INFSYS

RESEARCH

REPORT

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

# A KNOWLEDGE-BASED APPROACH TO INFORMATION SITE SELECTION


Thomas EITER Hans TOMPITS

Michael FINK



INFSYS RESEARCH REPORT 1843-02-09

JULY 2002



Institut für Informationssysteme

Abtg. Wissensbasierte Systeme

Technische Universität Wien

Favoritenstraße 9-11

A-1040 Wien, Austria

Tel: +43-1-58801-18405

Fax: +43-1-58801-18493

sek@kr.tuwien.ac.at

www.kr.tuwien.ac.at


TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-02-09, JULY 2002

# A KNOWLEDGE-BASED APPROACH TO INFORMATION SITE SELECTION

Thomas Eiter, Michael Fink, and Hans Tompits [1]

**Abstract.** With the World Wide Web, a vast number of information sources has become available. In order to access and process these data, suitable tools and methods for building an information infrastructure are necessary. One task for this purpose is the selection of relevant information sources in query answering. In this paper, we present an knowledge-based approach for information-site selection. We use extended logic programs (ELPs) to represent rich descriptions of the information sources, an underlying domain theory, and queries in a formal language. Moreover, we use ELPs for declarative query analysis and query abstraction. Central to our approach are declarative *site-selection programs*, for which we define a syntax and semantics. Due to the structured nature of data items, the semantics of such programs must carefully respect implicit context information in site-selection rules, and furthermore combine it with possible user preferences. A prototype implementation has been realized exploiting the `DLV` KR system and its `plp` front-end for prioritized ELPs. We describe experimental results for this implementation, using a representative example from the movie domain.

[1]Institut für Informationssysteme, Abteilung Wissensbasierte Systeme 184/3, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria. E-mail: [eiter, michael, tompits]@kr.tuwien.ac.at

**Acknowledgements**: This work was supported by the Austrian Science Fund (FWF) projects P13871-INF and N Z29-INF.
We acknowledge Compaq Equipment Corporation for providing the EachMovie Database.

# Contents

**1 Introduction** **1**

**2 System Overview** **2**
2.1 Architecture . . . . . 2
2.2 Site Selection Capability . . . . . 2

**3 Preliminaries** **4**

**4 Query Representation and Abstraction** **6**
4.1 Low-level Representation . . . . . 6
4.2 High-level Predicates . . . . . 7
4.3 Query Abstraction Program . . . . . 8

**5 Site Description** **11**

**6 Site Selection** **12**
6.1 Syntax . . . . . 12
6.2 Semantics . . . . . 14
6.3 Properties . . . . . 17

**7 Implementation and Application** **21**
7.1 Implementation . . . . . 21
7.2 Application: Movie Databases . . . . . 21
7.2.1 Movie Sites . . . . . 21
7.2.2 Domain Knowledge . . . . . 22
7.2.3 Site-Selection Program . . . . . 24

**8 Experiments** **25**
8.1 Results . . . . . 26
8.2 Results with Modified Selection Bases . . . . . 27

**9 Related Work and Conclusion** **28**
9.1 Conclusion and further work . . . . . 31

**A Appendix: XML DTD for the Movie Databases** **35**

**B Appendix: Low-level Query-Representation Examples** **36**

**C Appendix: High-level Query-Description Examples** **38**

**D Appendix: Experimental Movie Doamin Knowledge** **39**

**E Appendix: Experimental Site Description Program** **51**

**F Appendix: Experimental Site-Selection Program** **54**

**G Appendix: Experimental XML-QL Queries** **60**

# 1 Introduction

With the advent of the World Wide Web, a wealth of information has become available to a large group of users. However, the distribution of the information sources, their heterogeneity and diverse ways of accessibility have created the need for suitable tools and methods for building an information processing infrastructure. A promising approach is based on multi-agent systems, in which *mediators* [44] or special *information agents* ("middle agents" [13]) provide various services including finding, selecting, and querying relevant information sources.

The potential of knowledge-based approaches—and in particular of logic programming—for developing reasoning components of intelligent information agents is outlined in [17, 20, 41]. In this paper, we pursue this further and present a declarative approach to information source selection. Here, the problem is, given a query by the user, which out of a collection of information sources should the agent select for answering the query, such that the utility of the answer, in terms of quality of the result and other criteria (e.g., costs), is as large as possible for the user? Obviously, a sensible solution to this problem is nontrivial and involves various aspects such as basic properties of the information sources, knowledge about their contents, and knowledge about the particular application domain. These aspects have to be suitably combined, and reasoning may be needed to elicit implicit knowledge. Our approach, which incorporates such aspects, makes several contributions, which are briefly summarized as follows.

(1) We use extended logic programs (ELPs) [25] to represent rich descriptions of the information sources, an underlying domain theory, and queries in a formal language. We perform query analysis by ELPs and compute *query abstractions*. Here, we consider XML-QL [16], but other formal query languages could be handled as well.

(2) At the heart, a declarative *site selection program* represents *both qualitative and quantitative* criteria (e.g., site preference and costs). Its rules may access information supplied by the other programs, including object and value occurrences in the query. For example, a rule $r_1$ may state that a query about actor awards should be posed to source $s_1$.

(3) We consider the interesting and, to our knowledge, novel issue of *contexts* in logic programs. Structured data items require a careful definition of the selection semantics. If besides $r_1$ above a rule $r_2$ says that source $s_2$ is in general best for awards, then $r_1$ should have priority over $r_2$ and $s_1$ should be queried. Note that this priority is not based on inheritance, but on the *context* of the occurrence of awards, which is given by the query access path. Thus, inheritance-based approaches such as [9] do not apply obviously. Furthermore, implicit priorities, derived from context information, must be combined with explicit user preferences from the selection policy, and arising conflicts must be resolved.

(4) We have implemented a prototype, which we used to build an example application in the movie domain. It comprises several XML databases, wrapped from movie databases on the web, and handles queries in XML-QL. Experiments that we have conducted show that the system behaved intuitively on a number of natural queries, some of which require reasoning from the background knowledge to identify the proper selection.

Our motivation to use a knowledge-based approach—and in particular an answer-set programming approach—to site selection rather than a standard decision theoretic approach based on utility functions is driven by the following advantages:

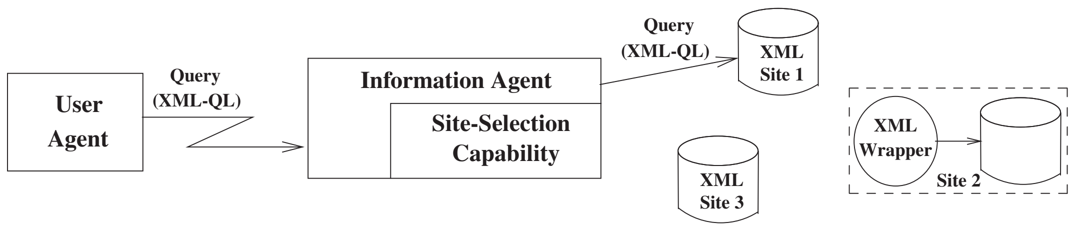


Figure 1: Architecture of the site selection system

- Site selection programs are declarative and have a clear formal semantics, both under qualitative as well as quantitative criteria.
- The formalism is capable of handling incomplete information and performing nonmonotonic inferences, which, arguably, is an inherent feature of the current problem domain.
- Changes in the specification of the site-selection process are easily incorporated by modifying or adding suitable rules or constraints, without the need for re-designing the given program, as might be the case, e.g., in procedural languages.
- Finally, the declarative nature of the answer set semantics formalism permits also a coupling with sophisticated ontology tools, and with reasoning engines for them, providing advanced features for the domain knowledge.

The rest of this paper is organized as follows. The next section briefly outlines our approach, and Section 3 introduces some background terminology. In Section 4, we describe elementary query representation and abstraction to high level concepts. Section 5 addresses the modeling of sites, while Section 6 is devoted to site-selection programs. The implementation and the movie application are addressed in Section 7, while experiments are reported in Section 8. Section 9 discusses related work and concludes the paper.

# 2 System Overview

## 2.1 Architecture

A prototypical site selection setup is shown in Figure 1, where data are stored in XML databases. Queries are posed in an XML query-language such as XML-QL [16], which we use here. Some of the databases may be wrapped from non-XML data sources. However, our approach is not committed to semistructured data and XML; other models and languages (e.g., relational databases and SQL) may also be used.

A query is handed over to an information agent, which has to pick one of several databases that comply with the same (universal) schema to answer the query. The answers to queries may differ on aspects such as contents, quantity, cost, reliability, and so on. Merging data from different sources, query rewriting, etc., do not play a role here.

## 2.2 Site Selection Capability

Our approach is based on the *answer-set programming* paradigm [35], in which problems are represented by extended logic programs (ELPs) [25], possibly augmented with priorities (e.g., [8, 15, 29]) and weak

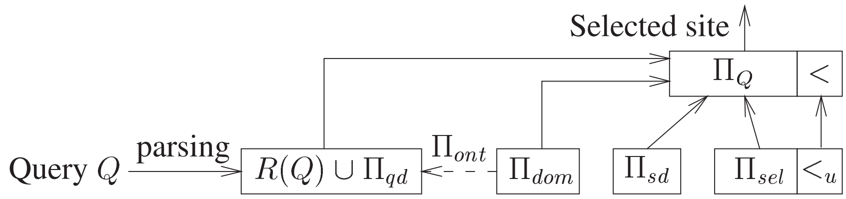


Figure 2: Using a selection base $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ for site selection

constraints [10], and solutions are obtained from the answer sets of the programs (see Section 3). The following scenario illustrates our ideas.

**Example 1** Consider an agent which knows XML information sources $s_1$, $s_2$, and $s_3$ about movies. Assume that the following XML-QL [16] query is handed to the agent, which informally asks a source for titles of all movies directed by Alfred Hitchcock:

```
FUNCTION HitchcockMovies($MovieDB:"Movie.dtd") {
  CONSTRUCT <MovieList> {
    WHERE <MovieDB> <Movie>
             <Title> $t </Title>
             <Director> <Personalia>
                 <FirstName> "Alfred" </FirstName>
                 <LastName> "Hitchcock" </LastName>
             </Personalia> </Director>
          </Movie> </MovieDB>
    IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
 } </MovieList> }
```

Suppose the agent knows that $s_1$ is a very good source for information about directors, while $s_2$ has usually good coverage about person data; all known about $s_3$ is that it is not much reliable. In this situation, we would expect the agent to select $s_1$ for querying.

While the above example is simple, it shows that the selection process uses different kinds of knowledge. In our approach, this is reflected by the components of a *selection base* $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ as described below (see Fig. 2):

**Query description** $\Pi_{qd}$**:** For any query $Q$ as in Example 1, a high-level abstract description is extracted from a low-level (syntactic) representation $R(Q)$, given as a set of elementary facts, by applying a logic program $\Pi_{qd}$ to $R(Q)$ and ontological knowledge $\Pi_{ont}$. Informally, the rules of $\Pi_{qd}$ single out the essential parts of $Q$, such as occurrence of attributes and values in the query, comparisons and joins, or access paths describing contexts.

**Domain theory** $\Pi_{dom}$**:** The agent's knowledge about the application domain (e.g., movies) is represented in a logic program $\Pi_{dom}$. It includes an ontology, $\Pi_{ont}$, and further background knowledge, permitting (modest) commonsense reasoning. The ontology may be established using meta-information about the data in the information sites (e.g., an XML DTD).

**Site description** $\Pi_{sd}$**:** Information about the sites is represented in a logic program $\Pi_{sd}$, using special predicates for query topics, cost aspects, and technical aspects.

**Site-selection program $\Pi_{sel}$:** The information site selection is specified by rules and constraints, which refer to predicates defined in the above programs. It comprises both *qualitative aspects* and *quantitative aspects* in terms of optimization criteria (concerning, e.g., cost or response time), which are expressed using weak constraints [10]. Furthermore, the user can define preferences between rules, in terms of a strict partial order, $<_u$. These preferences are combined with implicit priorities that emerge from the *context* in which site selection rules should be applied, and possible preference conflicts are resolved.

Given a query $Q$, the overall evaluation proceeds in three steps:

**Step 1 (query representation):** The input query $Q$ is parsed and mapped into the internal query representation $R(Q)$.

**Step 2 (qualitative selection):** From $R(Q)$, $\Pi_{sd}$, and $\Pi_{dom}$, the qualitative part of $\Pi_{sel}$ is used to single out different query options by respecting qualitative aspects only, where explicit preferences, $<_u$, and implicit priorities must be taken into account. To this end, a priority relation $<$ is computed on rules, which is then used in a prioritized logic program $(\Pi_Q, <)$. Candidate solutions are computed as answer sets of $(\Pi_Q, <)$.

**Step 3 (optimization):** Among the candidates of Step 2, the one is chosen which is best under the quantitative aspects of $\Pi_{sel}$, and the selected site is output.

## 3 Preliminaries

We briefly recall some concepts from answer-set programming. An *extended logic program* (ELP) [25] is a finite set of rules $r$ in a function-free first order language of form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n,$$

where each $L_i$, $0 \leq i \leq n$, is a *literal*, i.e., an atom $A$ or a negated atom $\neg A$; "*not*" denotes negation as failure. Intuitively, rule $r$ means that we can conclude $L_0$ if (i) $L_1, \ldots, L_m$ are known and (ii) $L_{m+1}, \ldots, L_n$ are *not* known. We call the literal $L_0$ the *head* of $r$ (denoted $H(r)$) and the set $\{L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\}$ the *body* of $r$ (denoted $B(r)$). Furthermore, we define $B^+(r) = \{L_1, \ldots, L_m\}$ and $B^-(r) = \{L_{m+1}, \ldots, L_n\}$. If $B(r) = \emptyset$, then $r$ is called a *fact*. To ease notation, for any program $\Pi$ and any set $S$ of literals, we write $\Pi \cup S$ to refer to the program $\Pi \cup \{L \leftarrow \mid L \in S\}$. Finally, for literal $L$, we write $\neg L$ to denote its complementary literal, i.e., $\neg L = A$ if $L = \neg A$, and $\neg L = \neg A$ if $L = A$, for any atom $A$.

The semantics of any ELP $\Pi$ is given in terms of the semantics of its ground instantiation, $ground(\Pi)$, over the Herbrand universe. In what follows, assume that all objects (literals, rules, etc.) are ground.

An *interpretation*, $I$, is a consistent set of (ground) literals, i.e., $I$ does not contain a complementary pair $A$, $\neg A$ of literals. A literal $L$ is *true* in $I$ (in symbols, $I \models L$) if $L \in I$ and *false* otherwise. The body $B(r)$ of a rule $r$ is true in $I$ (denoted $I \models B(r)$) iff (i) each $L \in B^+(r)$ is true in $I$ and (ii) each $L \in B^-(r)$ is false in $I$. Rule $r$ is true in $I$ ($I \models r$) iff $H(r)$ is true in $I$ whenever $B(r)$ is true in $I$. $I$ is a *model* of program $\Pi$, denoted $I \models \Pi$, if $I \models r$ for all $r \in \Pi$.

The *reduct*, $\Pi^S$, of a program $\Pi$ *relative to* a set $S$ of literals is defined by

$$\Pi^S = \{H(r) \leftarrow B^+(r) \mid r \in \Pi \text{ and } B^-(r) \cap S = \emptyset\}.$$

An interpretation $I$ is an *answer set* of a program $\Pi$, if $I$ is a minimal model of $\Pi^I$ with respect to set inclusion. Observe that any answer set of $\Pi$ is *a fortiori* a model of $\Pi$. The set of all *generating rules* of an answer set $S$ from $\Pi$ is given by $GR(\Pi, S) = \{r \in \Pi \mid S \models B(r)\}$.

**Example 2** Let $\Pi = \{\, s \leftarrow not\, t,\; n \leftarrow\,,\; t \leftarrow\; n, not\, s,\; w \leftarrow t \,\}$. For the interpretation $S = \{n, t, w\}$, we have $\Pi^S = \{n \leftarrow\,,\; t \leftarrow\; n,\; w \leftarrow t\}$. Clearly, $S$ is a minimal model of $\Pi^S$, and thus $S$ is an answer set of $\Pi$. Note that $S' = \{s, n\}$ is another answer set of $\Pi$.

A refinement of the answer-set semantics is the admission of preferences among the rules of a given ELP, yielding so-called *prioritized logic programs*. Several approaches along this way have been introduced in the literature, like, e.g., [8, 29]; here, we use a preference approach based on a method due to Delgrande, Schaub, and Tompits [15].

Let $\Pi$ be an ELP and $<$ a strict partial order between elements of $\Pi$ (i.e., $<$ is an irreflexive and transitive relation). Informally, for rules $r, s \in \Pi$, the relation $r < s$ expresses that $s$ has preference over $r$. Define the relation $<_{\mathcal{G}}$ over the ground instantiation $ground(\Pi)$ of $\Pi$ by setting $\hat{r} <_{\mathcal{G}} \hat{s}$ iff $r < s$, for $\hat{r}, \hat{s} \in ground(\Pi)$. Then, the pair $(\Pi, <)$ is called a *prioritized extended logic program*, or simply a *prioritized logic program*, if the relation $<_{\mathcal{G}}$ is a strict partial order.

The semantics of prioritized programs is as follows. Let $(\Pi, <)$ be a prioritized logic program where $\Pi$ is ground, and let $S$ be an answer set of $\Pi$. We call $S$ a *preferred answer set of* $(\Pi, <)$, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of $GR(\Pi, S)$ such that, for every $i, j \in I$, we have that:

$(P_1)$ $B^+(r_i) \subseteq \{H(r_k) \mid k < i\}$;

$(P_2)$ if $r_i < r_j$, then $j < i$; and

$(P_3)$ if $r_i < r'$ and $r' \in \Pi \setminus GR(\Pi, S)$, then $B^+(r') \not\subseteq S$ or $B^-(r') \cap \{H(r_k) \mid k < i\} \neq \emptyset$.

Conditions $(P_1)$ – $(P_3)$ realize a strongly "prescriptive" interpretation of preference, in the sense that, whenever $r < s$ holds, it is ensured that $s$ is known to be applied or blocked ahead of $r$ (with respect to the order of rule application). More specifically, $(P_2)$ guarantees that all generating rules are applied according to the given order, whilst $(P_3)$ assures that any preferred yet inapplicable rule is either blocked due to the non-derivability of its prerequisites or because it is defeated by higher ranked or unrelated rules. As shown in [15], preferred answer sets can be encoded by means of a suitable translation from prioritized logic programs into standard ELPs.

Preferred answer sets of a prioritized program $(\Pi, <)$ where $\Pi$ is non-ground are given by the preferred answer sets of the prioritized program $(ground(\Pi), <_{\mathcal{G}})$, where $<_{\mathcal{G}}$ is as above. Note that the concept of prioritization realizes a *filtering* of the answer sets of a given program $\Pi$, as every preferred answer set of $(\Pi, <)$ is an answer set of $\Pi$, but not vice versa.

Besides imposing *qualitative selection criteria*, like assigning preferences between different rules, another refinement of the answer-set semantics are *weak constraints* [10], representing a *quantitative filtering of answer sets*. Formally, a weak constraint is an expression of form

$$\Leftarrow\; L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n\; [w : l], \tag{1}$$

where each $L_i$, $1 \leq i \leq n$, is a literal and $w, l \geq 1$ are natural numbers. The number $w$ is the *weight* and $l$ is the *priority level* of the weak constraint (1). Informally, weak constraints work as follows: From the answer sets of the weak-constraint free part of a program $\Pi$, prune first those where the sum of weights of violated constraints in the highest priority level is not minimal, then those where the sum of weights of violated constraints in the next lower level is not minimal, and so on. We refer the reader to [10] for the precise formal definition.

# 4 Query Representation and Abstraction

An important aspect of our approach is a meaningful description of a given query $Q$. For our purposes, we need a suitable representation of the constituents of $Q$ in terms of predicates and objects. Simply mapping $Q$ to logical facts reflecting its syntactic structure, as given by the query grammar, will not serve our purposes. Rather, we need an *abstract* description which provides "interesting" information about $Q$, such as occurrence of an attribute or a value in $Q$, related to the scope of appearance. For example, the value "Hitchcock" occurs in the query of Example 1 in a selection condition, which is performed on the attribute $LastName$ reached by the access path $Movie/Director/Personalia$.

In our query abstraction method, we adopt a general view in which a query consists of a *construct part*, a *where part*, and a *source part*, corresponding to the SQL *select*, *where*, and *from parts*, respectively. We use a high-level description of $Q$ independent of a fixed query language. It is extracted from a low-level query representation $R(Q)$, which is generated by a parser and contains structural details, using a logic program $\Pi_{qd}$ and relevant ontology knowledge $\Pi_{ont}$. In detail, the high-level description is given by the (unique) answer set of $\Pi_{ont} \cup \Pi_{qd} \cup R(Q)$.

## 4.1 Low-level Representation

The representation $R(Q)$ is a first level of abstraction from the "physical" (syntactic) query expression $Q$. The query and its syntactic subqueries are named by constants (e.g., $q_1, q_2, \ldots$). References to items (attributes or concepts) which are selected or compared to values or other items are given by maximal access paths in $Q$, which we name by unique constants and refer to as *item reference paths (IRPs)*.

**Example 3** In Example 1, the query is named $q_1$. It has a subquery $q_2$ in the `CONSTRUCT <MovieList>` part. There are three IRPs, "*Movie/Title*", "*Movie/Director/Personalia/FirstName*", and "*Movie/Director/Personalia/LastName*", which we name $o_1$, $o_2$, and $o_3$, respectively.

The program $R(Q)$ is expressed in terms of the following predicates:

- $sub\_query(Q', Q)$: $Q'$ is a structural subquery of query $Q$ (possibly itself a subquery);
- $query\_cand(Q)$: identifies the overall query;
- $source(S, Q)$: query $Q$ accesses source $S$;
- $db\_name(S)$: source $S$ is a database;
- $whereRef(O, T, P, Q)$: an IRP $O$ accesses an item under element $T$ and remainder path $P$ in the where part of query $Q$;
- $subpath(O, T_1, P_1, T_2, P_2)$: the path $T_2/P_2$ is a direct subpath of $T_1/T_2$ in the IRP $O$;
- $whereRefCmp(O_1, R, O_2)$: the items of IRPs $O_1$ and $O_2$ are compared using operator $R$;
- $whereCmp(O, R, V)$: the item of IRP $O$ is compared to value $V$ using operator $R$;
- $consRef(O, T, P)$: the item of IRP $O$ is constructed under element $T$ and remainder path $P$ in the (answer-) construction part of query $Q$;

$R(Q)$ must respect that query languages may allow for nested queries. However, in a query expression, an outmost query as the "root" of nesting should be identifiable, as well as structural (syntactic) subqueries of it. They are described using $query\_cand$ and $sub\_query$, respectively.

