indiana Jones"`.Therefore, the query $Q'_2$ has at least this answer.

For the abstract query $Q_3$ the pattern of the translation $Q'_3$ is reduced to one node labelled with `Film`. The whole data tree $d$ matches also the query $Q'_3$ because it is rooted at a node labelled by `Film` which contains the strings `"Sean Connery"` and `"Indiana Jones"` by application of the closure valuation principle.

## 3.3 Automatic Generation of Mappings

We have seen previously that the mapping relationship is a crucial element of the data model for translating abstract queries into concrete ones. In this section, we address the problem of automatically finding the elements of the mapping relation which states the correspondence between paths of concrete tree types and paths of the abstract tree type. We have to handle heterogeneous names and structures because concrete tree types come from various sources designed by different persons who made personal choices on the names of the labels and on the tree structures. Therefore, automatic mapping generation have to deal with semantic and structure heterogeneity.

| Semantic relationship | Linked terms |
|---|---|
| Synonymy | Movie - Film, Picture - Image<br>Each couple of nouns<br>belongs to the same synset |
| Hyponymy / Hypernymy | Painting - Portrait,<br>because portrait is a kind of painting.<br>Portrait is a hyponym of Painting and<br>Painting is a hypernym of Portrait. |
| Meronymy / Holonymy | Movie - Scene,<br>because a movie is composed of several scenes.<br>Scene is a meronym of Movie and<br>Movie is a holonym of Scene. |
| Hyponymy composed<br>with hypernymy | Painting - Dance,<br>because Art is a hypernym<br>of both Painting and Dance |

Figure 10: Semantic relationships used in the SAMAG prototype

Below, we present the method for automatic mapping generation in Xyleme, based on two kinds of criteria: syntactic/semantic and structural. We report the first results of an experiment with a prototype, SAMAG, used to generate mappings for XML documents in the cultural domain.

**Syntactic and Semantic Matching** Automatic mappings generation is based on term matching. The idea is that a mapping is generated only if the abstract and the concrete terms identified by the mapped paths are semantically related. We check two kinds of relations between terms: syntactic and semantic matching.

Syntactic matching concerns the syntactic inclusion of a term, or of a part of a term, into another one. For example, the concrete term `Actor` is syntactically similar to the abstract term `Actor-Actress`. This method also includes techniques for detecting abbreviations, such as `nb` for `Number`, etc. For more details, see [29].

Semantic matching is based on the use of extra knowledge available through existing ontologies or thesauri. In our experimentation, we chose the WordNet thesaurus [23] .

WordNet groups English nouns, verbs, adjectives and adverbs into sets of synonyms that are linked through semantic relations. Each set of synonyms (synset) represents a concept. A word may belong to several synsets, each one representing a particular sense. Our approach only exploits semantic links between nouns because we chose to use only abstract terms that are nouns. The relationships used in our prototype are listed and illustrated in Figure 10.

**Dealing with Structure** Syntactic / semantic matching between words is not enough to obtain precise mappings. The meaning of a term depends on the place it occupies in the concrete tree type, which defines the interpretation context of the word. More precisely, a term may occur several times in the same concrete tree type with different senses. Its meaning may be influenced by the meaning of its predecessors in the tree (e.g. `Name` may be the name of an artist, of a museum, etc.).

Our approach is based on a very simple hypothesis: the terms of a concrete tree type are either object names (e.g. `Painting`), or property names (e.g. `Name`). We consider that property names have a meaning only in the context of the object that they characterise. So, an object, which is characterised by a set of properties, defines the interpretation context of its properties.

In order to decide if a term is an object or a property node, we use heuristics based on the translation of a concrete tree type into a conceptual database schema. We use conceptual schemas built according to the Entity-Relationship formalism, that naturally helps to determine the classes, called entities, in a domain. An entity is a class of instances that share the same properties called attributes. In our approach, we consider that object nodes are similar to entities and that property nodes are similar to attributes. Given a link $A$ - $B$ in the concrete tree type, one heuristics used in the translation process is that $B$ may refer to an entity (according to the above definition) only if $B$ has son nodes, these ones being interpreted as $B$'s attributes. Such a heuristics leads to consider that leaves in a concrete tree type always are property nodes (they can't refer to entities, so they can't be object nodes). Other heuristics are used to qualify internal nodes in a concrete tree type, in order to decide if they are object or property nodes. The exhaustive translation mechanism is described in [29]. However, very often, internal nodes are object nodes.

