Journal of Universal Computer Science, vol. 14, no. 7 (2008), 1048-1060
submitted: 1/10/07, accepted: 21/1/08, appeared: 1/4/08 © J.UCS

Schema Mappings and Agents' Actions in P2P Data Integration System [1]

Grażyna Brzykcy
(Poznań University of Technology, Poland
grazyna.brzykcy@put.poznan.pl)

Jerzy Bartoszek
(Poznań University of Technology, Poland
jerzy.bartoszek@put.poznan.pl)

Tadeusz Pankowski
(Poznań University of Technology,
Adam Mickiewicz University, Poznań, Poland
tadeusz.pankowski@put.poznan.pl)

Abstract: We propose specification of schema mappings and agents' actions in XML data integration task. We discuss the problem in a highly-dynamic environment consisting of a community of peer-to-peer cooperating partners (agents). Peers decide how to describe their local data, when to join and when to leave the system, how to communicate and share their information with partners. An agent responds to the query by asking its partners (friends), which are able to partly answer the query. All the answers are merged and final result is constructed. A peer propagates a query along semantic paths existing in the system. Semantic paths are determined by schema mappings defined between partners. We propose a method for specifying schema mappings and to translate them to XQuery expressions. Mappings are represented by means of logical formulas. We also propose a declarative specification of semantic-driven communication in the system. The specification is made in a peer-oriented extension of Prolog.

Key Words: agent system, data integration, Prolog-like computations
Category: H.3.5, H.2,5, I.2.1

1 Introduction

Information integration plays the central role in building of large scale systems of P2P databases and a new generation of internet applications, where data comes from many different sources with different schemas [Bernstein et al. 2002, Calvanese et al. 2004, Tatarinov et al. 2004, Shvaiko et al. 2006].

In this paper we address the following issues that are crucial for semantic data integration in P2P environment: *(1) Schema mappings between peer's*

[1] The work was supported in part by the Polish Ministry of Science and Higher Education under Grant N516 015 31/1553.

schemas of local data repositories. Schema mappings are specifications describing how data structured under one schema (the source schema) is to be transformed into data structured under another schema (the target schema). Schema mappings can be used in *data exchange* and *query reformulation* tasks. In the former, the data is to be restructured using the mappings between schemas [Fagin et al. 2005, Areas et al. 2005, Pankowski 2006]. In the latter, a query formulated under the target schema is to be translated (reformulated) into a query under the source schema [Lenzerini 2002, Yu et al. 2004]. In both cases the semantics of the data should be preserved that enables semantic interoperability between peers [Bouquet et al. 2004]. *(2) Semantic communication between peers and between peers and the broker*. Semantic communication is the problem of carrying out the communication based on the agreed meaning of the task that is to be performed by cooperating peers (partners). In this paper, such tasks are queries that should be answered by a community of cooperating peers (agents).

The novel scientific contributions of this paper are as follows:

1. We propose a method for specifying schema mappings by means of tree-pattern formulas and show that such mappings can be translated into XQuery queries producing an instance of the target schema for a given instance of the source schema. The method extends ideas described in [Areas et al. 2005, Yu et al. 2004] and in our previous work [Pankowski et al. 2007].

2. We develop a consistent set of rules modeling semantic communication in the system of semantic data integration in P2P environment. We propose a declarative high-level notation to describe cooperation between agents (peers and the broker) involved in the process of data integration. The notation is based on a simple extension of Prolog, called *LogicPeer* proposed in [Loke 2006].

From the practical point of view, the logical representation of mappings may be used to check formal properties of them and to determine implied mappings (mappings compositions) that are not explicitly represented in the system [Meilicke et al. 2006]. The model of semantic communication supports a rapid prototyping of the system.

In Section 2 schema mappings are discussed. The agent-based architecture of the P2P semantic data integration system is proposed in Section 3. In Section 4 a declarative specification of semantic communication between agents is described. Section 5 concludes the paper.

2 Schema mappings

In Fig. 1 there are sample XML schema trees, S_1 and S_2, as well as their instances, where I_1 is an instance of S_1; I_2 and I_2' are two instances of S_2. The

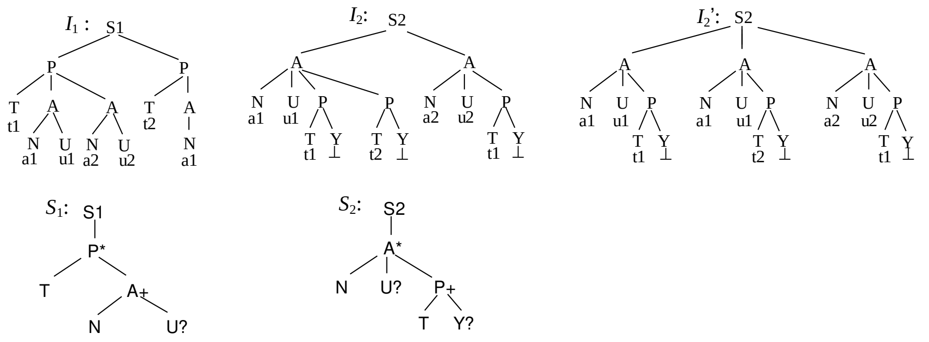

Figure 1: Sample XML schema trees and their instances

data represents the bibliographical data, where node labels are as follows: paper (P) and title (T) of the paper; author (A), name (N) and the affiliation (U) of the author; year (Y) of publication of the paper.