Along an IRP, item references relative to a position are captured by the $whereRef$ predicate, and suffix inclusions for this IRP are stored as $subpath$ facts. The predicates $whereRefCmp$ and $whereCmp$ mirror the comparison of two items and the comparison of an item with a value, respectively. Items that occur in the construction part of a query are also identified by an IRP and stored using $consRef$.

**Example 4** The low-level representation $R(Q)$ of the query in Example 1, contains

$$sub\_query(q_2, q_1).\ query\_cand(q_1).\ source(\text{“}MovieDB\text{”}, q_2).\ db\_name(\text{“}MovieDB\text{”}).$$

and, e.g., for the third IRP, $o_3$, which accesses "*LastName*", the facts:

$whereRef(o_3, \text{“}LastName\text{”}, \text{“ ”}, q_2).$
$whereRef(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_2).$
$whereRef(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_2).$
$whereRef(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_2).$
$whereRef(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_2).$
$subpath(o_3, \text{“}LastName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}LastName\text{”}).$
$subpath(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}).$
$subpath(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}).$
$subpath(o_3, \text{”}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”},$
$\quad \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}).$
$whereCmp(o_3, equal, \text{“}Hitchcock\text{”}).$

The complete low-level representation $R(Q)$ of the query is given in Appendix B.

### 4.2 High-level Predicates

The high-level description enriches and abstracts from $R(Q)$, and brings in ontological knowledge. The following notion is convenient.

**Definition 1** *A pair $(C, P)$, where $C$ is a concept from the ontology and $P$ is a path, is a* context-reference pair *(CRP) of a query $Q$, if $Q$ contains an IRP with suffix "$C/P$."*

In Example 1, $(\text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”})$ and $(\text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”})$ are some CRPs of the query $q_2$.

The following high-level description predicates are defined:

- $query(Q)$: identifies an "independent" (sub-)query $Q$ (i.e., executable on some site), which is, moreover, not embraced by a sourceless query $Q'$ merely restructuring the result of $Q$.
- $access(O, C, P, Q)$: states that $(C, P)$ is a CRP for $Q$ via IRP $O$ in the where-part of $Q$.
- $occurs(O, V)$: the value $V$ is associated with an IRP $O$ in the overall query.

- $selects(O, R, V)$: like $occurs$, but details the association with a comparison operator $R$.
- $constructs(O, I, P)$: states that the IRP $O$, by use of a variable, also appears in the construct-part of the global query, as an item $I$ under path $P$ (which may be different from the path in the where-part).
- $joins(O_1, O_2, R)$: records (theta-)joins of (or within) queries between IRPs $O_1$ and $O_2$ under comparison operator $R$.

**Example 5** Reconsidering the query in Example 1, we show part of its high-level description. The complete description is given in Appendix C.

We have $query(q_1)$, but not $query(q_2)$ since the embracing query $q_1$ has no source and merely structures the result of $q_2$. The following $access$ facts result from $o_1$ and $o_3$:

$access(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_1).$
$access(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_1).$
$access(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1).$

Here, $MovieDB$, $Movie$, $Director$, and $Person$ are concepts given by the ontology, and $Personalia$ is known to be a synonym of $Person$ (see Section 4.3 for further discussion).

The fact $occurs(o_3, \text{“}Hitchcock\text{”})$ states that value $Hitchcock$ is associated with $o_3$. This is detailed by $selects(o_3, equal, \text{“}Hitchcock\text{”})$, where $equal$ represents equality. For the $constructs$ predicate, the fact $constructs(o_1, \text{“}Movie\text{”}, \text{“ ”})$. is included. There are no $joins$ facts since the query has no join.

### 4.3 Query Abstraction Program

The high-level description facts are obtained by applying the query abstraction program, $\Pi_{qd}$, using the facts of $R(Q)$ and some ontological facts $\Pi_{ont}$ as input. $\Pi_{qd}$ comprises a set of rules, which we present in the rest of this section. They compute the predicates from Section 4.2, where auxiliary predicates are used. They furthermore enlarge the low-level predicate $subpath$, by taking them as well as $\Pi_{ont}$ into account as follows:[1]

$$
\begin{array}{lrcl}
qd_1: & subpath(O, T_1, P_1, T_3, P_3) & \leftarrow & subpath(O, T_1, P_1, T_2, P_2),\ subpath(O, T_2, P_2, T_3, P_3).\\
qd_2: & subpath(O, T, P_1, T_2, P_2) & \leftarrow & subpath(O, L, P_1, T_2, P_2),\ synonym(L, T).\\
qd_3: & subpath(O, T_1, P_1, T, P_2) & \leftarrow & subpath(O, T_1, P_1, L, P_2),\ synonym(L, T).
\end{array}
$$

Rules $qd_1$–$qd_3$ consider transitivity and synonyms for elements occurring in paths, which is imported ontological knowledge; $synonym$ applies to all pairs of synonymous element names (e.g., names of IDREF attributes).

The following two rules define useful projections of low-level predicates:

$$
\begin{array}{lrcl}
qd_4: & has\_source(Q) & \leftarrow & source(\_, Q).\\
qd_5: & is\_sub\_query(Q) & \leftarrow & sub\_query(Q, \_).
\end{array}
$$

[1] In a clean separation of $R(Q)$ and high-level description, a fresh predicate would be in order here. However, it is convenient and economic to reuse the predicate $subpath$, as it is only enlarged.

Using them, an auxiliary predicate $iquery\_cand$ is defined for candidates which might satisfy the $query$ predicate; these are the overall query and subqueries having a database or a document as its source.

$$\begin{array}{lcl} qd_6: iquery\_cand(Q) & \leftarrow & query\_cand(Q). \\ qd_7: iquery\_cand(Q) & \leftarrow & is\_sub\_query(Q), source(X,Q), db\_name(X). \end{array}$$

The high-level predicates are now defined as follows. Independent, separate queries are specified by respecting the nesting structure:

$$\begin{array}{rcl} qd_8: \; top\_query(S,Q) & \leftarrow & sub\_query(S,X), not\ iquery\_cand(S), top\_query(X,Q). \\ qd_9: \; top\_query(S,Q) & \leftarrow & sub\_query(S,X), iquery\_cand(S), top\_query(X,Q), \\ & & not\ has\_source(X). \\ qd_{10}: top\_query(Q,Q) & \leftarrow & iquery\_cand(Q), not\ is\_sub\_query(Q). \\ qd_{11}: top\_query(Q,Q) & \leftarrow & iquery\_cand(Q), sub\_query(Q,S), source(X,S). \\ qd_{12}: \quad query(Q) & \leftarrow & top\_query(Q,Q). \end{array}$$

A query is considered to be independent if it is the topmost independent query of itself ($qd_{12}$). This is the case if the query is a candidate for a separate query and it is either the outmost query ($qd_{10}$) or a direct structural subquery of a query to a source ($qd_{11}$). Intuitively, $qd_{11}$ says that a candidate separate query nested within another query is viewed as a separate query only if the nesting was not for mere syntactic reasons. In the latter case, and if a nested query is not a candidate for a separate query, its topmost independent query is the one of the embracing query ($qd_9$ and $qd_8$, respectively).

The next rules define the remaining high-level description predicates. An auxiliary predicate, namely $has\_constructs$, guarantees that at least one $constructs$ fact is generated for each context reference constructed in the query answer.

$$\begin{array}{lrcl} qd_{13}: & access(O,T,P,Q) & \leftarrow & whereRef(O,T,P,S), class0(T), top\_query(S,Q). \\ qd_{14}: & access(O,T,P,Q) & \leftarrow & whereRef(O,L,P,S), synonym(L,T), class0(T), \\ & & & top\_query(S,Q). \\ qd_{15}: & constructs(O,T,P) & \leftarrow & consRef(O,T,P), class0(T). \\ qd_{16}: & constructs(O,T,P) & \leftarrow & consRef(O,L,P), synonym(L,T), class0(T). \\ qd_{17}: & has\_constructs(O) & \leftarrow & consRef(O,T,P), class0(T). \\ qd_{18}: & has\_constructs(O) & \leftarrow & consRef(O,L,P), synonym(L,T), class0(T). \\ qd_{19}: & constructs(O,\text{“ ”},\text{“ ”}) & \leftarrow & consRef(O,\_,\_), not\ has\_constructs(O). \\ qd_{20}: & occurs(O,V) & \leftarrow & whereCmp(O,C,V). \\ qd_{21}: & selects(O,C,V) & \leftarrow & whereCmp(O,C,V). \\ qd_{22}: & joins(O_1,O_2,C) & \leftarrow & whereRefCmp(O_1,C,O_2). \end{array}$$

Note that some rules reference the ontology predicate $class0$. A fact $class0(e)$ should exist in (or entailed by) the domain ontology for all elements $e$ that are considered to be concepts.

When queries are joined over CRPs, then some of the occurrence, selection, and construction information of one CRP is also valid for the other. Hence, we can build a form of a closure over joined CRPs, which is expressed by the following rules:

$$\begin{array}{lrcl} qd_{23}: & constructs(O_1,T,P) & \leftarrow & joins(O_1,O_2,equal), constructs(O_2,T,P). \\ qd_{24}: & constructs(O_2,T,P) & \leftarrow & joins(O_1,O_2,equal), constructs(O_1,T,P). \end{array}$$

$$
\begin{array}{rrcl}
qd_{25}: & occurs(O_1, V) & \leftarrow & joins(O_1, O_2, C), occurs(O_2, V). \\
qd_{26}: & occurs(O_2, V) & \leftarrow & joins(O_1, O_2, C), occurs(O_1, V). \\
qd_{27}: & selects(O_1, C, V) & \leftarrow & joins(O_1, O_2, equal), selects(O_2, C, V). \\
qd_{28}: & selects(O_2, C, V) & \leftarrow & joins(O_1, O_2, equal), selects(O_1, C, V). \\
qd_{29}: & selects(O_1, notequal, V) & \leftarrow & joins(O_1, O_2, notequal), selects(O_2, equal, V). \\
qd_{30}: & selects(O_2, notequal, V) & \leftarrow & joins(O_1, O_2, notequal), selects(O_1, equal, V).
\end{array}
$$

We remark that as easily seen, the rules of $\Pi_{qd}$ form a stratified logic program, and thus $\Pi_{ont} \cup \Pi_{qd} \cup R(Q)$ has a unique answer set.

**Example 6** Let us consider how the high-level fact $access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1)$ is derived in $\Pi_{qd}$ given $R(Q)$ of the query in Example 1.

Since $query\_cand(q_1)$ is in $R(Q)$, we obtain by $qd_6$ $iquery\_cand(q_1)$. Since $is\_sub\_query(q_1)$ is not derivable, by $qd_{10}$ we can derive $top\_query(q_1, q_1)$ (i.e., that $q_1$ is independent). Next, we can derive $is\_sub\_query(q_2)$ by $qd_5$ and thus $iquery\_cand(q_2)$ by $qd_7$, since $R(Q)$ includes $sub\_query(q_2, q_1)$, $source(\text{“}MovieDB\text{”}, q_2)$, and $db\_name(\text{“}MovieDB\text{”})$. Since $q_1$ has no source (i.e., $has\_source(q_1)$ is not derivable), we can derive $top\_query(q_2, q_1)$ from $qd_9$. The fact $access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1)$ is now derived by means of $qd_{14}$, using $whereRef(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_2)$, from $R(Q)$, as well as $synonym(\text{“}Personalia\text{”}, \text{“}Person\text{”})$ and $class0(\text{“}Person\text{”})$ from the ontology, and the derived fact $top\_query(q_2, q_1)$. We remark that $access(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_1)$ is not derivable, since $class0(\text{“}Personalia\text{”})$ is not in $\Pi_{ont}$.

While our method works well for experimental queries that we considered (cf. also Section 8), we can not state that it is complete in some sense, and refinements will be desired. Due to syntactical dependencies and its generalizing nature, “intelligent” declarative query abstraction is, per se, an interesting area for further research, as the following example illustrates.