The basic idea underlying our structural constraints is that a mapping of a property node must be "compatible" with the mapping of the object node it characterizes. In other words, because the property and the object nodes in the concrete tree type are related (the object defines the context of the property), the corresponding abstract terms must be related in the same context-based manner.

Thus, if a concrete path identifying a property name is mapped to an abstract path $X$, this mapping will be valid only if the object characterized by that property is mapped to a predecessor of $X$ in the abstract tree type. This introduces a context constraint into the mapping generation process, because a property name $P$ of an object $O$ may only be mapped to the subset of abstract terms placed below the abstract terms mapped to $O$, i.e. the context of the object is transmitted to the property. This also means that mappings for object names must be computed first.

For instance, in the concrete tree type MyDTD, the terms `Title`, `Author` and `Museum` are properties of the object `Painting`. The mappings:

```
Culture/Art/WorkOfArt/Title <-> MyDTD/Painting/Title
Culture/Art/WorkOfArt/Author <-> MyDTD/Painting/Author
```

are valid, because there exists a mapping between the object `MyDTD/Painting` and the abstract term `Culture/Art/WorkOfArt`.

Notice that this context-based constraint may seem too strong in some cases. For instance, the automatic generation algorithm will not find the mapping `Culture/Art/Museum <-> MyDTD/Painting/Museum` if there is no mapping from `MyDTD/Painting` to a prefix of `Culture/Art/Museum`. Imagine that the system does not semantically match `Painting` with `Art`, then the "obvious" mapping `Culture/Art/Museum <-> MyDTD/Painting/Museum` is missed. However, a more careful analysis of this case shows that the missed mapping is not so obvious, because it is not clear if `Culture/Art/Museum` is a museum with paintings.

The structural, context-based constraints for automatic mapping generation significantly improve precision if mappings are correctly found for object nodes. On the other hand, an error at an object node will propagate to all its property nodes.

**Experiment** A prototype, SAMAG (Semi-Automatic MApping Generator), has been implemented in Java. We report here the first results of this work.

The corpus of concrete tree types used in our experiment is related to the cultural domain. An abstract tree type on this domain has been written and mappings have been established manually between this abstract tree type and all the concrete tree types associated with real DTDs of the corpus. We used these manual mappings as a reference set for evaluating the results of the SAMAG system.

SAMAG is a semi-automatic tool. User-validation is necessary each time a syntactic relationship between an abstract term and a concrete term is detected. Indeed, two syntactically similar terms may refer to different concepts and may not have the same meaning. Only a human is actually able to guarantee the semantic consistency of such a mapping.

SAMAG has been implemented as a modular system that can be parameterised. This makes the evaluation of our method easier because results obtained with various parameters from the same inputs can be compared. SAMAG's parameters are:

1. The semantic relationships used by the system to detect a semantic link between abstract and concrete terms, such as synonymy, hypernymy, holonymy, etc.

2. The structural constraints to apply on terms of concrete tree types. One can decide to apply or not these constraints. Furthermore, they can be applied only on property nodes in order to consider that property names have a meaning only in the context of the object that they characterise. But they can also be applied on all the nodes of a concrete tree type considering that the meaning of object names is also influenced by the meaning of their predecessors in the tree.

Notice that in all the cases, the system tests the existence of a syntactic relationship between an abstract term and a concrete term.

The results of the automatic mappings generation process are analysed according to (1) the number of relevant mappings among those automatically generated, and (2) the number of mappings manually created that are retrieved by the system. In particular, we study the following categories of mappings:

1. Mappings that are both manually and automatically generated, such as `Culture/Art/Artist <-> School/Painters_list/Painter`[4]. Such a mapping is generated thanks to a synonymy relationship between `Painter` and `Artist`,

2. Mappings manually created that are not retrieved by the SAMAG system, such as
`Culture/Art/Artist/ArtMovement <-> School/Name`. Here, there is no semantic relationship in WordNet between `ArtMovement` and `Name`, and the detection of a similarity relationship between an abstract term and a concrete term is a precondition to generate a mapping,

3. Irrelevant mappings, that are automatically but not manually generated, such as
`Culture/Cinema/Movie/Picture <-> Painting/Image`. Here, there is a synonymy relationship between `Picture` and `Image`, but `Picture` is used in a cinema context and `Image` is used in a painting context. SAMAG is unable to detect such context differences, because `Image` is not a property node. Other irrelevant mappings may be generated because of the polysemy of terms (a word may have different meanings), not handled by SAMAG.