**Example 7** Consider the more complex XML-QL query asking for movies directed by Josef Sternberg in which Marlene Dietrich acted:

```
FUNCTION SternbergDietrichMovies($MovieDB:"Movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE <MovieDB> <Movie>
        <Title> $t </Title>
           <Director> <Personalia>
              <FirstName> "Josef" </FirstName>
              <LastName> "Sternberg" </LastName>
           </Personalia> </Director>
           <Actor Personalia=$pid>
           </Actor>
        </Movie>
        <Person ID=$pid>
             <FirstName> "Marlene" </FirstName>
             <LastName> "Dietrich" </LastName>
        </Person> </MovieDB>
     IN source($MovieDB)
     CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

This query, like the one of Example 1, complies with the XML-DTD in Appendix A. However the first- and last name information of director Josef Sternberg is accessed syntactically different than the respective information of actress Marlene Dietrich. The context references concerning Sternberg are implicit joins over `Personalia` of type IDREF, while the context references concerning Dietrich make this join explicit.[2] This syntactical difference is reflected in the high-level description of the query, which is provided in Appendix C. Consider, e.g., the following facts and compare the access predicates for last name Sternberg with those for Dietrich:

$access(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1).$
$access(o_4, \text{“}MovieDB\text{”}, \text{“}Movie/Actor/Personalia\text{”}, q_1).$
$access(o_5, \text{“}MovieDB\text{”}, \text{“}Person/ID\text{”}, q_1).$
$access(o_7, \text{“}MovieDB\text{”}, \text{“}Person/LastName\text{”}, q_1).$
$access(o_7, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1).$
$joins(o_4, o_5, equal).$

Although the explicit join is reflected in the high-level description, its connection with the name references has been lost.

These differences cannot be eliminated without the underlying XML-DTD. Thus, to keep the query abstraction process generic, syntactic issues like these must be tackled by the query parser, i.e., the low-level representation. Note, however, that these representational differences have no influence on the site-selection process and experiments in Sections 7 and 8, respectively.

## 5 Site Description

Besides query information and domain knowledge, site selection needs a suitable description of the information sites to select from. This is given in the site description part of the knowledge base, where the following *site description predicates* are available, covering several topics.

- **Thematic aspects:**

  - $accurate(S, T, V)$: site $S$, topic $T$, value $V$;
  - $specialized(S, T)$: site $S$, topic $T$;
  - $covers(S, T, V)$: site $S$, topic $T$, value $V$;
  - $relevant(S, T)$: site $S$, topic $T$.

  The first two predicates express the accuracy and coverage of a site for a topic, using values from $\{low, med, high\}$. The others are for stating that a site is specialized or relevant for a particular topic, respectively.

[2]Currently, XML-QL (version 0.9) does not support implicit joins over IDREF-references. They must be written as explicit joins over the corresponding attribute (e.g., as for the context references concerning Marlene Dietrich).

- **Cost aspects:**

  – $avg\_download\_time(S, V)$: site $S$, value $V$;
  – $charge(S, V)$: site $S$, value $V$;
  – $avg\_down\_time(S, V)$: site $S$, value $V$.

  Costs for accessing an information site can be expressed by these predicates, using values $low$, $med$, $high$, and, for $charge$, also $no$. While $charge$ is used for direct costs, $avg\_download\_time$ and $avg\_down\_time$ are indirect costs (affecting network traffic).

- **Technical aspects:**

  – $site\_type(S, T1, T2)$: site $S$, organizational type $T1$, query type $T2$;
  – $site\_language(S, L)$: site $S$, language $L$;
  – $data\_format(S, F)$: site $S$, format $F$;
  – $update\_frequency(S, V)$: site $S$, value $V$;
  – $last\_update(S, D)$: site $S$, date $F$;
  – $reliable(S, V)$: site$S$, value $V$;
  – $site(S)$: site $S$;
  – $up(S)$: site $S$.

  Different types of sites are distinguished by their organization (commercial or public) and by the type of data access provided (queryable, downloadable, or both). Besides site language and data format (XML, relational, HTML, text, or other), the frequency of data update (low, medium, or high), the date of the last update, or the reliability of a site (low, medium, or high) may be criteria for site selection. Finally, $site$ and $up$ are used to identify sites and to express that a site is currently accessible, respectively.

# 6 Site Selection

In this section, we introduce site-selection programs, which are central to our architecture. Basically, a site-selection program $\Pi_{sel}$ is a prioritized logic program having four parts, namely (i) a core unit $\Pi^c_{sel}$, containing the actual site-selection rules, (ii) a set $\Pi^{aux}_{sel}$ of auxiliary rules, (iii) an order relation $<_u$ defined over members of $\Pi^c_{sel}$, and (iv) an optimization part $\Pi^o_{sel}$, containing weak constraints.

## 6.1 Syntax

**Definition 2** *The alphabet, $\mathcal{A}_{sel}$, of a site-selection program $\Pi_{sel}$ in a selection base $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ consists of the following pairwise disjoint categories:*

- *the function-free alphabets $\mathcal{A}_{qd}$, $\mathcal{A}_{sd}$, and $\mathcal{A}_{dom}$ of the query description $\Pi_{qd}$, the site description $\Pi_{sd}$, and the domain theory $\Pi_{dom}$, respectively;*

- *the predicate $query\_site(S, Q)$, expressing that site $S$ is selected for evaluating query $Q$;*

- *the predicates* $default\_class(O,C,Q)$ *and* $default\_path(O,P,Q)$*, which are projections of the predicate* $access(O,C,P,Q)$*; and*
- *a set* $\mathcal{A}_{aux}$ *of auxiliary predicates;*

Informally, $default\_class(O,C,Q)$ and $default\_path(O,P,Q)$ serve to specify a default status for selection rules depending on context-reference pairs matched in the query $Q$. For example, a predicate $default\_class(O, \text{"}Person\text{"}, Q)$ in the body of rule $r$ says that $r$ is eligible, if the concept $Person$ occurs in the access path $O$ and there is no other rule $s$ that refers to some CRP $(C', P')$ matched in $Q$. These defaults are semantically realized using a suitable rule ordering.

Rules are constructed as in Section 3. We use upper-case letters for variables and lower-case letters for constants. We write $r(X_1,\ldots,X_n)$ to indicate that rule $r$ has variables $X_1,\ldots,X_n$. The set of all literals over atoms in $\mathcal{A}_\ell$, where $\ell \in \{qd, sd, dom, aux, sel\}$, is denoted by $Lit_\ell$.

**Definition 3** *A* site-selection program over $\mathcal{A}_{sel}$ *is a tuple* $(\Pi_{sel}, <_u)$*, where*

(*i*) $\Pi_{sel}$ *is a collection of rules over* $\mathcal{A}_{sel}$ *consisting of the following parts:*

(*a*) *the* core unit $\Pi^c_{sel}$ *containing rules of form*

$$query\_site(S,Q) \quad \leftarrow \quad L_1,\ldots,L_m, not\ L_{m+1},\ldots, not\ L_n;$$

(*b*) *a set* $\Pi^{aux}_{sel}$ *of* auxiliary rules *of form*

$$L_0 \quad \leftarrow \quad L_1,\ldots,L_m, not\ L_{m+1},\ldots, not\ L_n;$$

(*c*) *an* optimization part $\Pi^o_{sel}$ *containing weak constraints of form*

$$\Leftarrow\ L_1,\ldots,L_m, not\ L_{m+1},\ldots, not\ L_n\ [w:l],$$

*where* $L_0$ *is either a literal from* $Lit_{aux}$ *or is of form* $\neg query\_site(\cdot,\cdot)$*, and* $L_i \in Lit_{sel}$ *for* $1 \le i \le n$*, and* $w, l \ge 1$ *are integers.*

(*ii*) *The relation* $<_u$ *is a strict partial order between rules in* $\Pi^c_{sel}$*; the elements of* $<_u$ *are called* user defined preferences. *If* $r <_u s$*, then* $s$ *is said to have* preference over $r$.

The rules in the core unit $\Pi^c_{sel}$ serve for selecting a site, based on information from the domain description $\Pi_{dom}$, the site description $\Pi_{sd}$, the query description $R(Q) \cup \Pi_{qd}$, and possibly from auxiliary rules. The latter may be used, e.g., for evaluating complex conditions. By $<_u$, preference of site selection can be expressed. As well, the weak constraints in $\Pi^o_{sel}$ are used to filter answer sets under quantitative conditions.

**Example 8** Consider a simple site-selection program, $(\Pi_{sel}, <_u)$, for our movie domain:

$$\begin{array}{rrcl}
r_1: & query_site(s_2,Q) & \leftarrow & default_class(O, \text{"}Person\text{"}, Q). \\
r_2: & query_site(s_1,Q) & \leftarrow & selects(O, equal, \text{"}Hitchcock\text{"}), \\
 & & & access(O, \text{"}Director\text{"}, \text{"}Personalia/LastName\text{"}, Q). \\
r_3: & query_site(S,Q) & \leftarrow & default_path(O, \text{"}LastName\text{"}, Q), default_class(O,T,Q), \\
 & & & accurate(S,T,high). \\
r_4: & high_acc(T,Q) & \leftarrow & access(O,T,P,Q), accurate(S,T,high). \\
r_5: & high_cov(T,Q) & \leftarrow & access(O,T,P,Q), covers(S,T,high).
\end{array}$$

$c_1$: $\Leftarrow$ $query\_site(S, Q), high\_acc(T, Q), not\ accurate(S, T, high).$ [10 : 1]
$c_2$: $\Leftarrow$ $query\_site(S, Q), high\_cov(T, Q), not\ covers(S, T, high).$ [5 : 1]

$r_1(Q, \_) <_u r_3(Q, \_, \_, \_).$

Intuitively, $r_1$ advises to choose site $s_2$ if the query involves persons and no more specific rule is eligible. The rule $r_2$ says to choose site $s_1$ if the query contains an explicit select on the movie director Hitchcock. Rule $r_3$ demands to choose a site if, on some query access path, "*LastName*" is accessed under some concept $T$ (with arbitrary intermediate access path), and the site is highly accurate for $T$. Rules $r_4$ and $r_5$ define auxiliary predicates which hold on concepts $T$ appearing in the query such that some site with high accuracy and coverage, respectively, for $T$ exists. The weak constraints $c_1$ and $c_2$ state penalties for choosing a site that does not have high accuracy (weight 10) resp. coverage (weight 5) for a concept in the query while such a site exists. Finally, $r_1(Q, \_) <_u r_3(Q, \_, \_, \_)$ expresses preference of instances of $r_3$ over $r_1$ on the same query.

## 6.2 Semantics

The semantics of a site-selection program $(\Pi_{sel}, <_u)$ in a selection base $\mathcal{S} = \langle \Pi_{qd}, \Pi_{dom}, \Pi_{sd}, \Pi_{sel}, <_u \rangle$ on a query $Q$ is given by an *answer set* of $(\Pi_{sel}, <_u)$, which is defined as a preferred answer set of a prioritized ELP $\mathcal{E}(\mathcal{S}, Q) = (\Pi_Q, <)$ associated with $\mathcal{S}$ and $Q$. The program $\Pi_Q$ contains ground instances of rules and constraints in $\Pi_{sel}$, and further rules ensuring that a single site is selected per query and defining the default context predicates. The order relation $<$ is formed from the user preferences $<_u$ and the implicit priorities derived from context references in the core unit and from auxiliary rules. In that, we must suitably combine preference information and handle conflicts, which we do by a cautious conflict elimination policy.

We commence the formal details with the following notation: For any rule $r$, its *defaultization*, $r^{\Delta}$, is given by $H(r) \leftarrow B(r), not\ \neg H(r)$. We assume that user-defined preferences between rules carry over to their defaultizations.

**Definition 4** *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ be a selection base and $Q$ a query. Then, the program $\Pi_Q$ contains all ground instances of the rules and constraints in $\Pi^{aux}_{sel} \cup \Pi^{o}_{sel}$, as well as all ground instances of the following rules:*

(*i*) *the defaultization $r^{\Delta}$ of $r$, for each $r \in \Pi^{c}_{sel}$;*

(*ii*) *the* structural rule

$$\neg query\_site(S, Q) \quad \leftarrow \quad query\_site(S', Q), S \neq S'; \tag{2}$$

(*iii*) *the* default context rules

$$default\_class(O, C, Q) \quad \leftarrow \quad access(O, C, \_, Q); \tag{3}$$

$$default\_path(O, P, Q) \quad \leftarrow \quad access(O, \_, P, Q). \tag{4}$$

Intuitively, the defaultization makes the selection rules in $\Pi_{sel}$ defeasible with respect to $query\_site(\cdot, \cdot)$; the structural rule enforces that only one source is selected; and the default context rules define the two default predicates. Since our language has no function symbols, $\Pi_Q$ is finite, and its size depends on the constants appearing in $\Pi_{qd}$, $R(Q)$, $\Pi_{sd}$, and $\Pi_{dom}$.

For $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ and query $Q$, we call any answer set of $\Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom}$ a *selection input of* $\mathcal{S}$ *for* $Q$. The set of all selection inputs of $\mathcal{S}$ for $Q$ is denoted by $Sel(\mathcal{S}, Q)$. For $I \in Sel(\mathcal{S}, Q)$, we define

$$I_{def} = I \cup \{ \mathit{default\_class}(o, c, q),\ \mathit{default\_path}(o, p, q) \mid \mathit{access}(o, c, p, q) \in I \}.$$

Notice that, in general, a selection base may admit multiple selection inputs for a query $Q$. However, in many cases there may exist only a single selection input, in particular, if the site description $\Pi_{sd}$ and the domain knowledge $\Pi_{dom}$ have unique answer sets. In our framework, this is ensured if, e.g., these components are represented by stratified programs.

**Definition 5** *A rule* $r \in \Pi_Q$ *is* relevant for $Q$ *iff there is some* $I \in Sel(\mathcal{S}, Q)$ *such that* $B^{\dagger}(r)$ *is true in* $I_{def}$*, where* $B^{\dagger}(r)$ *results from* $B(r)$ *by deleting each element which does not contain a predicate symbol from* $\mathcal{A}_{qd} \cup \mathcal{A}_{sd} \cup \mathcal{A}_{dom} \cup \{\mathit{default\_class}, \mathit{default\_path}\}$.

In the sequel, we denote for any binary relation $R$ its transitive closure by $R^*$.

We continue with the construction of the preference relation $<$, used for interpreting a site-selection program $(\Pi_{sel}, <_u)$ relative to a selection base $\mathcal{S}$ and a query $Q$, in terms of an associated prioritized logic program $(\Pi_Q, <)$.

Informally, the specification of $<$ depends on the following auxiliary relations:

- the preference relation $\preceq_c$, taking care of implicit context priorities;
- the intermediate relation $\trianglelefteq$, representing a direct combination of user-defined preferences with context preferences; and
- the preference relation $<'$, removing possible conflicts within the joined relation $\trianglelefteq$ and ensuring transitivity of the resultant order $<$.

More specifically, the relation $\preceq_c$ is the first step towards $<$, transforming structural context information into explicit preferences, in virtue of the following specificity conditions:

- relative to the same IRP and query, rules that mention default concepts or a CRP matching the query are considered more specific than rules mentioning default paths;
- default contexts for concepts are assumed to be more specific than default contexts for attributes; and
- with respect to the same IRP, rules with a larger CRP $(C, P)$ are considered more specific than rules with a shorter CRP $(C', P')$ (i.e., $P'$ is a subpath of $P$).

The second step in the construction of $<$ is the relation $\trianglelefteq$, which is just the union of the user preferences $<_u$ and the context priorities $\preceq_c$. In general, this will not be a strict partial order. To enforce irreflexivity, we remove all priorities $n_r \trianglelefteq n_s$ which lie on a cycle $n_r \trianglelefteq n_s \trianglelefteq n_{s_1} \trianglelefteq \cdots \trianglelefteq n_{s_n} \trianglelefteq n_r$, resulting in $<'$. Finally, taking the transitive closure of $<'$ yields $<$. The formal definition of relation $<$ is as follows.

**Definition 6** *Let* $\mathcal{S}$ *be a selection base,* $Q$ *a query, and* $\Pi_Q$ *as in Definition 4. We introduce the following binary relations over the rules in* $\Pi_Q$*: For* $r, s \in \Pi_Q$*,*

1. $r \preceq_c s$ *iff* $r, s$ *are relevant for* $Q$*,* $r \neq s$*, and one of* $(O_1)$*–*$(O_3)$ *holds:*

   $(O_1)$ $\mathit{default\_path}(o_1, p_1, q) \in B(r)$*, and we have either* $\mathit{default\_class}(o_2, t_2, q) \in B(s)$ *or it holds that* $\mathit{access}(o_2, t_2, p_2, q) \in B(s)$*;*

   $(O_2)$ $\mathit{default\_class}(o_1, t_1, q) \in B(r)$ *and* $\mathit{access}(o_2, t_2, p_2, q) \in B(s)$*;*

   $(O_3)$ $\mathit{access}(o, t_1, p_1, q) \in B(r)$*,* $\mathit{access}(o, t_2, p_2, q) \in B(s)$*, and* $t_1/p_1$ *is a subpath of* $t_2/p_2$*;*

2. $r \trianglelefteq s$ *iff* $r <_u s$ *and* $r, s$ *are relevant for* $Q$*, or* $r \preceq_c s$*;*

3. $r <' s$ *iff* $r \trianglelefteq s$ *but not* $s \trianglelefteq^* r$*.*

*Then, the relation* $<$ *is given as the transitive closure of relation* $<'$*.*

**Example 9** Reconsider $(\Pi_{sel}, <_u)$ from Example 8. Suppose the domain ontology contains the concepts "*MovieDB*", "*Actor*", "*Movie*", "*Director*", and "*Person*", and that "*Personalia*" and "*Person*" are synonymous. Assume further that the query $Q$ of Example 1 has a unique selection input $I$, containing the following site description facts:

$$\mathit{accurate}(s_1, \text{“}\mathit{Director}\text{”}, \mathit{high});\;\; \mathit{covers}(s_2, \text{“}\mathit{Person}\text{”}, \mathit{high});\;\; \mathit{reliable}(s_3, \mathit{low}),$$

plus the following facts resulting from the query description and the default context rules:

$\mathit{default\_path}(o_3, \text{“}\mathit{LastName}\text{”}, q_1);$
$\mathit{default\_class}(o_2, \text{“}\mathit{Person}\text{”}, q_1);$
$\mathit{default\_class}(o_3, \text{“}\mathit{Person}\text{”}, q_1);$
$\mathit{default\_class}(o_3, \text{“}\mathit{Director}\text{”}, q_1);$
$\mathit{selects}(o_3, \mathit{equal}, \text{“}\mathit{Hitchcock}\text{”});$
$\mathit{access}(o_2, \text{“}\mathit{Person}\text{”}, \text{“}\mathit{FirstName}\text{”}, q_1);$
$\mathit{access}(o_2, \text{“}\mathit{Director}\text{”}, \text{“}\mathit{Personalia/FirstName}\text{”}, q_1);$
$\mathit{access}(o_3, \text{“}\mathit{Person}\text{”}, \text{“}\mathit{LastName}\text{”}, q_1);$
$\mathit{access}(o_3, \text{“}\mathit{Director}\text{”}, \text{“}\mathit{Personalia/LastName}\text{”}, q_1).$

These elements are exactly those contributing to relevant instances of $\Pi_Q$. The relevant instances of $r_1$, $r_2$, and $r_3$ are given by the ground rules $r_1(q_1, o_2)$, $r_2(q_1, o_3)$, $r_2(q_1, o_3)$, and $r_3(q_1, s_1, o_3, \text{“}D\text{”})$ (for brevity, we write here "$D$" for "$\mathit{Director}$"). Intuitively, we expect $r_2(q_1, o_3)$ to have highest priority among these rule instances, since the bodies of the instances of $r_1$ and $r_3$ contain default predicates while $r_2$ references a specific context. Actually, the order relation $<$ includes for the relevant instances of $r_1$, $r_2$, and $r_3$ the following pairs: $r_1(q_1, o_2) < r_2(q_1, o_3)$, $r_1(q_1, o_3) < r_2(q_1, o_3)$, and $r_3(q_1, s_1, o_3, \text{“}D\text{”}) < r_2(q_1, o_3)$.

Note that both $r_4$ and $r_5$ have two relevant instances. However, they do not influence the above rule ordering. Informally, they are either unrelated to or "ranked between" $r_2(q_1, o_3)$ and the relevant instances of $r_1$ and $r_3$ (since the $\mathit{access}$ predicates of $r_4$ and $r_5$ refer to the same context as the context referenced in the body of $r_2$, or to a subpath of such a context). Hence, the relevant instance of $r_2$ has highest priority.

As for $r_1$ and $r_3$, the auxiliary relation $\trianglelefteq$ contains two further structural priorities, $r_3(q_1, s_1, o_3, \text{“}D\text{”}) \preceq_c r_1(q_1, o_2)$ and $r_3(q_1, s_1, o_3, \text{“}D\text{”}) \preceq_c r_1(q_1, o_3)$. They are in conflict with the user preferences $r_1(q_1, o_2) <_u r_3(q_1, s_1, o_3, \text{“}D\text{”})$ and $r_1(q_1, o_3) <_u r_3(q_1, s_1, o_3, \text{“}D\text{”})$, respectively. This is resolved in the resultant relation $<$ by removing these preferences.

Note that in Definition 6, the final order $<$ enforces a cautious conflict resolution strategy, in the sense that it remains "agnostic" with respect to priority information causing conflicts. Alternative definitions of $<'$, such as removal of a minimal cutset eliminating all cycles in $\trianglelefteq$, may be considered; however, this may lead to a nondeterministic choice since, in general, multiple such cutsets exist. Different choices lead to different orders $<$, which may lead to different results of the site selection program. Thus, unless a well-defined *particular* minimal cutset is singled out, by virtue of preference conflicts the result of the site selection process might not be deterministic. Furthermore, an extended logic program component computing a final order based on minimal cutsets is more involved than a component computing the relations in Definition 6.

Combining Definitions 4 and 6, we obtain the translation $\mathcal{E}(\cdot, \cdot)$ as follows:

**Definition 7** *Let $\mathcal{S}$ be a selection base and $Q$ a query. Then, the evaluation $\mathcal{E}(\mathcal{S}, Q)$ of $\mathcal{S}$ with respect to $Q$ is given by the prioritized logic program $(\Pi_Q, <)$, where $\Pi_Q$ and $<$ are as in Definitions 4 and 6, respectively.*

Answer sets of site-selection programs are then obtained as follows:

**Definition 8** *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ be a selection base, and let $\mathcal{E}(\mathcal{S}, Q) = (\Pi_Q, <)$ for query $Q$. Then, $S \subseteq Lit_{sel}$ is an* answer set of $(\Pi_{sel}, <_u)$ for $Q$ with respect to *$\mathcal{S}$ iff $S$ is a preferred answer set of the prioritized logic program $(\Pi_Q \cup I, <)$ for some $I \in Sel(\mathcal{S}, Q)$. A site $s$ is* selected for $Q$ *iff $query\_site(s, q)$ belongs to some answer set of $(\Pi_{sel}, <_u)$ for $Q$ (with respect to $\mathcal{S}$), where the constant $q$ represents $Q$.*

**Example 10** In our running example, $(\Pi_{sel}, <_u)$ has a unique answer set $S$, with respect to $\mathcal{S}$, for the query $Q$ (represented by $q_1$) from Example 8. It contains $query\_site(s_1, q_1)$, which is derived from the core rule $r_2(q_1, o_3)$, having the highest priority among the applicable rules leading to a single preferred answer set for the weak-constraint free part of $\Pi_{sel}$. If we replace, e.g., $r_1$ by the rule $query\_site(s_2, Q) \leftarrow access(O, \text{"}Person\text{"}, P, Q)$ and adapt the corresponding user preference to $r_1(Q, \_, \_) <_u r_3(Q, \_, \_, \_)$, then the weak-constraint free part of $\Pi_{sel}$ has two preferred answer sets: one, $S_1$, is identical to $S$ (where applying $r_2(q_1, o_3)$ is preferred to applying $r_1(q_1, o_3)$, given that $r_1(q_1, o_3) < r_2(q_1, o_3)$); in the other answer set, $S_2$, the rule $r_1(q_1, o_2)$ is applied and $query\_site(s_2, q_1)$ is derived. Informally, the replacement removes the preference of $r_2(q_1, o_3)$ over $r_1(q_1, o_2)$, since their $access$ predicates refer to different contexts ("$.../FirstName$" and "$.../LastName$", respectively). Consequently, $r_1(q_1, o_2)$ has maximal preference like $r_2(q_1, o_3)$.

Given that $S_1$ has weight 5, caused by violation of $c_2(s_1, q_1, \text{"}Person\text{"})$, but $S_2$ has weight 10, caused by violation of $c_1(s_2, q_1, \text{"}Director\text{"})$, $S_1$ is the answer set of $(\Pi_{sel}, <_u)$ for $Q$.

## 6.3 Properties

Finally, we discuss some properties of our framework. To this end, we introduce the following concept: Given logic programs $\Pi_1$ and $\Pi_2$, we say that $\Pi_1$ is *independent* of $\Pi_2$, if each predicate symbol occurring in some rule head of $\Pi_2$ does not occur in $\Pi_1$. Intuitively, if $\Pi_1$ is independent of $\Pi_2$, then $\Pi_1$ may serve as an "input" for $\Pi_2$. This idea is made precise by the following proposition, which is an immediate consequence of results in [22, 32]:

**Proposition 1** *Let $\Pi_1$ and $\Pi_2$ be two extended logic programs, possibly containing weak constraints, and let $S$ be a set of ground literals. If $\Pi_1$ is independent of $\Pi_2$, then $S$ is an answer set of $\Pi_1 \cup \Pi_2$ iff there is some answer set $I$ of $\Pi_1$ such that $S$ is an answer set of $\Pi_2 \cup I$.*

Now, taking the specific structure of our site-selection architecture into account, we obtain the following characterization, which links the evaluation method of site-selection programs to the usual semantics of prioritized logic programs.

**Theorem 1** *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{dom}, \Pi_{sd}, \Pi_{sel}, <_u \rangle$ be a selection base, let $Q$ be a query, and let $\mathcal{E}(\mathcal{S},Q) = (\Pi_Q, <)$. Then, $S$ is an answer set of $(\Pi_{sel}, <_u)$ for $Q$ with respect to $\mathcal{S}$ iff $S$ is a preferred answer set of $(\Pi_Q \cup \Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom}, <)$.*

**Proof**. Let $\Pi_0$ denote the program $\Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom}$. Recall that $S$ is an answer set of $(\Pi_{sel}, <_u)$ for $Q$ with respect to $\mathcal{S}$ iff $S$ is a preferred answer set of $(\Pi_Q \cup I, <)$, for some answer set $I$ of $\Pi_0$. Since the predicate symbols occurring in the heads of rules in a site-selection program do not occur in rules from the query description, the site description, or the domain theory, we obviously have that $\Pi_0$ is independent of $\Pi_Q$. Moreover, it holds that $S$ is a preferred answer set of $(\Pi_Q \cup I, <)$ only if $S$ is an answer set of $\Pi_Q \cup I$. Hence, applying Proposition 1, we have that $S$ is an answer set of $\Pi_Q \cup I$ for some answer set $I$ of $\Pi_0$ iff $S$ is an answer set of $\Pi_Q \cup \Pi_0$. From this, the assertion of the theorem is an immediate consequence. □

Strengthening Theorem 1, the construction of $\mathcal{E}(\mathcal{S}, Q)$ can itself be realized in terms of a *single* logic program, by describing preference relations *directly at the object level*. For this purpose, we introduce a set $N$ of constants serving as names for rules, and a new binary predicate $pref(\cdot,\cdot)$, defined over $N$, expressing preference between rules. The extended alphabet $\mathcal{A}_{sel} \cup \{pref(\cdot,\cdot)\} \cup N$ is denoted by $\bar{\mathcal{A}}_{sel}$. We furthermore assume an injective function $n(\cdot)$ which assigns to each rule $r \in \Pi_Q$ a name $n(r) \in N$. To ease notation, we also write $n_r$ instead of $n(r)$. Finally, $Lit_{pref}$ denotes the set of literals having predicate $pref$. Note that $Lit_{pref} \cap Lit_{sel} = \emptyset$.

**Theorem 2** *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{dom}, \Pi_{sd}, \Pi_{sel}, <_u \rangle$ be a selection base, let $Q$ be a query, and let $\mathcal{E}(\mathcal{S}, Q) = (\Pi_Q, <)$. Furthermore, let $\Pi_{\mathcal{S}} = \Pi_{qd} \cup R(Q) \cup \Pi_{dom} \cup \Pi_{sd} \cup \Pi_Q$. Then, there exists a logic program $\Pi_{obj}$ over an alphabet $\hat{\mathcal{A}} \supseteq \bar{\mathcal{A}}_{sel}$ such that every answer set $S$ of $\Pi_{\mathcal{S}} \cup \Pi_{obj}$ satisfies the following conditions:*

1. *$S \cap Lit_{pref}$ represents $<$, i.e., $pref(n_r, n_{r'}) \in S$ iff $r < r'$; and*
2. *$S \cap Lit_{sel}$ is an answer set of $(\Pi_{sel}, <_u)$ for $Q$ with respect to $\mathcal{S}$ iff $S$ is a preferred answer set of the prioritized program $(\Pi_{\mathcal{S}} \cup \Pi_{obj}, <)$.*

**Proof**. We describe $\Pi_{obj}$ here, but omit the detailed argument that it works properly. Informally, $\Pi_{obj}$ has two parts: the first, $\Pi_{rel}$, is derived from $\Pi_{\mathcal{S}}$ and takes care of computing the relevant rules for $Q$, by utilizing weak constraints; the second part, $\Pi_{pref}$, is a stratified logic program and determines the relations in Definition 6. We start with the construction of $\Pi_{rel}$.

For each predicate $p \in \mathcal{A}_{sel}$ and each $r \in \Pi_Q$, we introduce a new predicate $\overline{p}_r$ of the same arity as $p$. In addition, we introduce a new atom $rel_r$, informally expressing that rule $r$ is relevant. If $\alpha$ is either a literal, a set of literals, a rule, or a program, then $\lceil \alpha \rceil_r$ denotes the result of uniformly replacing each atom $p(x_1, , \ldots, x_n)$ occurring in $\alpha$ by $\overline{p}_r(x_1, \ldots, x_n)$.

For each $r \in \Pi_Q$, we define a program $\Pi_r$ containing the following items:

(i) each rule in $\lceil \Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom} \rceil_r$;

(ii) the rule $rel_r \leftarrow \lceil B^\dagger(r) \rceil_r$; and

(iii) the extended default context rules

$$\begin{array}{rcl}\overline{default\_class}_r(O,C,Q) & \leftarrow & \overline{access}_r(O,C,\_,Q);\\ \overline{default\_path}_r(O,P,Q) & \leftarrow & \overline{access}_r(O,\_,P,Q).\end{array}$$

As easily checked, $r$ is relevant for $Q$ iff $\Pi_r$ has some answer set containing $rel_r$. Now, $\Pi_{rel}$ is defined as the collection of each of the programs $\Pi_r$, together with weak constraints of form

$$\Leftarrow\ not\ rel_r\ [1 : m+1], \tag{5}$$

for every $r \in \Pi_Q$, where $m$ is the maximal priority level of the weak constraints occurring in $\Pi^o_{sel}$. Since, for any $r, s \in \Pi_Q$ with $r \neq s$, the programs $\Pi_r$ and $\Pi_s$ are over disjoint alphabets, and given the inclusion of the weak constraints (5) in $\Pi_{rel}$, we obtain that $\Pi_{rel}$ satisfies:

($*$) for every answer set $S$ of $\Pi_{rel}$ and every $r \in \Pi_Q$, $r$ is relevant for $Q$ iff $rel_r \in S$.

These answer sets are used as inputs for the program $\Pi_{pref}$, which is defined next.

Let $pr(n,m)$ and $pr'(n,m)$ be new binary predicates, where $n, m$ are names. Then,

$$\begin{array}{rcl}\Pi_{pref} & = & \{\ pr(n_r,n_s) \ \leftarrow\ rel_r, rel_s \mid r,s \in \Pi_Q \text{ and either } r <_u s \text{ or}\\ & & \qquad r,s \text{ satisfy Condition } (O_1) \text{ or } (O_2) \text{ of Definition 6 } \}\\ & \cup & \{\ pr(n_r,n_s) \ \leftarrow\ rel_r, rel_s, subpath(o,t_1,p_1,t_2,p_2) \mid r,s \in \Pi_Q,\\ & & \qquad access(o,t_1,p_1,q) \in B(r), access(o,t_2,p_2,q) \in B(s)\ \}\\ & \cup & \{\ pr'(N_1,N_2) \ \leftarrow\ pr(N_1,N_2);\\ & & \quad pr'(N_1,N_3) \ \leftarrow\ pr'(N_1,N_2), pr(N_2,N_3);\\ & & \quad pref(N_1,N_2) \ \leftarrow\ pr(N_1,N_2), not\ pr'(N_2,N_1);\\ & & \quad pref(N_1,N_3) \ \leftarrow\ pref(N_1,N_2), pref(N_2,N_3)\ \}.\end{array}$$

Obviously, $\Pi_{pref}$ is a stratified program. Moreover, in view of Condition ($*$), and since $\Pi_{rel}$ is independent of $\Pi_{pref}$ and $\Pi_\mathcal{S}$ is independent of $\Pi_{obj}$, for every answer set $S$ of $\Pi_\mathcal{S} \cup \Pi_{obj} = \Pi_\mathcal{S} \cup \Pi_{rel} \cup \Pi_{pref}$, we have that (i) $pr(n_r,n_s) \in S$ iff $r \trianglelefteq s$; (ii) $pr'(n_r,n_s) \in S$ iff $r \trianglelefteq^* s$; and (iii) $pref(n_r,n_s) \in S$ iff $r < s$. This shows Condition 1 of the theorem.

As for Condition 2, consider some answer set $S$ of $\Pi_\mathcal{S} \cup \Pi_{obj}$. Since $\Pi_\mathcal{S}$ is independent of $\Pi_{obj}$, $S$ is of form $I \cup J$, where $I$ is an answer set of $\Pi_\mathcal{S}$ and $J$ is a set of ground literals disjoint from $Lit_{sel}$. Hence, $S \cap Lit_{sel} = I$. According to Theorem 1, $I$ is an answer set of $(\Pi_{sel}, <_u)$ for $Q$ with respect to $\mathcal{S}$ iff $I$ is a preferred answer set of $(\Pi_\mathcal{S}, <)$. But it is easily seen that the latter holds just in case $I \cup J$ is a preferred answer set of $(\Pi_\mathcal{S} \cup \Pi_{obj}, <)$. This proves the result. □

We remark that $\Pi_{rel}$ can be simplified by taking into account independence of subprograms of $\Pi_\mathcal{S}$ and unique answer sets for them in case. For example, if $\Pi_{sd}$ has a unique answer set, then we might use in each program $\Pi_r$ simply $\lceil \alpha \rceil_r = \alpha$ for each literal on $\mathcal{A}_{sd}$. In particular, if the program $\Pi_{qd} \cup R(Q) \cup \Pi_{dom} \cup \Pi_{sd}$ has a unique answer set (e.g., if this program is stratified), then we might simply take as $\Pi_{rel}$ this program plus all rules $rel_r \leftarrow B^\dagger(r)$, for $r \in \Pi_Q$.

The program $\Pi_\mathcal{S} \cup \Pi_{obj}$ in Theorem 2 thus represents, via preferred answer sets for a dynamic rule preference given by the atoms over $pref$, the answer sets of $(\Pi_{sel}, <_u)$ for $Q$. It can be easily adapted to a fixed program $\Pi'_{\mathcal{S}}$ such that for any query $Q$, the dynamic preferred answer sets of $\Pi'_{\mathcal{S}} \cup R(Q)$ represent the answer sets of $(\Pi_{sel}, <_u)$ for $Q$.

One of the desiderata of our approach is that each answer set selects a unique source, for any query $Q$. The following result states that this property is fulfilled.

**Theorem 3** *Let $S$ be an answer set of $(\Pi_{sel}, <_u)$ for query $Q$ with respect to $\mathcal{S}$. Then, for any constant $q$ it holds that $|\{s \mid query\_site(s, q) \in S\}| \leq 1$.*

**Proof**. The presence of the structural rule (2) in the evaluation program $\Pi_Q$ enforces that, whenever $S$ contains two ground atoms of form $query\_site(s, q)$ and $query\_site(s', q)$, $S$ must be inconsistent, and thus $S$ violates the consistency criterion of answer sets. □

Lastly, the following result concerns the order of application of site-selection rules, stating that site selection is blocked in terms of priorities as desired.

**Theorem 4** *Let $S$ be an answer set of $(\Pi_{sel}, <_u)$ for query $Q$ with respect to $\mathcal{S}$, and let $r^\Delta \in \Pi_Q$ be the defaultization of some rule $r$ belonging to the grounding of $\Pi^c_{sel}$ for $Q$ with respect to $\mathcal{S}$.*

*Suppose that $B(r)$ is true in $S$ but $H(r) \notin S$. Then, there is some $s \in \Pi_Q$ such that*

1. *either $s$ belongs to the grounding of $\Pi^{aux}_{sel}$ for $Q$ with respect to $\mathcal{S}$ and $H(s) = \neg H(r)$, or $s$ is the defaultization of a rule from the grounding of $\Pi^c_{sel}$ for $Q$ with respect to $\mathcal{S}$;*
2. *$B(s)$ and $H(s)$ are true in $S$; and*
3. *either $r^\Delta$ and $s$ are incompatible with respect to $<$, or else $r^\Delta < s$ holds.*

**Proof**. Given that $S$ is an answer set of $(\Pi_{sel}, <_u)$ for query $Q$ with respect to $\mathcal{S}$, we have that $S$ is a preferred answer set of the prioritized logic program $(\Pi_Q \cup I, <)$, where $I$ is some selection input of $\mathcal{S}$ for query $Q$, and, *a fortiori*, that $S$ is an answer set of $\Pi_Q \cup I$. From the latter and $H(r) \notin S$, it follows that $r^\Delta = H(r) \leftarrow B(r), not\, \neg H(r)$ is not a member of $GR(\Pi, S)$. Hence, in view of the assumption that $B(r)$ is true in $S$, we get that $\neg H(r) \in S$ must hold.

Since $S$ is a preferred answer set of $(\Pi_Q \cup I, <)$, there is some enumeration $\langle r_i \rangle_{i \in I}$ of $GR(\Pi_Q \cup I, S)$ such that Conditions $(P_1)$ – $(P_3)$ hold (cf. Section 3). We take $s = r_\ell$, where $\ell$ is as follows. Given that $\neg H(r) \in S$, there is some smallest index $i_0 \in I$ such that $r_{i_0} \in GR(\Pi_Q \cup I, S)$ and $H(r_{i_0}) = \neg H(r)$. If $r_{i_0}$ belongs to the grounding of $\Pi^{aux}_{sel}$ for $Q$ w.r.t. $\mathcal{S}$, then $\ell = i_0$. Otherwise, by the syntactic form of a site-selection program, $r_{i_0}$ must be an instance of the structural rule (2). By $(P_1)$ – $(P_3)$, the defaultization $r'$ of a rule from the grounding of $\Pi^c_{sel}$ for $Q$ w.r.t. $\mathcal{S}$ must exist such that $r' = r_{i'_0} \in GR(\Pi_Q \cup I, S)$ and $i'_0 < i_0$. In this case, $\ell = i'_0$.

We show that $r_\ell$ satisfies Conditions 1–3. Clearly, (1) is satisfied. Furthermore, (2) is an immediate consequence of the fact that $r_\ell \in GR(\Pi_Q \cup I, S)$. It remains to show that (3) holds.

Towards a contradiction, assume that $r_\ell < r^\Delta$. Since $r_\ell \in GR(\Pi, S)$ and $r^\Delta \notin GR(\Pi, S)$, from Condition $(P_3)$ we get that $B^-(r^\Delta) \cap \{H(r_k) \mid k < \ell\} \neq \emptyset$, as $B^+(r) \subseteq S$ and $B^+(r^\Delta) = B^+(r)$. Now, obviously $\{H(r_k) \mid k < \ell\} \subseteq S$. Moreover, since $B^-(r) \cap S = \emptyset$ and $B^-(r^\Delta) = B^-(r) \cup \{\neg H(r)\}$, we obtain that $\neg H(r) \in \{H(r_k) \mid k < \ell\}$. Hence, there must be some $k_0 < \ell$ such that $r_\ell \in GR(\Pi, S)$ and $H(r_\ell) = \neg H(r)$. But this contradicts the condition that $i_0$ ($\geq \ell$) is the smallest index $i$ such that $r_i \in GR(\Pi, S)$ and $H(r_i) = \neg H(r)$ holds. Hence, we either have that $r_\ell$ and $r^\Delta$ are incompatible with respect to $<$, or $r^\Delta < r_\ell$ must hold. □

# 7 Implementation and Application

## 7.1 Implementation

We have implemented a site-selection capability as described in Section 6, on top of the `DLV` system [19] and its front end `plp` [14] for prioritized logic programs. The evaluation of site-selection programs proceeds in three steps: First, the set of all selection inputs for a query $Q$ is computed from $R(Q)$, $\Pi_{qd}$, $\Pi_{sd}$, and $\Pi_{dom}$, using `DLV` (cf. Section 6). A second call to `DLV` calculates the priority relation, $<$, from the set of selection inputs and $\Pi_{sel}$, and – third – the answer sets of $(\Pi_Q, <)$ are determined by employing `plp` and `DLV`.

The entire process is implemented as an Eclipse Prolog program and is independent of the actual query language. For XML-QL queries, however, a query parser, written in C++, for generating the low-level representation $R(Q)$ of a query $Q$ has been developed. A query parser for SQL queries will be soon available. Furthermore, we have "agentized" the site-selection system in the *IMPACT* agent platform [43], such that site-selection agents can be built which also may issue the execution of XML-QL queries on XML-data sources.

## 7.2 Application: Movie Databases

As an application domain, we considered movie databases, and we have built an experimental environment for site selection in this domain, using the prototype implementation of the site selection described above.

### 7.2.1 Movie Sites

We used the *Internet Movie Database (IMDB)* [4] as a main source for raw data, and besides the *EachMovie Database* provided by Compaq Equipment Corporation [1], to generate the suite of XML movie databases below. To this end, (parts of) the large databases were wrapped offline to XML, using a DTD (provided in Appendix A) which we modeled from a set of movie relevant concepts captured by the *Open Directory Project* [3].

- **RandomMovies (RM):** Contains numerous movies randomly wrapped from the IMDB. Each item comprises, besides title and language (always value "English") – where available – genre, release date, running time, reviews, the two main actors, directors, screenwriters, as well as corresponding soundtrack (some with composer). Persons are recorded by name.

- **RandomPersons (RP):** Contains data of numerous actors, directors, screenwriters, and some composers randomly wrapped from the IMDB. Person data comprises – besides names – birth date, country, and a biography, and may again be incomplete, of course.

- **EachMovie (EM):** Wrapped from Compaq's EachMovie database, this site stores movies in English plus ratings. For most, it has genre information, and for half of them a release date (after 1995). It has no information on actors, directors, soundtracks, etc.

- **Hitchcock (HC):** Wrapped from the IMDB, this site stores all movies directed by Alfred Hitchcock, in the format of RandomMovies but with all involved actors listed. For each person, it also has information (if available) on birth date, country, and biography.

- **KellyGrant (KG):** Similar as HC, this site stores all movies in which either Grace Kelly or Cary Grant did act in, as well as all persons involved.
- **Horror60 (H60):** A collection of horror movies from the 1960's, found in the IMDB. Movie and person data are as before, but almost no soundtracks and composers are stored.

Since the complete formal descriptions of these sites, using the predicates in Section 5 is by far too extensive (see Appendix E), we merely consider one site here.

**Example 11** The description $\Pi_{sd}$ of site KellyGrant contains the following facts:

$site(s\_KellyGrant).\quad up(s\_KellyGrant).$
$data\_format(s\_KellyGrant, xml).$
$update\_frequency(s\_KellyGrant, low).$
$specialized(s\_KellyGrant, \text{"}Kelly\text{"}).$
$specialized(s\_KellyGrant, \text{"}Grant\text{"}).$
$covers(s\_KellyGrant, \text{"}Movie'\text{"}, low).$
$covers(s\_KellyGrant, c, high).$ for $c \in \{\text{"}ReleaseDate\text{"}, \text{"}Person\text{"}, \text{"}BirthDate\text{"}, \text{"}Actor\text{"}\}$,
$covers(s\_KellyGrant, fifties, high).$
$covers(s\_KellyGrant, sixties, high).$
$\neg relevant(s\_KellyGrant, p).$ for $p \in \{seventies, eighties, nineties, twothousands\}$.

Informally, KellyGrant is an XML site which is (currently) up and rarely updated. It is specialized on topics *Kelly* and *Grant*. It has high coverage of persons, especially actors, and their birth dates, while it lowly covers movies in general. However, it highly covers the release dates of the movies stored, most of which are from the fifties and sixties.

Further information on KellyGrant is derived from default rules like the one for site language given below, which states that English is the default language for all sites:

$$\begin{array}{lrcl} sd_1: & site\_language(S, \text{"}English\text{"}) & \leftarrow & site(S), not\ \neg site\_language(S, \text{"}English\text{"}). \\ sd_2: & \neg site\_language(S, \text{"}English\text{"}) & \leftarrow & site\_language(S, L), \neq (L, \text{"}English\text{"}). \end{array}$$

### 7.2.2 Domain Knowledge

The domain knowledge, $\Pi_{dom}$, consists of an *ontology* part and *background knowledge*. The former contains the relevant concepts of the application domain and their interrelationships. The ontological knowledge might be generated semi-automatically, e.g., from an underlying XML DTD, or it may stem from an external ontological reasoning capability, which is interfaced via this part of the domain knowledge. Domain models inferred from large scale ontologies could be automatically accessed by means of a (syntactic) translation of their formal representation into logic programming facts. For example, OIL [12] based ontologies could be accessed via the FaCT system [2], and ontologies built on Ontolingua [37] could be interfaced by translating corresponding models represented and communicated by KIF statements. Similarly, one could also interface OpenCyc's (`http://www.opencyc.org`) free version of CYC via a translation of its formal representation language CycL. The concrete technical realization of such interfaces is an interesting topic for further extensions.

For our experiments, we used a simple ontology by representing the concepts of the XML DTD as classes with appropriate attributes in a hierarchy using an *is-a* relation, together with simple ontological reasoning.

**Example 12** The concept `Movie` is captured by the fact $class(\text{“}Movie\text{”})$ and its attributes by the facts $has\_att(\text{“}Movie\text{”}, att)$, for $att \in \{title, alternativeTitles, genre, runningTime, releaseDate, review, language\}$ . Some rules for ontological reasoning are the standard inheritance rules:

$$
\begin{array}{lrcl}
s_1\colon & instance(I, C_2) & \leftarrow & instance(I, C_1), is\_a(C_1, C_2).\\
s_2\colon & has\_att(C_2, A) & \leftarrow & is\_a(C_1, C_2), has\_att(C_1, A).
\end{array}
$$

The ontology part of $\Pi_{dom}$ also serves to define concepts and possible synonyms for the query abstraction in Section 4. Using the XML DTD, we may extract the concepts given by the facts $class0(O)$, for $O \in \{\text{“}MovieDB\text{”}, \text{“}Movie\text{”}, \text{“}Director\text{”}, \text{“}Actor\text{”}, \text{“}Screenwriter\text{”}, \text{“}Composer\text{”}, \text{“}Person\text{”}, \text{“}Review\text{”}, \text{“}Soundtrack\text{”}\}$, and $synonym(\text{“}Personalia\text{”}, \text{“}Person\text{”})$.

The background part of $\Pi_{dom}$ serves to formalize ‘common sense’ knowledge of the application domain, which is an important source of information for the selection process process. This part is usually quite extensive. On the one hand, it has rules capturing typical relationships between ontological concepts and, on the other hand, it comprises ‘well-known’ instances of these concepts. For space reasons, we only show a few rules of $\Pi_{dom}$ here. Note however, that it also implements temporal reasoning over decades, by relating every year since 1920 to its decade.

**Example 13** Some (typical) rules from the background knowledge are:

$$
\begin{array}{lrcl}
s_3\colon & involved(P, M) & \leftarrow & directed(P, M).\\
s_4\colon & instance(P, \text{“}Director\text{”}) & \leftarrow & directed(P, M).\\
s_5\colon & life\_period(P, B, E) & \leftarrow & instance(P, \text{“}Person\text{”}), has\_att(P, birthDate, B1),\\
 & & & not\ dead(P), calender\_year(B1, B), current\_year(E).\\
s_6\colon & possible\_genre(M, G) & \leftarrow & involved(P, M), default\_genre(P, G), not\ defined\_genre(M).
\end{array}
$$

Intuitively, rules like $s_1$ and $s_2$ infer from roles *acted*, *directed*, etc between a person and a movie, that the corresponding person is an actor, director, etc and that she is involved in the movie. Rule $s_3$ assigns a life period to a person from her birth date, while rule $s_4$ infers a possible genre for a movie, if an involved person and her default genre are known.

The following facts are representations of well-known instances (actually, Grace Kelly and the movie ‘Arsenic and Old Lace’):

$instance(perKelly, \text{“}Actor\text{”})$.
$has\_att(perKelly, name, nameKelly)$.
$has\_att(perKelly, birthDate, 1929)$.
$has\_att(perKelly, dateOfDeath, 1982)$.
$prod\_period(perKelly, 1945, 1960)$.

$instance(nameKelly, name)$.
$has\_att(nameKelly, firstName, \text{“}Grace\text{”})$.
$has\_att(nameKelly, firstName, \text{“}Patricia\text{”})$.
$has\_att(nameKelly, lastName, \text{“}Kelly\text{”})$.

$instance(m12, \text{“}Movie\text{”})$.
$has\_att(m12, title, \text{“}Arsenic\ and\ Old\ Lace\text{”})$.
$has\_att(m12, releaseDate, 1944)$.

$acted(perGrant, m12)$.

### 7.2.3 Site-Selection Program

The experimental movie site selection program fills several pages, and is too complex to be listed and discussed here in detail (see Appendix F). We give here an informal description and highlight the most important aspects.

Among all site-selection rules, default rules have lowest priority, and are used only in the core. They make default suggestions for query sites in case no other core site-selection rule is eligible. Some examples are the following default rules:

$$
\begin{array}{lrcl}
r_1: & query\_site(S,Q) & \leftarrow & default\_path(O,P,Q),\, occurs(O,V),\\
& & & specialized(S,P).\\
r_{20}: & query\_site(s\_RandomMovies,Q) & \leftarrow & default\_class(O,\text{“}Movie\text{”},Q).\\
r_{21}: & query\_site(s\_RandomPersons,Q) & \leftarrow & default\_class(O,\text{“}Person\text{”},Q).
\end{array}
$$

The first rule, named $r_1$ by the atom $r_1(S,Q,O,P,V)$ in the body, is a generic rule, while the others are specific ones. Informally, $r_1$ advises to query site $S$, if it is specialized for $P$, where $P$ is some path of an access in the query that is compared to some value. For example, suppose $P$ is instantiated with “*LastName*”; if some site is specialized for last names, then it is chosen unless a site-selection rule with higher priority applies. Similarly, the specific rules suggest to select RandomMovies or RandomPersons if the query entails an access under object “*Movie*” or “*Person*”, respectively.

Non-default core site-selection rules also appear as generic and specific rules:

$$
\begin{array}{lrcl}
r_{40}: & query\_site(S,Q) & \leftarrow & site(S),\, query(Q),\, high\_coverage(S,Q).\\
r_{41}: & query\_site(S,Q) & \leftarrow & site(S),\, query(Q),\, special(S,Q).\\
r_{61}: & query\_site(s\_Hitchcock,Q) & \leftarrow & access(O_1,\text{“}P\text{”},\text{“}LN\text{”},Q),\, selects(O_1,equal,\text{“}Hitchcock\text{”}),\\
& & & access(O_2,\text{“}P\text{”},\text{“}FN\text{”},Q),\, selects(O_2,equal,\text{“}Alfred\text{”}).
\end{array}
$$

The generic rules $r_{40}$ and $r_{41}$ suggest to query any site that highly covers the query or is special for it, respectively. The specific rule $r_{61}$ advises to query the Hitchcock site if a query selects a person named Alfred Hitchcock. Note that $high\_coverage$ and $special$ are *auxiliary* predicates, defined by auxiliary rules (see below).

Note hat since no $access$ predicate and no default predicates occur in the bodies of rules $r_{40}$ and $r_{41}$, there is no (direct) structural precedence between them and rule $r_{61}$ as well as the above default rules. The following user preferences explicitly establish preferences among them ($r_i < r_j$ means $r_j$ is preferred over $r_i$):

$r_1(\_,Q,\_,\_,\_) < r_{40}(\_,Q).$
$r_1(\_,Q,\_,\_,\_) < r_{41}(\_,Q).$
$r_{20}(Q,\_) < r_{41}(\_,Q).$
$r_{21}(Q,\_) < r_{41}(\_,Q).$
$r_{40}(\_,Q) < r_{41}(\_,Q).$
$r_{40}(\_,Q) < r_{61}(Q,\_,\_).$
$r_{41}(\_,Q) < r_{61}(Q,\_,\_).$

Auxiliary rules are used to define auxiliary predicate as well as to filter irrelevant sites:

$$
\begin{array}{lrcl}
a_1\text{:} & special(S,Q) & \leftarrow & special\_topic(S,Q,T). \\
a_2\text{:} & special\_topic(S,Q,T) & \leftarrow & inferred\_topic(Q,T),\ specialized(S,T). \\
a_3\text{:} & inferred\_topic(S,Q,T) & \leftarrow & matchingMovie(Q,M),\ involved(P,M), \\
 & & & has\_att(P,name,N),\ has\_att(N,lastName,T). \\
a_4\text{:} & \neg query\_site(S,Q) & \leftarrow & irrelevant(S,Q). \\
r_{92}\text{:} & irrelevant(S,Q) & \leftarrow & access(O,\text{“}MDB\text{”},\text{“}M/RD/D\text{”},selects(O,equal,V), \\
 & & & calender\_year(V,Y),\ decade(Y,D),\ \neg relevant(S,D).
\end{array}
$$

(For technical reasons, rules that do not contribute to the preference order in the implementation have $true$ in place of a rule name in the body.) Informally, rule $a_1$ states that a site is special for a query, if a topic associated with the query exists for which it is special. Rule $a_2$ suggests that one way to associate a topic to a query is to infer a topic, e.g., by rule $a_3$. Hence, if the query accesses a movie that is known and $T$ is the last name of a person involved in it, then a site is concluded to be special for that query if it is specialized for $T$. Rule $a_4$ states that a site must not be queried if it is irrelevant for a query; by rule $r_{92}$, this is the case if the site is not relevant for the decade of a movie's release date.

Finally, the quantitative part of the site-selection program has weak constraints like:

$$
\begin{array}{lll}
w_1\text{:} & \Leftarrow & query\_site(S,Q), default\_class(O,T,Q), \\
 & & constructs(O,C,P), covers(S_1,T,high),\ not\ covers(S,T,high). \quad [3{:}1] \\
w_2\text{:} & \Leftarrow & query\_site(S,Q),\ high\_covered\_topic(S_1,Q,T), \\
 & & decade\_name(T),\ not\ high\_covered\_topic(S,Q,T). \quad [1{:}1]
\end{array}
$$

Intuitively, $w_1$ assigns a penalty of 3 per concept $T$ that is asked (resp. constructed) by the query, to any answer set which selects a site that does not highly cover $T$ while a site highly covering $T$ exists. Similarly, $w_2$ assigns a penalty of 1 per decade that is associated to the query, to any answer set which selects a site which does not highly cover this decade while some other highly covering site exists.

## 8 Experiments

For testing the approach on the movie application, we used a number of natural user queries listed below. Their formulation in XML-QL is straightforward; for $Q_1$ and $Q_2$ it appears in Example 1 and 7, respectively, while for the others it is given in Appendix G.

$Q_1$ : *Which movies were directed by Alfred Hitchcock?*

$Q_2$ : *In which movies directed by Josef von Sternberg acted Marlene Dietrich?*

$Q_3$ : *In which year the movie 'Arsenic and Old Lace' has been released?*

$Q_4$ : *In which movies directed by Alfred Hitchcock acted Marlene Dietrich?*

$Q_5$ : *In which film noirs did Marilyn Monroe act?*

$Q_6$ : *Which movies did Laurel and Hardy act in 1940?*

$Q_7$ : *Which movies acted by Frank Sinatra have a soundtrack by Elmer Bernstein?*

| *Query* | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ | $Q_8$ | $Q_9$ |
|---|---|---|---|---|---|---|---|---|---|
| *Candidates* | HC | RM, HC, H60, KG | KG | HC | RM, HC, KG | RM, HC, KG | RM, HC, H60, KG | RP | RM, HC, H60, KG |
| *Best* | HC | RM | KG | HC | RM | RM | RM | RP | RM |

| *Query* | $Q_{10}$ | $Q_{11}$ | $Q_{12}$ | $Q_{13}$ | $Q_{14}$ | $Q_{15}$ | $Q_{16}$ | $Q_{17}$ | $Q_{18}$ |
|---|---|---|---|---|---|---|---|---|---|
| *Candidates* | KG | RM, HC, KG | RM, HC, H60, KG | H60 | RM, HC, KG | RM, HC, KG | RP | RM, HC, H60, KG | EM |
| *Best* | KG | RM | RM | H60 | RM | RM | RP | RM | EM |

Table 1: Experimental Results for Original Selection Base

$Q_8$ : *When was James Dean born?*

$Q_9$ : *In which movies did Jane Russell and Marilyn Monroe act together?*

$Q_{10}$ : *In which movies did Cary Grant and Marilyn Monroe act together?*

$Q_{11}$ : *In which comedies did John Wayne act?*

$Q_{12}$ : *Which movies acted by Sean Connery have a soundtrack by John Barry?*

$Q_{13}$ : *In which horror movies of the sixties did Leonard Nimoy act?*

$Q_{14}$ : *In which animation movies did Vincent Price act?*

$Q_{15}$ : *In which movies, except crime movies, did James Cagney act between 1930 and 1940?*

$Q_{16}$ : *Where is Giorgio Moroder from?*

$Q_{17}$ : *In which year the movie 'Witness for the Prosecution' has been released?*