4. Mappings automatically generated, that may be considered relevant, even if they have not been found manually, such as `Culture/Art <-> Painting`.
The above discussion about the mapping
`Culture/Art/Museum <-> MyDTD/Painting/Museum`
demonstrates that several meanings are possible for a term in this context, and that sometimes the semantics is a matter of choice. We also considered as relevant mappings that "have a sense", such as `Culture/Art/Artist <-> School/Painters_list`, but that were considered as useless at manual mappings creation because there is a tag `Painter` below `School/Painters_list`.

The table in Figure 11 gathers some significant statistical results about the mappings generated by the SAMAG system.

**Results analysis** The first column in the table describes results that are obtained when all the semantic relationships in WordNet are used and no structural constraints are applied on terms of the concrete tree types. With this configuration, one obtains the maximum number of mappings that SAMAG can generate. The only condition to generate a mapping is the existence of a similarity relationship (semantic or syntactic) between abstract and concrete

[4]The abstract tree type in Figure 5 is just a part of the one used in this experiment.

| **Semantic relationships** | All | Synonymy | All |
|---|---|---|---|
| **Structural constraints** | None | All | Property nodes |
| **Generated mappings** | 683 | 32 | 267 |
| **Relevant mappings** | 145 | 28 | 115 |
| **Percentage of relevant mappings** | 21% | 87.5% | 43% |
| **Manual mappings retrieved** | 74 / 111 | 18 / 111 | 52 / 111 |

Figure 11: Behavior of the SAMAG prototype

terms. In this configuration a lot of mappings are not relevant because the system does not consider the interpretation context of terms.

The second column in the table describes the results when SAMAG is highly constrained. We consider that the meaning of all terms in the tree is influenced by the meaning of its predecessor. Moreover, only one semantic relationship, synonymy, is checked between terms. It is the relation that introduces the lowest distance between terms. In this configuration, almost all the generated mappings are relevant. However, few mappings are generated, so few manual mappings are retrieved.

The last column in the table seems a good trade-off between the number of relevant mappings and the number of mappings automatically retrieved by SAMAG among manual mappings. In this configuration, all the semantic relationships in WordNet are used and only property names are considered to have a meaning in the context of their predecessor in the tree.

The relatively high number of mappings not found by SAMAG (33%) is explained by the fact that many manually created mappings are based on corpus-specific knowledge, e.g. `Culture/Art/WorkOfArt/Artist <-> XML_DOC/Document/Title`, because a document `XML_DOC` is about a painter whose name is contained in the document's title `XML_DOC/Document/Title`. Manually created mappings take advantage of such knowledge, while SAMAG cannot. Other mappings are missed because the concrete terms are non-standard abbreviations instead of significant nouns. The last category of manual mappings missed by the system are mappings that connect an object to some identifying property, e.g. a director and his name, such as in `Culture/Cinema/Director/Name <-> List/Movie/Directed_by/Director`.

The major issue explored in this experiment is the way the interpretation context of terms involved in mappings should be considered in an automated process. We proposed a solution based on structural constraints and the analysis of our results must be viewed as a contribution to better understand how to take context into account.

First, we conclude that structural constraints are necessary to limit the number of generated mappings and to produce relevant ones. The number of irrelevant mappings is too high when no structural constraint is applied (145 relevant mappings among 683 when the configuration in column three gives 115 relevant mappings among 267).

The results also show that applying contextual or structural constraints only on property nodes is not enough restrictive. In the last column of the table in Figure 11, more than 50% of automatically generated mappings are not relevant. An analysis of the irrelevant mappings shows that most of terms involved in bad mappings are objects nodes. Thus, structural constraints must also be applied on object nodes. However, if we apply structural constraints on all the nodes (see results in the second column), the system becomes too constrained and does not generate enough mappings.

The question is then how to define the interpretation context for object nodes. In many cases, the context of an object node is not only given by its predecessor node within the associated path. For an object node, the number of nodes within the path that are significant to define its context of interpretation depends on it, so this number is variable from an object node to another. For instance, the context of interpretation of the object node `Actor` in `Culture/Cinema/Movie/Actor` is `Movie` whereas the context of interpretation of the object node `Museum` in `Culture/Painting/Oil/Museum` is `Painting, Oil`.