$Q_{18}$ : *Which movies from 1999 have been rated above 90 points (out of 100)?*

Site selection for these queries was performed for the suite of movie databases from above, as well as for modifications. Each process took from few seconds up to some tens of seconds, which is due to the large logic program to be processed. However, performance was not an issue; since our prototype implementation and the tools used are unoptimized, there is large potential for runtime improvements.

## 8.1 Results

The results of site selection for $Q_1$–$Q_{18}$ using the site descriptions from above are shown in Table 1. Note that, by the semantics of site-selection programs, per answer set and query, a single site is chosen. Thus, query decomposition is not considered here, even if our method of query abstraction would, in principle, allow for it. The entries show the sites which are selected for querying by different answer sets, where "Candidates" and "Best" mean selection with optimization part dropped (i.e., qualitative selection) and enabled (optimization), respectively.

The results can be informally explained as follows. For $Q_1$, a specific core site-selection rule, $r_{61}$, which has highest preference, fires and HC is chosen, as expected.

For $Q_2$, there is some background knowledge about Marlene Dietrich, but no site can be found special for this query, while generic default site-selection rules trigger for all sites. Nonetheless, RP and EM are recognized as irrelevant for $Q_2$ and discarded: $Q_2$ asks for resp. ranges over concepts which these sites are not relevant for (“*Movie*” and “*Person*”, respectively). The best site of the candidates RM, H60, HC, KG is RM, since it is the only one highly covering the concept asked for, i.e. movies.

For $Q_3$, since ‘Arsenic and Old Lace’ is in the background knowledge (cf. above), as well as that Cary Grant acted in it, we would expect KG to be queried since it is specialized for Cary Grant. This is what actually happens. It is not a specific core site-selection rule that triggers, because Cary Grant does not explicitly appear in the query, but Grant is inferred as a query topic from the background knowledge and, thus, the generic core site selection rule suggesting to query KG has highest priority.

$Q_4$ is a syntactic variant of $Q_1$; the same specific core site-selection rule, $r_{61}$, triggers.

Similar as for $Q_2$, RM is chosen for $Q_5$–$Q_7$, but for $Q_5$ and $Q_6$, H60 is recognized as irrelevant on different grounds. $Q_5$ asks for film noirs, and so H60, which contains horror movies, is eliminated by reasoning over genre information. $Q_6$ asks for movies of 1940, and thus H60, which contains only movies of the 1960’s, is excluded by temporal reasoning.

RP is chosen for $Q_8$, as expected: A specific default site-selection rule triggers for RP, which has precedence over generic default rules that would trigger for other sites.

Queries $Q_9$ and $Q_{10}$ are similar to query $Q_2$. While $Q_9$ yields the same candidates and result as $Q_2$, this does not hold for $Q_10$, which explicitly asks for movies acted by Cary Grant, hence, triggering a specific core site-selection rule for KG.

$Q_{11}$, $Q_{12}$, $Q_{14}$, and $Q_{15}$ are variants of queries $Q_5$–$Q_7$, which is also reflected in respective results. For another syntactic variant of these queries, query $Q_{13}$, however, the outcome differs. It explicitly asks for horror movies of the sixties, for which a special site, H60, as well as a specific core site-selection rule suggesting this site exist.

RP is chosen for $Q_{16}$ for the same reasons as it is chosen for $Q_8$. $Q_{17}$ is a variation of query $Q_3$, and, again, the movie asked for, namely ‘Witness for the Prosecution’, is an instance of the background knowledge (cf. Appendix D). This time however, background reasoning does not infer a special site for the query, yielding RM to be the best site among the candidates because of its high coverage of movies.

Finally, EM is the site selected for $Q_{18}$, since a specific core site-selection rule, concerning queries asking for movie ratings, has highest priority.

## 8.2 Results with Modified Selection Bases

In a slightly different scenario, RM highly covers composers and cowboy/western movies, too, and a new random movie site *RandomMoviesNew (RMN)* similar to RM, but with less coverage of genres, release dates, composers, and cowboy/western movies, but with high coverage of directors, dramas, and comedies, is introduced. Respective changes to the site descriptions and addition of a specific default site-selection rule for RMN (similar to rule $r_{20}$ for RM in Section 7.2.3) and according user preferences to the site-selection program yield an “extended” selection base, on which the results for $Q_1$–$Q_{18}$ are shown in Table 2.

The change has no influence on the results of the site-selection process for $Q_1$, $Q_3$, $Q_4$, $Q_8$, $Q_{10}$, $Q_{13}$, $Q_{16}$, and $Q_{18}$; this is intuitive, since the change does not affect the suitability of the sites chosen for these queries. For the other queries, the new site RMN is an additional candidate, since the generic default site-selection rule is also applicable to RMN. By an analogous argument as before, RM and RMN are better than the other candidates. For $Q_2$, $Q_9$, and $Q_{17}$ RMN is ranked before RM: it highly covers both, dramas and comedies, which are inferred topics for these queries, since in the background knowledge, drama is a default

| *Query* | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ | $Q_8$ | $Q_9$ |
|---|---|---|---|---|---|---|---|---|---|
| *Candidates* | HC | RM, HC, H60, KG, RMN | KG | HC | RM, HC, KG, RMN | RM, HC, KG, RMN | RM, HC, H60, KG, RMN | RP | RM, HC, H60, KG, RMN |
| *Best* | HC | RMN | KG | HC | RM, RMN | RM, RMN | RM | RP | RMN |

| *Query* | $Q_{10}$ | $Q_{11}$ | $Q_{12}$ | $Q_{13}$ | $Q_{14}$ | $Q_{15}$ | $Q_{16}$ | $Q_{17}$ | $Q_{18}$ |
|---|---|---|---|---|---|---|---|---|---|
| *Candidates* | KG | RM, HC, KG, RMN | RM, HC, H60, KG, RMN | H60 | RM, HC, KG, RMN | RM, HC, KG, RMN | RP | RM, HC, H60, KG, RMN | EM |
| *Best* | KG | RMN | RM | H60 | RM, RMN | RM, RMN | RP | RMN | EM |

Table 2: Experimental Results for "Extended" Selection Base

genre for Marlene Dietrich (and Marlene Dietrich acted in 'Witness for the Prosecution') and comedy is a default genre for Marilyn Monroe. RMN is also ranked before RM for $Q_{11}$, since its high coverage of comedies.

RM is ranked before RMN for $Q_7$ and $Q_{12}$ since RM highly covers composers, a concept occurring in the queries (which ask for composers Elmer Bernstein and John Barry, respectively). RM and RMN are ranked equal for $Q_5$, $Q_6$, $Q_{14}$, and $Q_{15}$: they highly cover actors (and are equally relevant for the thirties), but the background knowledge has no information about Laurel and Hardy, or James Cagey; and the facts, that Marilyn Monroe's default genre is drama and that Vincent Price's default genre is horror, have no consequences for $Q_5$ and $Q_{14}$, since they explicitly ask for film noirs and animation movies, respectively.

An alternative change yields a "reduced" selection base where RM is down and, thus, cannot be queried. The query results are given in Table 3. The candidate sites remain the same, except that RM is missing; thus, the change has no impact on $Q_1$, $Q_3$, $Q_4$, $Q_8$, $Q_{10}$, $Q_{13}$, $Q_{16}$, and $Q_{18}$, as one would expect. For $Q_6$ and $Q_{15}$, the optimization part does not impose preference between HC and KG; interestingly, it selects KG as best for $Q_2$, $Q_5$, $Q_7$, $Q_9$, $Q_{11}$, $Q_{12}$, $Q_{14}$, and $Q_{17}$. This is because the background knowledge entails information about the productive period of the persons Dietrich, Monroe, Bernstein, Wayne, Connery, Barry, and Price, respectively, occurring in the queries. Thus, the 1950's and 1960's are inferred to be relevant topics of these queries and KG, covering both decades highly, outperforms HC and H60, which highly cover only one of the decades each.

# 9 Related Work and Conclusion

Selection of data sources is an ingredient to many information integration systems [6, 7, 11, 24, 26, 28, 31]. However, most center around mappings between a global scheme and local schemes, on query rewriting, and on query planning to optimally reconstruct dispersed information. We focused on *qualitative* selection from different alternatives, based on rich meta-knowledge, which is not an issue there.

More related is [27], which outlines an interactive tool for information specialists in query design. It relieves them from searching through data source specifications and can suggest sources to determine trade-offs. However, no formal semantics or richer domain theories, capable of handling incomplete and default information, is presented.

| *Query* | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ | $Q_8$ | $Q_9$ |
|---|---|---|---|---|---|---|---|---|---|
| *Candidates* | HC | H60, HC, KG | KG | HC | HC, KG | HC, KG | H60, HC, KG | RP | H60, HC, KG |
| *Best* | HC | KG | KG | HC | KG | HC, KG | KG | RP | KG |

| *Query* | $Q_{10}$ | $Q_{11}$ | $Q_{12}$ | $Q_{13}$ | $Q_{14}$ | $Q_{15}$ | $Q_{16}$ | $Q_{17}$ | $Q_{18}$ |
|---|---|---|---|---|---|---|---|---|---|
| *Candidates* | KG | HC, KG | H60, HC, KG | H60 | HC, KG | HC, KG | RP | H60, HC, KG | EM |
| *Best* | KG | KG | KG | H60 | KG | HC, KG | RP | KG | EM |

Table 3: Experimental Results for "Reduced" Selection Base

Remotely related to our work is [23], which presents a decision-theoretic model for selecting data sources based on retrieval cost and typical IR parameters.

Goto et al. [42] consider a problem setting related to ours, where source descriptions include semantic knowledge about the source. They develop a source selection method based on a user's query and these source descriptions. In contrast to our work, however, their representations are solely term based. A query is viewed as a set of terms, and a source description is a thesaurus automatically constructed from the documents of the source. A further thesaurus, WordNet, is used for the source evaluation algorithm, which is based on the calculation of weighted similarity measures. The main differences to our approach are that the selection method is not declarative and just numeric, that semantic knowledge is limited to a thesaurus, and that no further background knowledge, reasoning or semantic query analysis is involved.

Semantic analysis of queries has been incorporated to document retrieval in [18]. Starting from conventional IR methods that accept natural language queries against text collections and calculate similarity measures for query keywords, semantic modeling was introduced by trying to detect entity attributes and thematic roles from the query to the effect of a modified similarity computation. While richer ontological knowledge than thesauri is used, source descriptions have no semantic knowledge. Again, the approach is not declarative and numeric in nature, and neither rich domain theories nor automated reasoning is involved.

Another step towards richer ontologies is [34]. The authors describe an Expert Locator System developed for Boing, that uses a thesaurus as a conceptual vocabulary for knowledge-based search. A technical thesaurus is used as an initial source for conceptual vocabulary and domain knowledge. In combination with AI techniques of search, subsumption computation, and language processing it serves as a ontology for the application domain. With system assistance a user can (re-)formulate a query in terms of concepts, which are then matched against manually generated concept indices characterizing target resources.

Chiang et al. [39] follow a different approach. Instead of using a general ontology, their smart Web query engine performs context sensitive web queries by employing particular domain ontologies, so-called *context ontologies*. The context of a Web query, i.e., a context ontology, is determined by a comparison of search terms provided by the user. The most relevant context ontology is then used for expanding query terms and for result document filtering.

An early attempt towards knowledge-based information retrieval is [36]. It introduces FAQ FINDER, a natural language question-answering system that uses files of frequently asked questions as its knowledge base. FAQ FINDER uses standard information retrieval technology to narrow the search to one particular FAQ file and to calculate a term-vector metric for the user's question and every question/answer pair. In addition to term vectors, a comparison of question type – according to a taxonomy defined by the syntactic expression of a query – and a semantic similarity score are used during the question matching process.

The semantic similarity metric is calculated by passing through the hypernym links, i.e., isa links, through WordNet.

More recent proposals for Web-based information retrieval build on ontology-based information agents which search for, maintain, and mediate relevant information on behalf of their users or other agents [40, 30, 45]. Sim and Wong [30] have developed a society of software agents: Query processing agents which assist users in selecting Web-pages, information filtering agents for actual information retrieval, and information monitoring agents to report changes. Query processing agents search for URLs using conventional search engines. They use ontological relations from WordNet for query specialization or generalization in order to keep the number of located, relevant URLs within user specified limits.

Chen and Soo [45] propose an architecture for ontology-based information gathering agents that also use general search engines, but special domain search engines and Web-documents as well. They use a traditional object-oriented language to represent ontologies. Queries are represented as partial instances of ontological concepts. Classes of the ontology have procedure attachments for actually performing searches. A planning operation, based on general priority criteria (e.g. consider documents before specific search engines before general search engines) and heuristics, identifies the best procedure attachment for the agent's next step.

Like – or in addition to – ontologies, knowledge-based methods are mainly used for query expansion in information retrieval. Early work by Bodner and Song [38], for example, discusses a conventional (statistical) information retrieval system based on the vector space model. Artificial intelligence methods are employed after the query has been indexed but before it is compared to the document collection. AI techniques employed comprise methods for searching and combining knowledge bases (general world and domain specific) organized as semantic networks.

Hunter [5] argued for using non-monotonic logics, default logic in particular, as a richer alternative to probabilistic approaches for handling semantic information about keywords. Once semantic information is represented as a set of default rules – obtained by a training process – reasoning, called *activity positioning*, can be employed to determine whether a *keyphrase* (a conjunction of literals representing a query) is too general, or too specialized and whether it should be expanded, reduced, or replaced by another keyphrase at all.

More recently, DOLORES [33], a fully logic-based system for retrieving multimedia objects, has been developed. DOLORES integrates logic-based information retrieval and classical retrieval algorithms and data structures by means of a layered approach. The logical layer is based on POOL, a probabilistic object-oriented logic used to represent and query structured hypermedia documents (including multimedia objects). On the physical level, the contents of documents are stored explicitly by probabilistic indexing facts. POOL programs and queries are translated to probabilistic Datalog, which is used as inference engine. While DOLORES allows for hypermedia retrieval from a large database consisting of a huge number of hypermedia documents, the concern of our work was to select one out of a number of information sites to answer a query. In the DOLORES system the contents of a database, i.e., the data, is represented and can be queried by a probabilistic object-oriented logic. Hence, complete knowledge of the data is required in contrast to our approach, where we use extended logic programs to describe information sources without the need for a full knowledge of their contents. Thus, DOLORES relies on appropriate document indexing methods, most of which are currently restricted to the syntactical level of multimedia objects, while we build on semantic knowledge about information sources, which we combine with ontological knowledge in order to reason about the sources.

### 9.1 Conclusion and further work

We have presented a knowledge-based approach to information site selection, which uses answer set programming in order to provide a fully declarative and flexible formalism for expressing site selection policies. In particular, it is capable of handling incomplete information and allows for using default rules, and enables reasoning tasks which involve different components such as background and ontological knowledge, site descriptions, and query constituents. To this end, we have developed methods for automated query analysis and abstraction, and for preference handling in site selection, which combines implicit rule priorities, given by the context of rule application, and explicit user preferences. Different from other work, our approach has a clear, model-theoretic semantics, which makes it more accessible to reason about its behavior.

The results that we have obtained in the implementation of the experimental movie application are encouraging, and suggest several subjects for further work in order to make the approach broadly available. The current setting of single site selection can be generalized to ranked site selection, in which the $k$ given best sites are computed for a given $k$. This can be easily accomplished by computing ranked models in the underlying `DLV` system. Another issue concerns the supply of rich background and common sense knowledge. The coupling with available ontology and common sense engines via suitable interfaces (cf. Section 7.2.2) is suggestive to this end. Further issues are to provide adaptive source selection, and to combine it with query rewriting and query planning, which is subject of future research.

## References

[1] Compaq EachMovie Database. `http://www.research.compaq.com/SRC/eachmovie/`.

[2] Fast Classification of Terminologies. `http://www.cs.man.ac.uk/~horrocks/FaCT`.

[3] Open Directory Project. `http://dmoz.org`.

[4] The Internet Movie Database. `http://imdb.com`.

[5] Anthony Hunter. Using Default Logic in Information Retrieval. In Christine Froidevaux and Jurg Kohlas, editor, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 235–242. Springer, 1995.

[6] Y. Arens, C. Chee, C. Hsu, and C. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *Int. J. of Cooperative Information Systems*, 2(2):127–158, 1993.

[7] R. Bayardo, B. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Semantic Integration of Information in Open and Dynamic Environments (Experience Paper). In *SIGMOD Conf. 1997*, pages 195–206, 1997.

[8] G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, 109(1–2):297–356, 1999.

[9] F. Buccafurri, N. Leone, and P. Rullo. Stable Models and their Computation for Logic Programming with Inheritance and True Negation. *J. of Logic Programming*, 27(1):5–43, 1996.

[10] F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Trans. on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[11] P. Cannata, M. Huhns, N. Jacobs, T. Ksiezyk, K. Ong, A. Sheth, M. Singh, C. Tomlinson, and D. Woelk. The Carnot Heterogeneous Database Project: Implemented Applications. *Distributed and Parallel Databases*, 5(2):207–225, 1997.

[12] D. Fensel and F. van Harmelen and I. Horrocks and D. McGuinness and P.F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001. `http://www.computer.org/intelligent/ex2001/x2toc.htm`.

[13] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proc. 15th Int. Joint Conf. on Artificial Intelligence* (*IJCAI'97*), volume 1, pages 578–583. Morgan Kaufmann, 1997.

[14] J. Delgrande, T. Schaub, and H. Tompits. plp: A Generic Compiler for Ordered Logic Programs. In T. Eiter, W. Faber, and M. Truszczyński, editors, *Proc. 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'01*), pages 411–415. Springer, 2001.

[15] J. Delgrande, T. Schaub, and H. Tompits. A Framework for Compiling Preferences in Logic Programs. *Theory and Practice of Logic Programming*, 2002.

[16] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proc. 8th International World Wide Web Conference (WWW8)*, 1999. Computer Networks 31(11-16): 1155-1169. See also `http://www.w3.org/TR/NOTE-xml-ql/`.

[17] Y. Dimopoulos and A. Kakas. Information Integration and Computational Logic. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net*, 2000. URL: http://www.compulog.org/net/Forum/Supportdocs.html.

[18] Edgar B. Wendlandt and James R. Driscoll. Incorporating a Semantic Analysis into a Document Retrieval Strategy. In Abraham Bookstein and Yves Chiaramella and Gerard Salton and Vijay V. Raghavan, editor, *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Chicago, Illinois, USA, October 13-16, 1991 (Special Issue of the SIGIR Forum)*, pages 270–279. ACM, 1991.

[19] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.

[20] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Using Methods of Declarative Logic Programming for Intelligent Information Agents. *Theory and Practice of Logic Programming*, 2002. To appear. Available from `http://xxx.lanl.gov/abs/cs.MA/0108008`.

[21] T. Eiter, M. Fink, and H. Tompits. A Knowledge-Based Approach to Information Site Selection. Technical Report INFSYS RR-1843-02-08, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, June 2002. Forthcoming.

[22] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[23] N. Fuhr. A Decision-Theoretic Approach to Database Selection in Networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.

[24] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *J. of Intelligent Information Systems*, 8(2):117–132, 1997.

[25] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3–4):365–386, 1991.

[26] M. Genesereth, A. Keller, and O. Duschka. Infomaster: An Information Integration System. In *Proc. ACM SIGMOD Conference*, volume 26(2) of *SIGMOD Record*, pages 539–542, New York, 1997. ACM Press.

[27] S. B. Huffman and D. Steier. A Navigation Assistant for Data Source Selection and Integration. In *Working Notes of AAAI-95 Fall Symposium Series on AI Applications in Knowledge Navigation and Retrieval, Cambridge, MA*, pages 72–77, 1995.

[28] M. Huhns and M. Singh. The Semantic Integration of Information Models. In *AAAI Workshop on Cooperation among Heterogeneous Intelligent Agents*, 1992.

[29] K. Inoue and C. Sakama. Prioritized Logic Programming and Its Applications to Commonsense Reasoning. *Artificial Intelligence*, 123(1–2):185–222, 2000.

[30] Kwang Mong Sim and Pui Tak Wong. Web-Based Information Retrieval Using Agent and Ontology. In N. Zhong et al., editor, *In Proceedings of the First Asia-Pacific Conference on Web Intelligence, WI 2001, Maebashi City, Japan*, volume 2198 of *LNAI*, pages 384–388. Springer, 2001.

[31] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In T. Vijayaraman, A. Buchmann, C. Mohan, and N. Sarda, editors, *Proc. 22th Int. Conf. on Very Large Data Bases* (*VLDB'96*), pages 251–262. Morgan Kaufmann, 1996.

[32] V. Lifschitz and H. Turner. Splitting a Logic Program. In *Proceedings ICLP-94*, pages 23–38, Santa Margherita Ligure, Italy, June 1994. MIT-Press.

[33] Norbert Fuhr and Norbert Govert and Thomas Rolleke. DOLORES: A System for Logic-Based Retrieval of Multimedia Objects. In *Research and Development in Information Retrieval*, pages 257–265, 1998.

[34] P. Clarke and J. Thompson and H. Holmback and L. Duncan. Exploiting a Thesaurus-Based Semantic Net for Knowledge-Based Search. In *In Proceedings of the 12th Conference on Innovative Applications of AI (AAAI/IAAI"00)*, pages 988–995, 2000.

[35] A. Provetti and T. C. Son, editors. *Proceedings AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning, Stanford, CA (Workshop Technical Report SS-01-01)*. AAAI Press, March 2001. ISBN 1-57735-129-0.

[36] R. Burke and K. Hammond and J. Kozlovsky. Knowledge-Based Information Retrieval from Semi-Structured Text. In *Working Notes of AAAI-95 Fall Symposium, Series on AI Applications in Knowlege Navigation and Retrieval, Cambridge, MASemantic Ana*, pages 15–19, 1995.

[37] R. Fikes and A. Farquhar and J. Rice. Tools for Assembling Modular Ontologies in Ontolingua, March 1997. `http://www.ksl.stanford.edu/software/ontolingua/project-papers.html`.

[38] Richard C. Bodner and Fei Song. Knowledge-Based Approaches to Query Expansion in Information Retrieval. In *Canadian Conference on AI*, pages 146–158, 1996.

[39] Roger H.L. Chiang and Cecil E.H. Chua and Veda C. Storey. A smart web query method for semantic retrieval of web data. *Data and Knowledge Engineering*, 38:63–84, 2001.

[40] S. Luke and L. Spector and D. Rager and J. Hendler. Ontology-based Web Agents. In W.L. Johnson, editor, *In Proceedings of the First International Conference on Autonomous Agents (AutonomousAgents97), New York, Association for Computing Machinery*, pages 59–66, 1997.

[41] F. Sadri and F. Toni. Computational Logic and Multi-Agent Systems: a Roadmap. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net*, 2000. Avaliable from `http://www.compulog.org/net/Forum/Supportdocs.html`.

[42] Shoji Goto and Tadachika Ozono and Toramatsu Shintani. A Method for Information Source Selection using Thesaurus for Distributed Information Retrieval. In *Proceedings of the Pacific Asian Conference on Intelligent Systems 2001 (PAIS 2001)*, pages 272–277, November 2001.

[43] V. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross. *Heterogeneous Agent Systems: Theory and Implementation*. MIT Press, 2000. 640 pages.

[44] G. Wiederhold. Intelligent Integration of Information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 434–437, Washington, DC, 1993.

[45] Yi-Jia Chen and Von-Wun Soo. Ontology-Based Information Gathering Agents. In N. Zhong et al., editor, *In Proceedings of the First Asia-Pacific Conference on Web Intelligence, WI 2001, Maebashi City, Japan*, volume 2198 of *LNAI*, pages 423–427. Springer, 2001.

# A Appendix: XML DTD for the Movie Databases