The results also depend on the extra knowledge used to establish semantic relationships between terms. WordNet was used as a resource to look for semantic relationships among terms. This choice has been principally motivated by the fact that data in WordNet are easily accessible for automatic applications. It offers a large coverage of general lexicon in English. However, because a term in natural language has often many meanings, WordNet returns synsets for all the senses of a given term. However, we don't know how to select automatically the right sense of a term, because it directly depends of its interpretation context. So when the system, for instance, asks to WordNet the set of synonyms of a given term, this one, being not domain-specific, takes synonyms of all the senses of the term.

An important remark concerns the influence of mapping generation on query translation. Of course, if a relevant mapping is missed by SAMAG, the query processor will miss the results concerning that mapping. However, additional irrelevant mappings do not necessarily produce irrelevant answers because query translation need all the mappings in the translation of the query tree to be compatible (i.e. to form a tree that respects the descendant relationship among terms). Yet, reducing the number of irrelevant mappings is important for storage saving reasons, which are critical for web-scale domains.

# 4 Conclusion and perspectives

PICSEL and Xyleme are two mediator-based information integration agents that have been designed in very different settings. As a result, different choices have been done for designing the mediated schema and the semantic mappings between the mediated schema and the schemas of the sources.

In PICSEL, a logical formalism combining the expressive powers of (non recursive) DATALOG rules and the $\mathcal{ALN}$ description logic has been used for modeling the semantic relationships between source schemas through a rich mediated schema. It has proved very useful for expressing fine-grained differences between sources containing closely related data. This choice is appropriate for building specialized information servers over a reasonable number of data sources.

Xyleme is representative of information integration agents that have to integrate a huge number of sources covering very broad topics. In such a setting, the mediated schema and the mappings must be simple because the challenging issue is to obtain them as automatically as possible. In particular, the number of mappings is too huge to build them manually.

We have outlined the impact of the choice of the knowledge representation formalism on the query reformulation problem, which is the core algorithmic problem for answering queries in an information integration system. Clearly, as the languages for describing data sources, the mediated schema, or the users' queries become more expressive, the query reformulation problem becomes harder. The key challenge is then to identify formalisms offering a reasonable tradeoff between expressive power and good computational properties for the accompanying reformulation algorithm.

Despite their differences, PICSEL and Xyleme both illustrate a centralized approach of mediation based on a single mediated schema. For scaling up to the Web, this centralized approach of mediation is probably not flexible enough, and *distributed systems of mediation* are more appropriate.

The Semantic Web envisions a world-wide distributed architecture where data and computational resources will easily interoperate to coordinate complex tasks such as answering queries or global computing. Semantic marking up of web resources using *ontologies* is expected to provide the necessary glue for making this vision work. I think that the de-centralized nature of the Web makes inevitable that communities of users or software developers will use their own ontologies to describe their data or services. In this vision of the Semantic Web based on distributed ontologies, the key point is the mediation between data, services and users, using mappings between ontologies. Complex mappings and reasoning about those mappings are necessary for comparing and combining ontologies, and for integrating data or services described using different ontologies. For an easy deployment of distributed peer to peer data management systems at the scale of the Web, it will be essential to use expressive and declarative languages for describing semantic relationships between ontologies serving as schemas of distributed data or services. CARIN has the potential to be a useful modeling tool for expressing and reasoning on complex mappings between ontologies.

## Acknowledgments:

We would like to thank the PICSEL and Xyleme teams for many fruitful discussions and particularly Serge Abiteboul, Bernd Aman, Sophie Cluet, Claude Delobel, Christine Froidevaux, Irini Fundulaki, Francois Goasdoue, Tova Milo, Brigitte Safar, Jean-Pierre Sirot, Pierangelo Veltri and Dan Vodislav.

## References


[1] V. Aguiléra, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez. Querying the XML Documents on the Web. In *Proc. of the ACMSIGIR Workshop on XML and I. R., Athens*, July 28, 2000.

[2] B. Amann, I. Fundulaki, and M. Scholl. Integrating Ontologies and Thesauri for RDF Schema Creation and Medata Querying. *International Journal of Digital Libraries*, 2000.

[3] Yigal Arens, Chung-Nan Hsu, and Craig A. Knoblock. Query processing in the SIMS information mediator. In Austin Tate, editor, *Advanced Planning Technology*, pages 61–69. AAAI Press, Menlo Park, California, 1996.

[4] Yigal Arens and Craig A. Knoblock. SIMS: Retrieving and integrating information from multiple sources. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 562–563, Washington, D.C., 26–28 May 1993.

[5] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The MOMIS project demonstration. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10–14, 2000, Cairo, Egypt*, pages 611–614.