```
<!ELEMENT MovieDB (Movie|Actor|Director|Screenwriter|Composer|Person|Award|
                   Filmfestival)*>
<!ELEMENT Movie (Title,AlternativeTitle*,ReleaseDate?,RunningTime?,Culture?,
                 LeadingRole*,Role*,Actor*,Director*,Screenwriter*,Soundtrack*,
                 Review*,Award*)>
<!ATTLIST Movie
          Genre (Action|Animation|Classic|Comedy| CowboyWestern|CultMovie|
                 Documentary|Experimental|FilmNoir|Horror|Romance|SciFiFantasy|
                 Series|Silent|Travel|Other)
          #IMPLIED Language CDATA "English">
<!ELEMENT Person (FirstName*,LastName,BirthDate?,Country?,Biography?)>
<!ATTLIST Person ID ID #REQUIRED Gender (male|female) #IMPLIED>
<!ELEMENT Award (AwardTitle,Date,AwardType?,AwardCategory?)>
<!ELEMENT Character (#PCDATA)>
<!ELEMENT Filmfestival (#PCDATA)>
<!ELEMENT Actor (Award*)>
<!ATTLIST Actor Personalia IDREF #REQUIRED>
<!ELEMENT Director (Award*)>
<!ATTLIST Director Personalia IDREF #REQUIRED>
<!ELEMENT Screenwriter (Award*)>
<!ATTLIST Screenwriter Personalia IDREF #REQUIRED>
<!ELEMENT Composer (Award*)>
<!ATTLIST Composer Personalia IDREF #REQUIRED>
<!ELEMENT Soundtrack (Title,Composer*,Award*)>
<!ELEMENT Biography (#PCDATA)>
<!ELEMENT AlternativeTitle (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT BirthDate (Date)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT AwardCategory (#PCDATA)>
<!ELEMENT AwardType (#PCDATA)>
<!ELEMENT AwardTitle (#PCDATA)>
<!ELEMENT ReleaseDate (Date)>
<!ELEMENT RunningTime (#PCDATA)>
<!ELEMENT LeadingRole (Character,Award*)>
<!ATTLIST LeadingRole Actor IDREF #REQUIRED>
<!ELEMENT Role (Character,Award*)>
<!ATTLIST Role Actor IDREF #REQUIRED>
<!ELEMENT Review (ReviewText,Rating?)>
<!ELEMENT ReviewText (#PCDATA)>
<!ELEMENT Rating (#PCDATA)>
<!ELEMENT Culture (#PCDATA)>
```

# B Appendix: Low-level Query-Representation Examples

The low-level representation $R(Q)$ of the query in Example 1, comprises the following set of facts:

$$
\begin{array}{l}
\{\ db\_name(\text{“}MovieDB\text{”}),\ query\_cand(q_1),\ sub\_query(q_2, q_1),\ source(\text{“}MovieDB\text{”}, q_2),\\
\quad whereRef(o_1, \text{“}Title\text{”}, \text{“ ”}, q_2),\ whereRef(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_2),\\
\quad whereRef(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_2),\\
\quad subpath(o_1, \text{“}Title\text{”}, \text{“ ”}, \text{“}Movie\text{”}, \text{“}Title\text{”}),\\
\quad subpath(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}),\\
\quad whereRef(o_2, \text{“}FirstName\text{”}, \text{“ ”}, q_2),\\
\quad whereRef(o_2, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}, q_2),\\
\quad whereRef(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}, q_2),\\
\quad whereRef(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}, q_2),\\
\quad whereRef(o_2, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}, q_2),\\
\quad subpath(o_2, \text{“}FirstName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}),\\
\quad subpath(o_2, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}),\\
\quad subpath(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”},\\
\qquad\qquad \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}),\\
\quad subpath(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”},\\
\qquad\qquad \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}),\\
\quad whereRef(o_3, \text{“}LastName\text{”}, \text{“ ”}, q_2),\\
\quad whereRef(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_2),\\
\quad whereRef(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_2),\\
\quad whereRef(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_2),\\
\quad whereRef(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_2),\\
\quad subpath(o_3, \text{“}LastName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}LastName\text{”}),\\
\quad subpath(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}),\\
\quad subpath(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”},\\
\qquad\qquad \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}),\\
\quad subpath(o_3, \text{”}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”},\\
\qquad\qquad \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}),\\
\quad whereCmp(o_2, equal, \text{“}Alfred\text{”}),\ whereCmp(o_3, equal, \text{“}Hitchcock\text{”}),\\
\quad consRef(o_1, \text{“}Movie\text{”}, \text{“ ”}),\ consRef(o_1, \text{“}MovieList\text{”}, \text{“}Movie\text{”})\}
\end{array}
$$

The low-level representation $R(Q)$ of the query in Example 7, comprises the following set of facts:

$$
\begin{array}{l}
\{\ db\_name(\text{“}MovieDB\text{”}),\ query\_cand(q_1),\ sub\_query(q_2, q_1),\ source(\text{“}MovieDB\text{”}, q_2),\\
\quad whereRef(o_1, \text{“}Title\text{”}, \text{“ ”}, q_2),\ whereRef(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_2),\\
\quad whereRef(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_2),\\
\quad subpath(o_1, \text{“}Title\text{”}, \text{“ ”}, \text{“}Movie\text{”}, \text{“}Title\text{”}),\\
\quad subpath(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}),\\
\quad whereRef(o_2, \text{“}FirstName\text{”}, \text{“ ”}, q_2),\\
\quad whereRef(o_2, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}, q_2),
\end{array}
$$

*whereRef*($o_2$, “*Director*”, “*Personalia/FirstName*”, $q_2$),
*whereRef*($o_2$, “*Movie*”, “*Director/Personalia/FirstName*”, $q_2$),
*whereRef*($o_2$, “*MovieDB*”, “*Movie/Director/Personalia/FirstName*”, $q_2$),
*subpath*($o_2$, “*FirstName*”, “ ”, “*Personalia*”, “*FirstName*”),
*subpath*($o_2$, “*Personalia*”, “*FirstName*”, “*Director*”, “*Personalia/FirstName*”),
*subpath*($o_2$,“*Director*”, “*Personalia/FirstName*”,
“*Movie*”, “*Director/Personalia/FirstName*”),
*subpath*($o_2$, “*Movie*”, “*Director/Personalia/FirstName*”,
“*MovieDB*”, “*Movie/Director/Personalia/FirstName*”),
*whereRef*($o_3$, “*LastName*”, “ ”, $q_2$),
*whereRef*($o_3$, “*Personalia*”, “*LastName*”, $q_2$),
*whereRef*($o_3$, “*Director*”, “*Personalia/LastName*”, $q_2$),
*whereRef*($o_3$, “*Movie*”, “*Director/Personalia/LastName*”, $q_2$),
*whereRef*($o_3$, “*MovieDB*”, “*Movie/Director/Personalia/LastName*”, $q_2$),
*subpath*($o_3$, “*LastName*”, “ ”, “*Personalia*”, “*LastName*”),
*subpath*($o_3$, “*Personalia*”, “*LastName*”, “*Director*”, “*Personalia/LastName*”),
*subpath*($o_3$, “*Director*”, “*Personalia/LastName*”,
“*Movie*”, “*Director/Personalia/LastName*”),
*subpath*($o_3$, ”*Movie*”, “*Director/Personalia/LastName*”,
“*MovieDB*”, “*Movie/Director/Personalia/LastName*”),
*whereRef*($o_4$, “*MovieDB*”, “*Movie/Actor/Personalia*”, $q_2$),
*whereRef*($o_4$, “*Movie*”, “*Actor/Personalia*”, $q_2$),
*whereRef*($o_4$, “*Actor*”, “*Personalia*”, $q_2$),
*whereRef*($o_4$, “*Personalia*”, “ ”, $q_2$),
*subpath*($o_4$, “*Personalia*”, “ ”, “*Actor*”, “*Personalia*”),
*subpath*($o_4$, “*Actor*”, “*Personalia*”, “*Movie*”, “*Actor/Personalia*”),
*subpath*($o_4$, ”*Movie*”, “*Actor/Personalia*”, “*MovieDB*”, “*Movie/Actor/Personalia*”),
*whereRef*($o_5$, “*MovieDB*”, “*Person/ID*”, $q_2$),
*whereRef*($o_5$, “*Person*”, “*ID*”, $q_2$),
*whereRef*($o_5$, “*ID*”, “ ”, $q_2$),
*subpath*($o_5$, “*ID*”, “ ”, “*Person*”, “*ID*”),
*subpath*($o_5$, “*Person*”, “*ID*”, “*MovieDB*”, “*Person/ID*”),
*whereRef*($o_6$, “*MovieDB*”, “*Person/FirstName*”, $q_2$),
*whereRef*($o_6$, “*Person*”, “*FirstName*”, $q_2$),
*whereRef*($o_6$, “*FirstName*”, “ ”, $q_2$),
*subpath*($o_6$, “*FirstName*”, “ ”, “*Person*”, “*FirstName*”),
*subpath*($o_6$, “*Person*”, “*FirstName*”, “*MovieDB*”, “*Person/FirstName*”),
*whereRef*($o_7$, “*MovieDB*”, “*Person/LastName*”, $q_2$),
*whereRef*($o_7$, “*Person*”, “*LastName*”, $q_2$),
*whereRef*($o_7$, “*LastName*”, “ ”, $q_2$),
*subpath*($o_7$, “*LastName*”, “ ”, “*Person*”, “*LastName*”),
*subpath*($o_6$, “*Person*”, “*LastName*”, “*MovieDB*”, “*Person/LastName*”),

$whereCmp(o_2, equal, \text{"}Josef\text{"}), whereCmp(o_3, equal, \text{"}Sternberg\text{"}),$
$whereCmp(o_6, equal, \text{"}Marlene\text{"}), whereCmp(o_3, equal, \text{"}Dietrich\text{"}),$
$consRef(o_7, \text{"}Movie\text{"}, \text{" "}), consRef(o_1, \text{"}MovieList\text{"}, \text{"}Movie\text{"}), whereRefCmp(o_4, equal, o_5)\}$

## C Appendix: High-level Query-Description Examples

The complete high-level description (except auxiliary predicates and the completion of the subpath predicates) of the query in Example 1 is given by the following set of facts:

$\{\ query(q_1),$
$access(o_1, \text{"}MovieDB\text{"}, \text{"}Movie/Title\text{"}, q_1),$
$access(o_1, \text{"}Movie\text{"}, \text{"}Title\text{"}, q_1),$
$access(o_2, \text{"}MovieDB\text{"}, \text{"}Movie/Director/Personalia/FirstName\text{"}, q_1),$
$access(o_2, \text{"}Movie\text{"}, \text{"}Director/Personalia/FirstName\text{"}, q_1),$
$access(o_2, \text{"}Director\text{"}, \text{"}Personalia/FirstName\text{"}, q_1),$
$access(o_2, \text{"}Person\text{"}, \text{"}FirstName\text{"}, q_1),$
$access(o_3, \text{"}MovieDB\text{"}, \text{"}Movie/Director/Personalia/LastName\text{"}, q_1),$
$access(o_3, \text{"}Movie\text{"}, \text{"}Director/Personalia/LastName\text{"}, q_1),$
$access(o_3, \text{"}Director\text{"}, \text{"}Personalia/LastName\text{"}, q_1),$
$access(o_3, \text{"}Person\text{"}, \text{"}LastName\text{"}, q_1),$
$occurs(o_2, \text{"}Alfred\text{"}), occurs(o_3, \text{"}Hitchcock\text{"}),$
$selects(o_2, equal, \text{"}Alfred\text{"}), selects(o_3, equal, \text{"}Hitchcock\text{"}),$
$constructs(o_1, \text{"}Movie\text{"}, \text{" "})\ \}$

The complete high-level description of the query in Example 7 is given by the following set of facts (auxiliary predicates and the completion of the subpath predicates again not included):

$\{\ query(q_1),$
$access(o_1, \text{"}MovieDB\text{"}, \text{"}Movie/Title\text{"}, q_1),$
$access(o_1, \text{"}Movie\text{"}, \text{"}Title\text{"}, q_1),$
$access(o_2, \text{"}MovieDB\text{"}, \text{"}Movie/Director/Personalia/FirstName\text{"}, q_1),$
$access(o_2, \text{"}Movie\text{"}, \text{"}Director/Personalia/FirstName\text{"}, q_1),$
$access(o_2, \text{"}Director\text{"}, \text{"}Personalia/FirstName\text{"}, q_1),$
$access(o_2, \text{"}Person\text{"}, \text{"}FirstName\text{"}, q_1),$
$access(o_3, \text{"}MovieDB\text{"}, \text{"}Movie/Director/Personalia/LastName\text{"}, q_1),$
$access(o_3, \text{"}Movie\text{"}, \text{"}Director/Personalia/LastName\text{"}, q_1),$
$access(o_3, \text{"}Director\text{"}, \text{"}Personalia/LastName\text{"}, q_1),$
$access(o_3, \text{"}Person\text{"}, \text{"}LastName\text{"}, q_1),$
$access(o_4, \text{"}MovieDB\text{"}, \text{"}Movie/Actor/Personalia\text{"}, q_1),$
$access(o_4, \text{"}Movie\text{"}, \text{"}Actor/Personalia\text{"}, q_1),$
$access(o_4, \text{"}Actor\text{"}, \text{"}Personalia\text{"}, q_1),$
$access(o_4, \text{"}Person\text{"}, \text{" "}, q_1),$
$access(o_5, \text{"}MovieDB\text{"}, \text{"}Person/ID\text{"}, q_1),$

$access(o_5, \text{“}Person\text{”}, \text{“}ID\text{”}, q_1),$
$access(o_6, \text{“}MovieDB\text{”}, \text{“}Person/FirstName\text{”}, q_1),$
$access(o_6, \text{“}Person\text{”}, \text{“}FirstName\text{”}, q_1),$
$access(o_7, \text{“}MovieDB\text{”}, \text{“}Person/LastName\text{”}, q_1),$
$access(o_7, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1),$
$occurs(o_2, \text{“}Josef\text{”}), occurs(o_3, \text{“}Sternberg\text{”}),$
$occurs(o_6, \text{“}Marlene\text{”}), occurs(o_7, \text{“}Dietrich\text{”}),$
$selects(o_2, equal, \text{“}Josef\text{”}), selects(o_3, equal, \text{“}Sternberg\text{”}),$
$selects(o_6, equal, \text{“}Marlene\text{”}), selects(o_7, equal, \text{“}Dietrich\text{”}),$
$constructs(o_1, \text{“}Movie\text{”}, \text{“ ”}), joins(o_4, o_5, equal)\ \}$

# D Appendix: Experimental Movie Doamin Knowledge

In our experiments the following ELP served as the domain theory, $\Pi_{dom}$, for the movie domain:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% input for query abstraction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


class0(s_MovieDB).
class0(s_Movie).
class0(s_Director).
class0(s_Actor).
class0(s_Screenwriter).
class0(s_Composer).
class0(s_Person).
class0(s_Soundtrack).
class0(s_Review).
synonym(s_Personalia,s_Person).


%%%%%%%%%%%%%%%%%
% domain ontology
%%%%%%%%%%%%%%%%%

% classes

class(s_Movie).
class(s_Person).
class(name).
class(s_Actor).
class(s_Director).
class(s_Screenwriter).
class(s_Composer).
class(s_Soundtrack).
class(s_Review).
```

```
% subclasses

is_a(s_Actor,s_Person).
is_a(s_Director,s_Person).
is_a(s_Screenwriter,s_Person).
is_a(s_Composer,s_Person).

% attributes

has_att(s_Movie,title).
has_att(s_Movie,alternativeTitles).
has_att(s_Movie,genre).
has_att(s_Movie,language).
has_att(s_Movie,runningTime).
has_att(s_Movie,releaseDate).
has_att(s_Movie,review).

has_att(s_Person,id).
has_att(s_Person,gender).
has_att(s_Person,name).
has_att(s_Person,birthDate).
has_att(s_Person,dateOfDeath).
has_att(s_Person,country).
has_att(s_Person,biography).

has_att(name,firstName).
has_att(name,lastName).

has_att(nameList,name).

has_att(s_Actor,personalia).
has_att(s_Director,personalia).
has_att(s_Screenwriter,personalia).
has_att(s_Composer,personalia).

has_att(s_Soundtrack,title).
has_att(s_Soundtrack,composer).

has_att(s_Review,text).
has_att(s_Review,rating).

% inheritance

instance(I,C2) :- instance(I,C1), is_a(C1,C2).
has_att(C2,A) :- is_a(C1,C2), has_att(C1,A).


%%%%%%%%%%%%%%%%%%%%%%
% background knowledge
%%%%%%%%%%%%%%%%%%%%%%

% common sense

#maxint=2020.
```

```
genre("Action").
genre("Animation").
genre("Classic").
genre("Comedy").
genre("CowboyWestern").
genre("CultMovie").
genre("Documentary").
genre("Experimental").
genre("FilmNoir").
genre("Horror").
genre("Romance").
genre("SciFiFantasy").
genre("Series").
genre("Silent").
genre("Travel").
genre("Other").
genre(G) :- att_val(M,genre,G).


calender_year(X,Y) :- <(X,100), Y=X+1900.
%calender_year(X,X) :- #int(X), >=(X,100).
calender_year(X,X) :- #int(X), >=(X,1850).
year(Y) :- calender_year(X,Y).
current_year(2002).

decade_name(twenties).
decade_name(thirties).
decade_name(fourties).
decade_name(fifties).
decade_name(sixties).
decade_name(seventies).
decade_name(eighties).
decade_name(nineties).
decade_name(twothousands).
decade(Y,twenties) :- calender_year(X,Y), >=(Y,1920), <(Y,1930).
decade(Y,thirties) :- calender_year(X,Y), >=(Y,1930), <(Y,1940).
decade(Y,fourties) :- calender_year(X,Y), >=(Y,1940), <(Y,1950).
decade(Y,fifties) :- calender_year(X,Y), >=(Y,1950), <(Y,1960).
decade(Y,sixties) :- calender_year(X,Y), >=(Y,1960), <(Y,1970).
decade(Y,seventies) :- calender_year(X,Y), >=(Y,1970), <(Y,1980).
decade(Y,eighties) :- calender_year(X,Y), >=(Y,1980), <(Y,1990).
decade(Y,nineties) :- calender_year(X,Y), >=(Y,1990), <(Y,2000).
decade(Y,twothousands) :- calender_year(X,Y), >=(Y,2000), <(Y,2010).
decade_before(D1,D2) :- decade(Y1,D1), decade(Y2,D2), !=(D1,D2), <(Y1,Y2).
decade_after(D1,D2) :- decade(Y1,D1), decade(Y2,D2), !=(D1,D2), >(Y1,Y2).
decade_between(D1,D2,D3) :- decade(Y1,D1), decade(Y2,D2), decade(Y3,D3),
                            <(Y2,Y3), <=(Y2,Y1), >=(Y3,Y1).

rating_val(X) :- #int(X), <=(X,100).

involved(P,M) :- directed(P,M).
involved(P,M) :- acted(P,M).
involved(P,M) :- screen_written(P,M).
involved(P,M) :- sound_composed(P,M).
```

```
sound_composed(P,M) :- sound_track(S,M), composed(P,S).

instance(P,s_Director) :- directed(P,M).
instance(P,s_Actor) :- acted(P,M).
instance(P,s_Screenwriter) :- screen_written(P,M).
instance(P,s_Composer) :- sound_composed(P,M).

life_period(P,B,E) :- instance(P,s_Person), att_val(P,birthDate,B1),
                      not dead(P), calender_year(B1,B),
                      current_year(E).
life_period(P,B,E) :- instance(P,s_Person), att_val(P,birthDate,B1),
                      att_val(P,dateOfDeath,E1), calender_year(B1,B),
                      calender_year(E1,E).
dead(P) :- att_val(P,dateOfDeath,D).

defined_releaseDate(M) :- att_val(M,releaseDate,Y).
release_bound(M,X,Y) :- directed(M,D), prod_period(D,B,E), calender_year(B,X),
                        calender_year(E,Y), not defined_releaseDate(M).
release_bound(M,X,Y) :- acted(M,A), prod_period(A,B,E), calender_year(B,X),
                        calender_year(E,Y), not defined_releaseDate(M).
defined_genre(M) :-  att_val(M,genre,G).
possible_genre(M,G) :-  involved(P,M), default_genre(P,G), not defined_genre(M).
possible_genre(P,G) :-  involved(P,M), possible_genre(M,G), not defined_genre(M).
possible_genre(P,G) :-  default_genre(P,G).

att_val(M,genre,"Silent") :- instance(M,s_Movie), att_val(M,releaseDate,D),
                             calender_year(D,R), <(R,1920).
att_val(M,genre,"Silent") :- instance(M,s_Movie), att_val(M,releaseDate,D),
                             calender_year(D,R), <(R,1930),
                             not -att_val(M,genre,"Silent").
-att_val(M,genre,"Silent") :- att_val(M,genre,G), !=(G,"Silent").

:- prod_period(P,X,Y), calender_year(X,B), calender_year(Y,E), >(B,E).
:- prod_period(P,X,Y), att_val(P,birthDate,B), calender_year(X,B1),
   calender_year(B,B2), <(B1,B2).
:- prod_period(P,X,Y), att_val(P,dateOfDeath,D), calender_year(Y,E1),
   calender_year(D,E2), >(E1,E2).


% Persons:
%%%%%%%%%%

% Alfred Hitchcock

instance(perHitchcock,s_Director).
instance(perHitchcock,s_Screenwriter).
instance(perHitchcock,s_Actor).
att_val(perHitchcock,name,nameHitchcock).

instance(nameHitchcock,name).
att_val(nameHitchcock,firstName,"Alfred").
att_val(nameHitchcock,firstName,"Joseph").
att_val(nameHitchcock,firstName,"Sir").
att_val(nameHitchcock,lastName,"Hitchcock").
```

```
prod_period(perHitchcock,1920,1980).
default_genre(perHitchcock,"Thriller").
default_genre(perHitchcock,"Crime").

% Marlene Dietrich

instance(perDietrich,s_Actor).
att_val(perDietrich,name,nameDietrich).
att_val(perDietrich,birthDate,1901).
att_val(perDietrich,dateOfDeath,1992).

instance(nameDietrich,name).
att_val(nameDietrich,firstName,"Marlene").
att_val(nameDietrich,firstName,"Maria").
att_val(nameDietrich,firstName,"Magdalena").
att_val(nameDietrich,lastName,"Dietrich").

prod_period(perDietrich,1920,1960).
default_genre(perDietrich,"Drama").

% Marilyn Monroe

instance(perMonroe,s_Actor).
att_val(perMonroe,name,nameMM).
att_val(perMonroe,name,nameMonroe).
att_val(perMonroe,birthDate,1926).
att_val(perMonroe,dateOfDeath,1962).

instance(nameMM,name).
att_val(nameMM,firstName,"Marilyn").
att_val(nameMM,lastName,"Monroe").

instance(nameMonroe,name).
att_val(nameMonroe,firstName,"Norma").
att_val(nameMonroe,firstName,"Jean").
att_val(nameMonroe,lastName,"Mortensen").

prod_period(perMonroe,1948,1962).
default_genre(perMonroe,"Comedy").

% Cary Grant

instance(perGrant,s_Actor).
att_val(perGrant,name,nameCG).
att_val(perGrant,name,nameGrant).
att_val(perGrant,birthDate,1904).
att_val(perGrant,dateOfDeath,1986).

instance(nameCG,name).
att_val(nameCG,firstName,"Cary").
att_val(nameCG,lastName,"Grant").

instance(nameGrant,name).
att_val(nameGrant,firstName,"Archibald").
att_val(nameGrant,firstName,"Alec").
```

```
att_val(nameGrant,lastName,"Leach").

prod_period(perGrant,1920,1970).
default_genre(perGrant,"Comedy").

% Grace Kelly

instance(perKelly,s_Actor).
att_val(perKelly,name,nameKelly).
att_val(perKelly,birthDate,1929).
att_val(perKelly,dateOfDeath,1982).

instance(nameKelly,name).
att_val(nameKelly,firstName,"Grace").
att_val(nameKelly,firstName,"Patricia").
att_val(nameKelly,lastName,"Kelly").

prod_period(perKelly,1945,1960).

% John Barry

instance(perBarry,s_Composer).
att_val(perBarry,name,nameBarry).
att_val(perBarry,birthDate,1933).

instance(nameBarry,name).
att_val(nameBarry,firstName,"John").
att_val(nameBarry,firstName,"Prendergast").
att_val(nameBarry,lastName,"Barry").

prod_period(perBarry,1955,X):-current_year(X).

% Elmer Bernstein

instance(perBernstein,s_Composer).
att_val(perBernstein,name,nameBernstein).
att_val(perBernstein,birthDate,1922).

instance(nameBernstein,name).
att_val(nameBernstein,firstName,"Elmer").
att_val(nameBernstein,lastName,"Bernstein").

prod_period(perBernstein,1955,2000).

% Vincent Price

instance(perPrice,s_Actor).
att_val(perPrice,name,namePrice).
att_val(perPrice,birthDate,1911).
att_val(perPrice,dateOfDeath,1993).

instance(namePrice,name).
att_val(namePrice,firstName,"Vincent").
att_val(namePrice,lastName,"Price").
```

```
prod_period(perPrice,1935,1990).
default_genre(perPrice,"Horror").

% John Wayne

instance(perWayne,s_Actor).
instance(perWayne,s_Director).
att_val(perWayne,name,nameWayne).
att_val(perWayne,birthDate,1907).
att_val(perWayne,dateOfDeath,1979).

instance(nameWayne,name).
att_val(nameWayne,firstName,"John").
att_val(nameWayne,lastName,"Wayne").

prod_period(perWayne,1926,1975).
default_genre(perWayne,"CowboyWestern").

% Giorgio Moroder

instance(perMoroder,s_Composer).
att_val(perMoroder,name,nameMoroder).
att_val(perMoroder,birthDate,1940).

instance(nameMoroder,name).
att_val(nameMoroder,firstName,"Giorgio").
att_val(nameMoroder,lastName,"Moroder").

prod_period(perMoroder,1970,X):-current_year(X).

% Sean Connery

instance(perConnery,s_Actor).
att_val(perConnery,name,nameConnery).
att_val(perConnery,birthDate,1930).

instance(nameConnery,name).
att_val(nameConnery,firstName,"Sean").
att_val(nameConnery,lastName,"Connery").

prod_period(perConnery,1955,X):-current_year(X).
default_genre(perConnery,"Action").

% Movies:  (largest movie-id used: m65)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

instance(m1,s_Movie).
att_val(m1,title,"There's No Business Like Show Business").
att_val(m1,releaseDate,1954).
acted(perMonroe,m1).

instance(m2,s_Movie).
att_val(m2,title,"Niagara").
acted(perMonroe,m2).
```

```
instance(m3,s_Movie).
att_val(m3,title,"River of no Return").
att_val(m3,releaseDate,1954).
acted(perMonroe,m3).

instance(m4,s_Movie).
att_val(m4,title,"Bus Stop").
acted(perMonroe,m4).

instance(m5,s_Movie).
att_val(m5,title,"Some Like it Hot").
att_val(m5,releaseDate,1959).
acted(perMonroe,m5).


instance(m6,s_Movie).
att_val(m6,title,"Blaue Engel, Der").
att_val(m6,releaseDate,1930).
acted(perDietrich,m6).

instance(m7,s_Movie).
att_val(m7,title,"Shanghai Express").
att_val(m7,releaseDate,1932).
acted(perDietrich,m7).

instance(m8,s_Movie).
att_val(m8,title,"Blonde Venus").
att_val(m8,releaseDate,1932).
acted(perDietrich,m8).

instance(m9,s_Movie).
att_val(m9,title,"Devil Is a Woman, The").
att_val(m9,releaseDate,1935).
acted(perDietrich,m9).

instance(m10,s_Movie).
att_val(m10,title,"Witness for the Prosecution").
acted(perDietrich,m10).

instance(m11,s_Movie).
att_val(m11,title,"Notorious").
acted(perGrant,m11).
directed(perHitchcock,m11).

instance(m12,s_Movie).
att_val(m12,title,"Arsenic and Old Lace").
att_val(m12,releaseDate,1944).
acted(perGrant,m12).

instance(m13,s_Movie).
att_val(m13,title,"I Was a Male War Bride").
att_val(m13,releaseDate,1949).
acted(perGrant,m13).

instance(m14,s_Movie).
```

```
att_val(m14,title,"To Catch a Thief").
acted(perGrant,m14).
acted(perKelly,m14).
directed(perHitchcock,m14).

instance(m15,s_Movie).
att_val(m15,title,"Houseboat").
acted(perGrant,m15).

instance(m16,s_Movie).
att_val(m16,title,"Charade").
att_val(m16,releaseDate,1963).
acted(perGrant,m16).

instance(m17,s_Movie).
att_val(m17,title,"Rear Window").
acted(perKelly,m17).
directed(perHitchcock,m17).

instance(m18,s_Movie).
att_val(m18,title,"Dial M for Murder").
att_val(m18,releaseDate,1954).
acted(perKelly,m18).
directed(perHitchcock,m18).

instance(m19,s_Movie).
att_val(m19,title,"High Noon").
att_val(m19,releaseDate,1952).
acted(perKelly,m19).

instance(m20,s_Movie).
att_val(m20,title,"High Society").
acted(perKelly,m20).

instance(m21,s_Movie).
att_val(m21,title,"From Russia with Love").
sound_composed(perBarry,m21).
acted(perConnery,m21).

instance(m22,s_Movie).
att_val(m22,title,"Goldfinger").
att_val(m22,releaseDate,1964).
sound_composed(perBarry,m22).
acted(perConnery,m22).

instance(m23,s_Movie).
att_val(m23,title,"Ipcress File, The").
sound_composed(perBarry,m23).

instance(m24,s_Movie).
att_val(m24,title,"Born Free").
sound_composed(perBarry,m24).

instance(m25,s_Movie).
att_val(m25,title,"Lion in Winter, The").
```

```
sound_composed(perBarry,m25).

instance(m26,s_Movie).
att_val(m26,title,"Thunderball").
att_val(m26,releaseDate,1965).
sound_composed(perBarry,m26).

instance(m27,s_Movie).
att_val(m27,title,"King Kong").
att_val(m27,releaseDate,1976).
sound_composed(perBarry,m27).

instance(m28,s_Movie).
att_val(m28,title,"Deep, The").
sound_composed(perBarry,m28).

instance(m29,s_Movie).
att_val(m29,title,"Black Hole, The").
sound_composed(perBarry,m29).

instance(m30,s_Movie).
att_val(m30,title,"View to a Kill, A").
att_val(m30,releaseDate,1985).
sound_composed(perBarry,m30).

instance(m31,s_Movie).
att_val(m31,title,"Howard the Duck").
sound_composed(perBarry,m31).

instance(m32,s_Movie).
att_val(m32,title,"Indicent Proposal").
att_val(m30,releaseDate,1993).
sound_composed(perBarry,m32).

instance(m33,s_Movie).
att_val(m33,title,"Ten Commandments, The").
sound_composed(perBernstein,m33).

instance(m34,s_Movie).
att_val(m34,title,"Man with the Golden Arm, The").
sound_composed(perBernstein,m34).

instance(m35,s_Movie).
att_val(m35,title,"Buccaneer, The").
att_val(m35,releaseDate,1958).
sound_composed(perBernstein,m35).

instance(m36,s_Movie).
att_val(m36,title,"Magnificent Seven, The").
att_val(m36,releaseDate,1960).
sound_composed(perBernstein,m36).

instance(m37,s_Movie).
att_val(m37,title,"To Kill a Mockingbird").
sound_composed(perBernstein,m37).
```

```
instance(m38,s_Movie).
att_val(m38,title,"Cannon for Cordoba").
sound_composed(perBernstein,m38).

instance(m39,s_Movie).
att_val(m39,title,"Ghostbusters").
att_val(m39,releaseDate,1984).
sound_composed(perBernstein,m39).

instance(m40,s_Movie).
att_val(m40,title,"Wild Wild West").
att_val(m40,releaseDate,1999).
sound_composed(perBernstein,m40).

instance(m41,s_Movie).
att_val(m41,title,"Laura").
acted(perPrice,m41).

instance(m42,s_Movie).
att_val(m42,title,"Three Musketeers, The").
att_val(m42,releaseDate,1948).
acted(perPrice,m42).

instance(m43,s_Movie).
att_val(m43,title,"Bat, The").
acted(perPrice,m43).

instance(m44,s_Movie).
att_val(m44,title,"Fly, The").
att_val(m44,releaseDate,1958).
acted(perPrice,m44).

instance(m45,s_Movie).
att_val(m45,title,"Return of the Fly").
att_val(m45,releaseDate,1959).
acted(perPrice,m45).

instance(m46,s_Movie).
att_val(m46,title,"House of Usher").
att_val(m46,releaseDate,1960).
acted(perPrice,m46).

instance(m47,s_Movie).
att_val(m47,title,"Raven, The").
acted(perPrice,m47).

instance(m48,s_Movie).
att_val(m48,title,"Stagecoach").
acted(perWayne,m48).

instance(m49,s_Movie).
att_val(m49,title,"Alamo, The").
att_val(m49,releaseDate,1960).
acted(perWayne,m49).
```

```
directed(perWayne,m49).

instance(m50,s_Movie).
att_val(m50,title,"Green Berets, The").
att_val(m50,releaseDate,1968).
acted(perWayne,m50).
directed(perWayne,m50).

instance(m51,s_Movie).
att_val(m51,title,"Rio Grande").
att_val(m51,releaseDate,1950).
acted(perWayne,m51).

instance(m52,s_Movie).
att_val(m52,title,"Sands of Iwo Jima").
acted(perWayne,m52).

instance(m53,s_Movie).
att_val(m53,title,"Flying Tigers").
acted(perWayne,m53).

instance(m54,s_Movie).
att_val(m54,title,"Man Who Shot Liberty Valance, The").
att_val(m54,releaseDate,1962).
acted(perWayne,m54).

instance(m55,s_Movie).
att_val(m55,title,"American Gigolo").
att_val(m55,releaseDate,1980).
sound_composed(perMoroder,m55).

instance(m56,s_Movie).
att_val(m56,title,"Midnight Express").
att_val(m56,releaseDate,1978).
sound_composed(perMoroder,m56).

instance(m57,s_Movie).
att_val(m57,title,"Over the Top").
att_val(m57,releaseDate,1987).
sound_composed(perMoroder,m57).

instance(m58,s_Movie).
att_val(m58,title,"Dr. No").
att_val(m58,releaseDate,1962).
acted(perConnery,m58).

instance(m59,s_Movie).
att_val(m59,title,"Zardoz").
att_val(m59,releaseDate,1974).
acted(perConnery,m59).

instance(m60,s_Movie).
att_val(m60,title,"Bridge Too Far, A").
acted(perConnery,m60).
```

```
instance(m61,s_Movie).
att_val(m61,title,"Meteor").
att_val(m61,releaseDate,1979).
acted(perConnery,m61).

instance(m62,s_Movie).
att_val(m62,title,"Highlander").
att_val(m62,releaseDate,1986).
acted(perConnery,m62).

instance(m63,s_Movie).
att_val(m63,title,"Name der Rose, Der").
att_val(m63,releaseDate,1986).
acted(perConnery,m63).

instance(m64,s_Movie).
att_val(m64,title,"Hunt for Red October, The").
att_val(m64,releaseDate,1990).
acted(perConnery,m64).

instance(m65,s_Movie).
att_val(m65,title,"Avengers, The").
att_val(m65,releaseDate,1998).
acted(perConnery,m65).
```

# E Appendix: Experimental Site Description Program

The following ELP served as the site description program, $\Pi_{sd}$, for the sites used in our experiments:

```
%%%%%%%%%%%%%%%%%%%
% Technical aspects
%%%%%%%%%%%%%%%%%%%

site(s_Eachmovie).
site(s_Hitchcock).
site(s_Horror60).
site(s_KellyGrant).
site(s_RandomMovies).                     % not for "reduced" selection base
site(s_RandomPersons).
site(s_RandomMoviesNew).                  % only for "extended" selection base

up(s_Eachmovie).
up(s_Hitchcock).
up(s_Horror60).
up(s_KellyGrant).
up(s_RandomMovies).                       % not for "reduced" selection base
up(s_RandomPersons).
up(s_RandomMoviesNew).                    % only for "extended" selection base

site_type(S,public,queryable) :- site(S), not -site_type(S,public,queryable).
-site_type(S,public,queryable) :- site_type(S,T1,T2), !=(T1,public).
-site_type(S,public,queryable) :- site_type(S,T1,T2), !=(T2,queryable).
```

```
site_language(S,"English") :- site(S), not -site_language(S,"English").
-site_language(S,"English") :- site_language(S,L), !=(L,"English").

data_format(s_Eachmovie,xml).
data_format(s_Hitchcock,xml).
data_format(s_Horror60,xml).
data_format(s_KellyGrant,xml).
data_format(s_RandomMovies,xml).            % not for "reduced" selection base
data_format(s_RandomPersons,xml).
data_format(s_RandomMoviesNew,xml).         % only for "extended" selection base

reliable(S,high) :- site(S), not -reliable(S,high).
-reliable(S,high) :- reliable(S,V), !=(V,high).

update_frequency(s_Eachmovie,medium).
update_frequency(s_Hitchcock,low).
update_frequency(s_Horror60,low).
update_frequency(s_KellyGrant,low).
update_frequency(s_RandomMovies,high).      % not for "reduced" selection base
update_frequency(s_RandomPersons,high).
update_frequency(s_RandomMoviesNew,high).   % only for "extended" selection base

%%%%%%%%%%%%%%
% Cost aspects
%%%%%%%%%%%%%%

charge(S,no) :- site(S), not -charge(S,no).
-charge(S,no) :- charge(S,V), !=(V,no).

%%%%%%%%%%%%%%%%%%
% Thematic aspects
%%%%%%%%%%%%%%%%%%

topic(T) :- accurate(S,T,V).
topic(T) :- -accurate(S,T,V).
topic(T) :- covers(S,T,V).
topic(T) :- -covers(S,T,V).
topic(T) :- specialized(S,T).
topic(T) :- -specialized(S,T).
topic(T) :- relevant(S,T).
topic(T) :- -relevant(S,T).

accurate(S,T,high) :- site(S), topic(T), not -accurate(S,T,high).
-accurate(S,T,high) :- accurate(S,T,V), !=(V,high).
-accurate(S,T,high) :- -relevant(S,T).

covers(S,T,high) :- specialized(S,T), not -covers(S,T,high).
-covers(S,T,high) :- covers(S,T,V), !=(V,high).
covers(S,T,medium) :- relevant(S,T), not specialized(S,T),
                      not -covers(S,T,medium).
-covers(S,T,medium) :- covers(S,T,V), !=(V,medium).
covers(S,T,low) :- -relevant(S,T), not -covers(S,T,low).
-covers(S,T,low) :- covers(S,T,V), !=(V,low).

covers(s_Hitchcock,s_Movie,low).
```

```
covers(s_Hitchcock,s_ReleaseDate,high).
covers(s_Hitchcock,s_Review,high).
covers(s_Hitchcock,s_Soundtrack,low).
covers(s_Hitchcock,s_Person,high).
covers(s_Hitchcock,s_BirthDate,high).
covers(s_Hitchcock,s_Actor,high).
covers(s_Hitchcock,s_Director,low).
covers(s_Hitchcock,s_Composer,low).
covers(s_Hitchcock,fifties,high).

covers(s_Horror60,s_Movie,low).
covers(s_Horror60,s_ReleaseDate,high).
covers(s_Horror60,s_Genre,low).
covers(s_Horror60,s_Review,high).
covers(s_Horror60,s_Soundtrack,low).
covers(s_Horror60,s_Person,high).
covers(s_Horror60,s_Actor,high).
covers(s_Horror60,s_Composer,low).
covers(s_Horror60,sixties,high).

covers(s_KellyGrant,s_Movie,low).
covers(s_KellyGrant,s_ReleaseDate,high).
covers(s_KellyGrant,s_Person,high).
covers(s_KellyGrant,s_BirthDate,high).
covers(s_KellyGrant,s_Actor,high).
covers(s_KellyGrant,fifties,high).
covers(s_KellyGrant,sixties,high).

covers(s_RandomMovies,s_ReleaseDate,high).   % not for "reduced" selection base
covers(s_RandomMovies,s_Genre,high).         % not for "reduced" selection base
covers(s_RandomMovies,s_Review,low).         % not for "reduced" selection base
covers(s_RandomMovies,s_Soundtrack,high).    % not for "reduced" selection base
covers(s_RandomMovies,s_Person,high).        % not for "reduced" selection base
covers(s_RandomMovies,s_Actor,high).         % not for "reduced" selection base
covers(s_RandomMovies,s_Composer,high).      % only for "extended" selection base
covers(s_RandomMovies,s_Screenwriter,high).  % not for "reduced" selection base
covers(s_RandomMovies,"CowboyWestern",high). % only for "extended" selection base

covers(s_RandomMoviesNew,s_Review,low).      % only for "extended" selection base
covers(s_RandomMoviesNew,s_Soundtrack,high). % only for "extended" selection base
covers(s_RandomMoviesNew,s_Person,high).     % only for "extended" selection base
covers(s_RandomMoviesNew,s_Actor,high).      % only for "extended" selection base
covers(s_RandomMoviesNew,s_Director,high).   % only for "extended" selection base
covers(s_RandomMoviesNew,"Drama",high).      % only for "extended" selection base
covers(s_RandomMoviesNew,"Comedy",high).     % only for "extended" selection base

covers(s_RandomPersons,s_Movie,low).
covers(s_RandomPersons,s_Actor,high).
covers(s_RandomPersons,s_Director,high).
covers(s_RandomPersons,s_Screenwriter,high).

specialized(s_Eachmovie,s_Rating).
specialized(s_Hitchcock,"Hitchcock").
specialized(s_Horror60,"Horror").
specialized(s_KellyGrant,"Kelly").
```

```
specialized(s_KellyGrant,"Grant").
specialized(s_RandomMovies,s_Movie).         % not for "reduced" selection base
specialized(s_RandomPersons,s_Person).
specialized(s_RandomMoviesNew,s_Movie).      % only for "extended" selection base

relevant(S,T) :- site(S), topic(T), not -relevant(S,T).

-relevant(s_Eachmovie,s_Person).
-relevant(s_Eachmovie,s_BirthDate).
-relevant(s_Eachmovie,s_Soundtrack).
-relevant(s_Eachmovie,s_RunningTime).

-relevant(s_Hitchcock,eighties).
-relevant(s_Hitchcock,nineties).
-relevant(s_Hitchcock,twothousands).

-relevant(s_Horror60,twenties).
-relevant(s_Horror60,thirties).
-relevant(s_Horror60,fourties).
-relevant(s_Horror60,fifties).
-relevant(s_Horror60,seventies).
-relevant(s_Horror60,eighties).
-relevant(s_Horror60,nineties).
-relevant(s_Horror60,twothousands).

-relevant(s_KellyGrant,seventies).
-relevant(s_KellyGrant,eighties).
-relevant(s_KellyGrant,nineties).
-relevant(s_KellyGrant,twothousands).

-relevant(s_RandomMovies,s_BirthDate).       % not for "reduced" selection base
-relevant(s_RandomMoviesNew,s_BirthDate).    % only for "extended" selection base

-relevant(s_RandomPersons,s_ReleaseDate).
-relevant(s_RandomPersons,s_RunningTime).
-relevant(s_RandomPersons,s_Review).
-relevant(s_RandomPersons,s_Soundtrack).
```

# F Appendix: Experimental Site-Selection Program

The following site selection program, $\Pi_{sel}$, has been used for experimentation. Note some slight syntactical differences. Similar to `plp`, our implementation requires the rule name to be the first predicate in the body of a site-selection rule. The arity of the name predicate is determined by the open variables of the site-selection rule.[3] Furthermore, the performance of our current implementation can be enhanced, by using $true$ as a name predicate for all rules that neither occur in the user preference relation $<_u$, nor contribute to implicit preferences $\preceq_c$.[4]

[3] Note that a simple preprocessor could perform this syntactical transformation.