[6] A. Bidault, C. Froidevaux, and B. Safar. Repairing queries in a mediator approach. In *14th European Conference on Artificial Intelligence*, pages 406–410, Berlin, 2000.

[7] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. Alperin Resnick. CLASSIC: a structural data model for objects. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 18(2):58–67, June 1989.

[8] Diego Calvanese, Giuseppe De Giacomo, G. D., and Maurizio Lenzerini. Answering queries using views in description logics. In *Proceedings of AAAI 2000*.

[9] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer 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.

[10] J. Clark and S. DeRose (eds). *XML Path Language (XPath).* W3C Recommandation, 1999.
`http://www.w3c.org/TR/xpath`.

[11] Sophie Cluet, Pierangelo Veltri, and Dan Vodislav. Views in Large Scale XML Repository. In *Proc. Int. Conf. on Very Large Data Bases*, 2001.

[12] C-web project.
`http://cweb.inria.fr`.

[13] Claude Delobel, M-Christine Rousset, Chantal Reynaud, J-Pierre Sirot and Dan Vodislav. Semantic Integration in Xyleme: a Uniform Tree-based Approach. In *Journal on Data & Knowledge Engineering*, to appear.

[14] Oren Etzioni and Daniel Weld. A Softbot-Based Interface to the Internet. *Communications of the ACM*, 37(7):72–76, 1994.

[15] Marc Friedman and Daniel S. Weld. Efficiently executing information-gathering plans. In *15th International Joint Conference on Artificial Intelligence*, pages 785–791, Nagoya, Japan, 1997.

[16] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: an information integration system. In *Proceedings of SIGMOD 97* pages 539–542, New York, May 1997.

[17] François Goasdoué and Marie-Christine Rousset. Compilation and Approximation of Conjunctive Queries by Concept Descriptions. in *Proceedings of ECAI 2002*.

[18] François Goasdoué, Véronique Lattes, and Marie-Christine Rousset. The Use of CARIN Language and Algorithms for Information Integration: The PICSEL System. *International Journal of Cooperative Information Systems*, 9(4):383–401, 2000.

[19] Alon Halevy. Answering queries using views: a survey. *VLDB Journal*, 2001.

[20] Carl-Christian Kanne and Guido Moerkotte. Efficient storage of xml data. Technical report, University of Mannheim, http://pi3.informatik.uni-mannheim.de/, 1999.

[21] Alon Levy and Marie-Christine Rousset. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence Journal*, 104, pp 165-209, 1998.

[22] Laurent Mignet, Mihai Preda, Serge Abiteboul, Sebastien Ailleret, Bernd Amann, and Amelie Marian. Acquiring xml pages for a webhouse. In *BDA '00*, 2000.

[23] G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11), 1995.

[24] Benjamin Nguyen, Serge Abiteboul, Gregory Cobena, and Mihai Preda. Monitoring xml data on the web. In *VLDB'01*, 2001.

[25] J. Ordille, A. Levy, and A. Rajaraman. Querying Heteregeneous Information Sources Using Source Descriptions. In *Proc. Int. Conf. on Very Large Data Bases*, pages 251–262, 1996.

[26] R. MacGregor and R. Bates. The LOOM knowledge representation language. Technical Report ISI/RS-87-188, Information Science Institute, University of Southern California, Marina del Rey (CA), USA, 1987.

[27] E. Mena, Vipul Kashyap, Amit Sheth, and A. Illarramendi. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *4th Int. Conf. on Cooperative Information Systems*, pages 14–25, Bruessels, Belgium, 1996.

[28] Y. Papakonstantinou, H. Garcia-molina, and J. Widom. Object Exchange Across Heteregeneous Information Sources. In *ICDE Conf. on Management of Data*, 1995.

[29] Jean-Pierre Sirot. Documents xml et serveurs d'information: approche, techniques et outil d'indexation bases sur l'utilisation d'une ontologie. Technical report, Rapport de stage de DEA I3, University of Paris-Sud, 2000.

[30] V. S. Subrahmanian, Sibel Adali, Anne Brink, Ross Emery, James J. Lu, Adil Rajput, Timothy J. Rogers, Robert Ross, and Charles Ward. HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.

[31] XQuery 1.0: An XML Query Language.
`http://www.w3.org/TR/xquery/`.

[32] Xyleme.
`http://www.xyleme.com`.

[33] Lucie Xyleme. A dynamic warehouse for xml data of the web. *IEEE Data Engineering Bulletin*, 2001.