[4] Such optimizations could also be automated by means of the aforementioned preprocessor, since a site-selection rule $r$ does not contribute to implicit preferences $\preceq_c$, iff none of the predicates $access$, $default\_class$, or $default\_path$ occurs in $B(r)$.

```
%%%%%%%%%%%%%%%
% Default rules
%%%%%%%%%%%%%%%


query_site(S,Q) :- r1(S,Q,O,P,V), default_path(O,P,Q), occurs(O,V), specialized(S,P).
query_site(S,Q) :- r2(S,Q,O,P,V), default_path(O,P,Q), occurs(O,V), covers(S,P,high).
query_site(S,Q) :- r3(S,Q,O,P,V), default_path(O,P,Q), occurs(O,V), relevant(S,P).

query_site(S,Q) :- r4(S,Q,O,T,V), default_class(O,T,Q), occurs(O,V), specialized(S,T).
query_site(S,Q) :- r5(S,Q,O,T,V), default_class(O,T,Q), occurs(O,V), covers(S,T,high).
query_site(S,Q) :- r6(S,Q,O,T,V), default_class(O,T,Q), occurs(O,V), relevant(S,T).

% not for "reduced" selection base
query_site(s_RandomMovies,Q) :- r20(Q,O), default_class(O,s_Movie,Q).

query_site(s_RandomPersons,Q) :- r21(Q,O), default_class(O,s_Person,Q).

% only for "extended" selection base
query_site(s_RandomMoviesNew,Q) :- r22(Q,O), default_class(O,s_Movie,Q).



%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Core site-selection rules
%%%%%%%%%%%%%%%%%%%%%%%%%%%


query_site(S,Q) :- r40(S,Q), site(S), query(Q), high_coverage(S,Q).
query_site(S,Q) :- r41(S,Q), site(S), query(Q), special(S,Q).


query_site(s_Eachmovie,Q) :- r60(Q,O,C,V),
                             access(O,s_MovieDB,s_Movie_Review_Rating,Q),
                             selects(O,C,V).
query_site(s_Hitchcock,Q) :- r61(Q,O1,O2),
                             access(O1,s_Person,s_LastName,Q),
                             selects(O1,equal,"Hitchcock"),
                             access(O2,s_Person,s_FirstName,Q),
                             selects(O2,equal,"Alfred").
query_site(s_Horror60,Q) :- r62(Q,O), access(O,s_MovieDB,s_Movie_Genre,Q),
                            selects(O,equal,"Horror"),
                            inferred_topic(Q,sixties).
query_site(s_KellyGrant,Q) :-  r63(Q,O1,O2),
                               access(O1,s_Person,s_LastName,Q),
                               selects(O1,equal,"Kelly"),
                               access(O2,s_Person,s_FirstName,Q),
                               selects(O2,equal,"Grace").
query_site(s_KellyGrant,Q) :-  r64(Q,O1,O2),
                               access(O1,s_Person,s_LastName,Q),
                               selects(O1,equal,"Grant"),
                               access(O2,s_Person,s_FirstName,Q),
                               selects(O2,equal,"Cary").
```

```
%%%%%%%%%%%%%
% Preferences
%%%%%%%%%%%%%

r2(_,Q,_,_,_)<r1(_,Q,_,_,_).
r3(_,Q,_,_,_)<r1(_,Q,_,_,_).
r3(_,Q,_,_,_)<r2(_,Q,_,_,_).

r5(_,Q,_,_,_)<r4(_,Q,_,_,_).
r6(_,Q,_,_,_)<r4(_,Q,_,_,_).
r6(_,Q,_,_,_)<r5(_,Q,_,_,_).

r1(_,Q,_,_,_)<r40(_,Q).
r1(_,Q,_,_,_)<r41(_,Q).
r2(_,Q,_,_,_)<r40(_,Q).
r2(_,Q,_,_,_)<r41(_,Q).
r3(_,Q,_,_,_)<r40(_,Q).
r3(_,Q,_,_,_)<r41(_,Q).
r4(_,Q,_,_,_)<r40(_,Q).
r4(_,Q,_,_,_)<r41(_,Q).
r5(_,Q,_,_,_)<r40(_,Q).
r5(_,Q,_,_,_)<r41(_,Q).
r6(_,Q,_,_,_)<r40(_,Q).
r6(_,Q,_,_,_)<r41(_,Q).

r20(Q,_)<r41(_,Q).       % not for "reduced" selection base
r21(Q,_)<r41(_,Q).
r22(Q,_)<r41(_,Q).       % only for "extended" selection base

r40(_,Q)<r41(_,Q).

r60(Q,_,_,_)<r40(_,Q).
r60(Q,_,_,_)<r41(_,Q).

r20(Q,_)<r61(Q,_,_).     % not for "reduced" selection base
r21(Q,_)<r61(Q,_,_).
r22(Q,_)<r61(Q,_,_).     % only for "extended" selection base
r20(Q,_)<r62(Q,_).       % not for "reduced" selection base
r21(Q,_)<r62(Q,_).
r22(Q,_)<r62(Q,_).       % only for "extended" selection base
r20(Q,_)<r63(Q,_,_).     % not for "reduced" selection base
r21(Q,_)<r63(Q,_,_).
r22(Q,_)<r63(Q,_,_).     % only for "extended" selection base
r20(Q,_)<r64(Q,_,_).     % not for "reduced" selection base
r21(Q,_)<r64(Q,_,_).
r22(Q,_)<r64(Q,_,_).     % only for "extended" selection base
r40(_,Q)<r61(Q,_,_).
r41(_,Q)<r61(Q,_,_).
r40(_,Q)<r62(Q,_).
r41(_,Q)<r62(Q,_).
r40(_,Q)<r63(Q,_,_).
r41(_,Q)<r63(Q,_,_).
r40(_,Q)<r64(Q,_,_).
r41(_,Q)<r64(Q,_,_).
```

```
%%%%%%%%%%%%%%%%%
% Auxiliary rules (largest rule-id used: 103)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-query_site(s_RandomPersons,Q) :- r70(Q,O),
                     access(O,s_MovieDB,s_Movie_Actor_Personalia,Q).
-query_site(s_RandomPersons,Q) :- r71(Q,O),
                     access(O,s_MovieDB,s_Movie_Director_Personalia,Q).
-query_site(s_RandomPersons,Q) :- r72(Q,O),
                     access(O,s_MovieDB,s_Movie_Composer_Personalia,Q).
-query_site(s_RandomPersons,Q) :- r73(Q,O),
                     access(O,s_MovieDB,s_Movie_Screenwriter_Personalia,Q).

-query_site(S,Q) :- true, irrelevant(S,Q).


high_coverage(S,Q) :- true, high_covered_topic(S,Q,T).
special(S,Q) :- true, special_topic(S,Q,T).

high_covered_topic(S,Q,T) :- true, query_topic(Q,T), covers(S,T,high).
high_covered_topic(S,Q,T) :- true, inferred_topic(Q,T), covers(S,T,high).

special_topic(S,Q,T) :- true, query_topic(Q,T), specialized(S,T).
special_topic(S,Q,T) :- true, inferred_topic(Q,T), specialized(S,T).

query_topic(Q,T) :- r80(Q,T,O,C,P), access(O,C,P,Q), selects(O,equal,T).
query_topic(Q,T) :- r81(Q,T,O,D,Y),
                    access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                    selects(O,equal,D), calender_year(D,Y), decade(Y,T).
query_topic(Q,T) :- r82(Q,T,O), access(O,T,s_Personalia,Q).

inferred_topic(Q,T) :- true, asks_fixed_range(Q,D1,D2),
                       decade_between(T,D1,D2).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), involved(P,M),
                       att_val(P,name,N), att_val(N,lastName,T).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), possible_genre(M,T),
                       not asks_fixed_genre(Q).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), involved(P,M),
                       possible_genre(P,T), not asks_fixed_genre(Q).
inferred_topic(Q,T) :- true, matchingPerson(Q,O,P), possible_genre(P,T),
                       not asks_fixed_genre(Q).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), release_bound(M,B,E),
                       decade(B,D1), decade(E,D2), decade_between(T,D1,D2),
                       not asks_fixed_date(Q).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), involved(P,M),
                       prod_period(P,B,E), calender_year(B,B1),
                       calender_year(E,E1), decade(B1,D1), decade(E1,D2),
                       decade_between(T,D1,D2),  not asks_fixed_date(Q).
inferred_topic(Q,T) :- true,  matchingPerson(Q,O,P), prod_period(P,B,E),
                       calender_year(B,B1), calender_year(E,E1),
                       decade(B1,D1), decade(E1,D2), decade_between(T,D1,D2),
                       not asks_fixed_date(Q).
```

```
asks_fixed_date(Q) :- r83(Q,O,D),
                      access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                      selects(O,equal,D).

asks_fixed_genre(Q) :- r84(Q,O,G), access(O,s_MovieDB,s_Movie_Genre,Q),
                       selects(O,equal,G).

asks_fixed_range(Q,D1,D2) :- r85(Q,D1,D2,O,V1,V2,Y1,Y2,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greater,V1), selects(O,less,V2),
                       +(V1,1,Y1), V2=Y2+1, calender_year(Y1,B),
                       calender_year(Y2,E), decade(B,D1), decade(E,D2).
asks_fixed_range(Q,D1,D2) :- r86(Q,D1,D2,O,V1,V2,Y1,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greater,V1), selects(O,lessequal,V2),
                       +(V1,1,Y1), calender_year(Y1,B), calender_year(V2,E),
                       decade(B,D1), decade(E,D2).
asks_fixed_range(Q,D1,D2) :- r87(Q,D1,D2,O,V1,V2,Y2,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greaterequal,V1), selects(O,less,V2),
                       +(Y2,1,V2), calender_year(V1,B), calender_year(Y2,E),
                       decade(B,D1), decade(E,D2).
asks_fixed_range(Q,D1,D2) :- r88(Q,D1,D2,O,V1,V2,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greaterequal,V1), selects(O,lessequal,V2),
                       calender_year(V1,B), calender_year(V2,E),
                       decade(B,D1), decade(E,D2).

irrelevant(S,Q) :- r90(S,Q,O,C,P), access(O,C,P,Q), -relevant(S,C).
irrelevant(S,Q) :- r91(S,Q,O,C,P), access(O,C,P,Q), -relevant(S,P).
irrelevant(S,Q) :- r92(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,equal,V), calender_year(V,Y), decade(Y,D),
                   -relevant(S,D).
irrelevant(S,Q) :- r93(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,greaterequal,V), calender_year(V,Y), decade(Y,D),
                   -relevant(S,D), irrelevant_after(S,D).
irrelevant(S,Q) :- r94(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,greater,V), calender_year(V,Y), decade(Y,D),
                   site(S), irrelevant_after(S,D).
irrelevant(S,Q) :- r95(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,lessequal,V), calender_year(V,Y), decade(Y,D),
                   -relevant(S,D), irrelevant_before(S,D).
irrelevant(S,Q) :- r96(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,less,V), calender_year(V,Y), decade(Y,D),
                   site(S), irrelevant_before(S,D).
irrelevant(S,Q) :- r97(S,Q,O,V,G), access(O,s_MovieDB,s_Movie_Genre,Q),
                   selects(O,equal,V), specialized(S,G), genre(G),
                   !=(V,G), !=(V,"Other").
```

```
irrelevant(S,Q) :- true, matchingAlive(Q,P), prod_period(P,X,Y),
                   calender_year(X,B), calender_year(Y,E), decade(B,D1),
                   decade(E,D2), irrelevant_range(S,D1,D2).
irrelevant(S,Q) :- true, matchingAlive(Q,P), life_period(P,B,E),
                   decade(B,D1), decade(E,D2),  irrelevant_range(S,D1,D2).

irrelevant_before(S,D) :- true, site(S), decade_name(D), !=(D,twenties),
                          not relevant_before(S,D).
irrelevant_after(S,D) :- true, site(S), decade_name(D), !=(D,twothousands),
                         not relevant_after(S,D).
irrelevant_range(S,D1,D2) :- true, site(S), decade_name(D1), decade_name(D2),
                             not relevant_between(S,D1,D2).
relevant_before(S,D) :- true, decade_before(D1,D), relevant(S,D1).
relevant_after(S,D) :- true, decade_after(D1,D), relevant(S,D1).
relevant_between(S,D1,D2) :- true, decade_between(D,D1,D2), relevant(S,D).


matchingAlive(Q,P) :- r100(Q,P,O,P1), instance(P,s_Person),
                      matchingPerson(Q,O,P), access(O,s_Actor,P1,Q).
matchingAlive(Q,P) :- r101(Q,P,O,P1), instance(P,s_Person),
                      matchingPerson(Q,O,P), access(O,s_Director,P1,Q).

matchingPerson(Q,O,P) :- r102(Q,O,P,L,O1,F,N),
                         access(O,s_Person,s_LastName,Q), selects(O,equal,L),
                         access(O1,s_Person,s_FirstName,Q),
                         selects(O1,equal,F), instance(P,s_Person),
                         att_val(P,name,N), att_val(N,lastName,L),
                         att_val(N,firstName,F).

matchingMovie(Q,M) :- r103(Q,M,O,T), access(O,s_MovieDB,s_Movie_Title,Q),
                      selects(O,equal,T), instance(M,s_Movie),
                      att_val(M,title,T).



%%%%%%%%%%%%%%%%%%
% Weak constraints
%%%%%%%%%%%%%%%%%%

:~ query_site(S,Q), charge(S,high). [3:3]
:~ query_site(S,Q), charge(S,medium). [2:3]
:~ query_site(S,Q), charge(S,low). [1:3]

:~ query_site(S,Q), reliable(S,low). [8:2]
:~ query_site(S,Q), reliable(S,medium). [4:2]

:~ query_site(S,Q), query_topic(Q,T),accurate(S,T,low). [15:1]
:~ query_site(S,Q), special(S1,Q), not special(S,Q). [12:1]
:~ query_site(S,Q), high_covered_topic(S1,Q,T), not decade_name(T),
                    not class0(T), not high_coverage(S,Q). [10:1]
:~ query_site(S,Q), query_topic(Q,T), not class0(T), not specialized(S,T). [8:1]
:~ query_site(S,Q), query_topic(Q,T), not class0(T), not covers(S,T,high). [7:1]
:~ query_site(S,Q), inferred_topic(Q,T), not decade_name(T),
                    not specialized(S,T). [6:1]
```

```
:~ query_site(S,Q), inferred_topic(Q,T), not decade_name(T),
                    not covers(S,T,high). [5:1]
:~ query_site(S,Q), default_class(O,T,Q), constructs(O,C,P),
                    specialized(S1,T), not specialized(S,T). [4:1]
:~ query_site(S,Q), default_class(O,T,Q), constructs(O,C,P),
                    covers(S1,T,high), not covers(S,T,high). [3:1]
:~ query_site(S,Q), high_covered_topic(S1,Q,T), class0(T),
                    not high_covered_topic(S,Q,T). [2:1]
:~ query_site(S,Q), high_covered_topic(S1,Q,T), decade_name(T),
                    not high_covered_topic(S,Q,T). [1:1]
```

## G Appendix: Experimental XML-QL Queries

Below, the XML-QL representation of all experimental queries is given, except for queries $Q_1$ and $Q_2$ which appear in Examples 1 and 7:

$Q_3$ : *In which year the movie 'Arsenic and Old Lace' has been released?*

```
FUNCTION MovieQuery03($MovieDB:"movie.dtd") {
  CONSTRUCT <ArsenicAndOldLace> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> "Arsenic and Old Lace" </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
        </Movie>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <released> $d </released>
  } </ArsenicAndOldLace> }
```

$Q_4$ : *In which movies directed by Alfred Hitchcock acted Marlene Dietrich?*

```
FUNCTION MovieQuery04($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <Director Personalia=$pid>
          </Director>
          <Actor Personalia=$pact>
          </Actor>
        </Movie>
```

```
        <Person ID=$pid>
          <FirstName.PCDATA>"Alfred"</>
          <LastName.PCDATA>"Hitchcock"</>
        </Person>
        <Person ID=$pact>
          <FirstName.PCDATA>"Marlene"</>
          <LastName.PCDATA>"Dietrich"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

$Q_5$ : *In which film noirs did Marilyn Monroe act?*

```
FUNCTION MovieQuery05($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="FilmNoir">
          <Title> $t </Title>
          <ReleaseDate> $d </ReleaseDate>
          <Actor Personalia=$pact>
          </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Marilyn"</>
          <LastName.PCDATA>"Monroe"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_6$ : *Which movies did Laurel and Hardy act in 1940?*

```
FUNCTION MovieQuery06($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact1> </Actor>
          <Actor Personalia=$pact2> </Actor>
        </Movie>
```

```
        <Person ID=$pact1>
          <FirstName.PCDATA>"Stan"</>
          <LastName.PCDATA>"Laurel"</>
        </Person>
        <Person ID=$pact2>
          <FirstName.PCDATA>"Oliver"</>
          <LastName.PCDATA>"Hardy"</>
        </Person>
      </MovieDB> IN source($MovieDB),
      $d = 1940
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_7$ : *Which movies acted by Frank Sinatra have a soundtrack by Elmer Bernstein?*

```
FUNCTION MovieQuery07($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
          <Soundtrack>
            <Composer Personalia=$pcomp> </Composer>
          </Soundtrack>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Frank"</>
          <LastName.PCDATA>"Sinatra"</>
        </Person>
        <Person ID=$pcomp>
          <FirstName.PCDATA>"Elmer"</>
          <LastName.PCDATA>"Bernstein"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_8$ : *When was James Dean born?*

```
FUNCTION MovieQuery08($MovieDB:"movie.dtd") {
  CONSTRUCT <JamesDean> {
     WHERE
      <MovieDB>
        <Person>
          <FirstName.PCDATA>"James"</>
          <LastName.PCDATA>"Dean"</>
          <BirthDate>
            <Date> $d </Date>
          </BirthDate>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <born> $d </born>
  } </JamesDean> }
```

$Q_9$ : *In which movies did Jane Russell and Marilyn Monroe act together?*

```
FUNCTION MovieQuery09($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <Actor Personalia=$pact1>
          </Actor>
          <Actor Personalia=$pact2>
          </Actor>
        </Movie>
        <Person ID=$pact1>
          <FirstName.PCDATA>"Jane"</>
          <LastName.PCDATA>"Russell"</>
        </Person>
        <Person ID=$pact2>
          <FirstName.PCDATA>"Marilyn"</>
          <LastName.PCDATA>"Monroe"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

$Q_{10}$ : *In which movies did Cary Grant and Marilyn Monroe act together?*

```
FUNCTION MovieQuery10($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <Actor Personalia=$pact1>
          </Actor>
          <Actor Personalia=$pact2>
          </Actor>
        </Movie>
        <Person ID=$pact1>
          <FirstName.PCDATA>"Cary"</>
          <LastName.PCDATA>"Grant"</>
        </Person>
        <Person ID=$pact2>
          <FirstName.PCDATA>"Marilyn"</>
          <LastName.PCDATA>"Monroe"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

$Q_{11}$ : *In which comedies did John Wayne act?*

```
FUNCTION MovieQuery11($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="Comedy">
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact>
          </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"John"</>
          <LastName.PCDATA>"Wayne"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_{12}$ : *Which movies acted by Sean Connery have a soundtrack by John Barry?*

```
FUNCTION MovieQuery12($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
          <Soundtrack>
            <Composer Personalia=$pcomp> </Composer>
          </Soundtrack>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Sean"</>
          <LastName.PCDATA>"Connery"</>
        </Person>
        <Person ID=$pcomp>
          <FirstName.PCDATA>"John"</>
          <LastName.PCDATA>"Barry"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_{13}$ : *In which horror movies of the sixties did Leonard Nimoy act?*

```
FUNCTION MovieQuery13($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="Horror">
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Leonard"</>
          <LastName.PCDATA>"Nimoy"</>
        </Person>
      </MovieDB> IN source($MovieDB),
      $d > 1959,
      $d < 1970
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_{14}$ : *In which animation movies did Vincent Price act?*

```
FUNCTION MovieQuery14($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="Animation">
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Vincent"</>
          <LastName.PCDATA>"Price"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_{15}$ : *In which movies, except crime movies, did James Cagney act between 1930 and 1940?*

```
FUNCTION MovieQuery15($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre=$gen>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"James"</>
          <LastName.PCDATA>"Cagney"</>
        </Person>
      </MovieDB> IN source($MovieDB),
      $gen != "Crime",
      $d > 1930,
      $d < 1940
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

$Q_{16}$ : *Where is Giorgio Moroder from?*

```
FUNCTION MovieQuery16($MovieDB:"movie.dtd") {
  CONSTRUCT <Moroder> {
     WHERE
      <MovieDB>
        <Person>
          <FirstName.PCDATA>"Giorgio"</>
          <LastName.PCDATA>"Moroder"</>
          <Country> $c </Country>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <from> $c </from>
  } </Moroder> }
```

$Q_{17}$ : *In which year the movie 'Witness for the Prosecution' has been released?*

```
FUNCTION MovieQuery17($MovieDB:"movie.dtd") {
  CONSTRUCT <WitnessForTheProsecution> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> "Witness for the Prosecution" </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
        </Movie>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <released> $d </released>
  } </WitnessForTheProsecution> }
```

$Q_{18}$ : *Which movies from 1999 have been rated above 90 points (out of 100)?*

```
FUNCTION MovieQuery18($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Review><Rating> $r </Rating></Review>
        </Movie>
      </MovieDB> IN source($MovieDB),
      $d = 1999,
      $r > 90
    CONSTRUCT
      <Movie> $t </Movie>
      <rated> $r </rated>
  } </MovieList> }
```