To establish correspondence between S_1 and S_2, we can specify the following schema mapping (1) from S_1 to S_2:

$$\begin{aligned} M_{1,2} := /S1/P[T = x_T \wedge A[N = x_N \wedge U = x_U]] \\ \Rightarrow /S2/A[N = x_N \wedge U = x_U \wedge P[T = x_T]] \end{aligned} \tag{1}$$

In general, schema mappings are specified by means of expressions of the form

$$\forall \mathbf{x}(\pi(\mathbf{x}) \Rightarrow\ /top/\sigma(\mathbf{x})), \tag{2}$$

where: (a) $\mathbf{x}$ is a tuple of text-valued source variables; (b) π and $/top/\sigma$ are tree-pattern formulas [Areas et al. 2005, Pankowski et al. 2007] conforming to the following syntax (L is a set of labels, $l \in L$, *top* is the outermost label):

$\pi ::= /\sigma; \quad \sigma ::= P[E]; \quad E ::= x \mid P = x \mid \sigma \mid E \wedge E; \quad P ::= l \mid P/l$

Transformation of a schema mapping expressed by the formula (2) into a query in XQuery ([W3C 2002]) can be described by the following rules:

$$Tr(\pi \Rightarrow /top/\sigma) :=\ <top>\{\tau_\pi(\pi)\ \textbf{return}\ \kappa_\sigma(\sigma)\}</top>$$

where:

$$
\begin{array}{ll}
1.\ \tau_\pi(/P[E], var_1) & = \textbf{for } \$var_1 \textbf{ in } /P, \\
 & \quad\quad \tau_E(var_1, E), \\
2.\ \tau_\sigma(var_1, P[E], var_2) & = \$var_2 \textbf{ in } \$var_1/P, \\
 & \quad\quad \tau_E(var_2, E), \\
3.\ \tau_E(var_1, x) & = x \textbf{ in } \$var_1/text(), \\
4.\ \tau_E(var_1, P = x) & = x \textbf{ in if } (var_1[P]) \textbf{ then } var_1/P/text() \textbf{ else } "null", \\
5.\ \tau_E(var_1, \sigma) & = \tau_\sigma(var_1, \sigma, var_2), \\
6.\ \tau_E(var_1, E_1 \wedge E_2) & = \tau_E(var_1, E_1), \\
 & \quad \tau_E(var_1, E_2),
\end{array}
$$

$$
\begin{array}{ll}
1.\ \kappa_\sigma(l[E]) & = <l>\kappa_E(E)</l> \\
2.\ \kappa_\sigma(l/P[E]) & = <l>\kappa_\sigma(P[E])</l> \\
3.\ \kappa_E(x) & = \{\$x\} \\
4.\ \kappa_E(l = x) & = <l>\{\$x\}</l> \\
5.\ \kappa_E(l/P = x) & = <l>\kappa_E(P = x)</l> \\
6.\ \kappa_E(\sigma) & = \kappa_\sigma(\sigma) \\
7.\ \kappa_E(E_1 \wedge E_2) & = \kappa_E(E_1) \\
 & \quad \kappa_E(E_2)
\end{array}
$$

Using these rules to the mapping $M_{1,2}$ we obtain the following query in XQuery language:

```
<S2> {
    for $v1 in doc("i1.xml")/S1/P,
        $t in if ($v1[T]) then $v1/T/text() else "null",
        $v2 in if ($v1[A]) then $v1/A else "null",
          $n in if ($v2[N]) then $v2/N/text() else "null",
          $u in if ($v2[U]) then $v2/U/text() else "null"
        return
        <A>
            <N>{ $n }</N>
            <U>{ $u }</U>
            <P>
              <T>{ $t }</T>
            </P>
        </A> }
</S2>
```

For the input XML document I_1, the query produces the result document I_2' (see Fig. 1) - the *canonical solution* to I_1 under the mapping $M_{1,2}$. Unknown values of Y ale replaced with null values denoted by $\perp$. In order to specify which of possible instances of the target schema should be produced, we can apply *automappings* over the schema. The automapping is a *key-pattern* formula and captures *keys* defined in the schema. Automappings and their applications to management of schema mappings, we have investigated in [Pankowski 2006, Pankowski et al. 2007].

Specification of schema mappings is a crucial problem in data integration systems. They are usually defined manually or using quasi-automating methods [Rahm et al. 2001]. Once introduced into the system, mappings can be used for many purposes. The most important application is data exchange and query reformulation in semantic data integration.

3 Overview of the peer-to-peer system for semantic data integration

We discuss a system built of autonomous agents (peers) each of which can independently decide how to structure its local data. The system, SIX-P2P, for semantic integration of XML data, is currently under development in Pozna University of Technology. Possibly heterogeneous semantic models that are developed by different peers are reflected by means of a peers' local schemas.

An agent in SIX-P2P system sees a set of another agents, its partners (peers), and may ask queries only to these partners. However, a query may be propagated to partners of each peer inducing a significant extension of the set of possible "knowledge sources". So, cooperative query evaluation is performed also by agents indirectly connected to the enquirer.

We make the following assumptions about agents in SIX-P2P system:

1. Each peer is identified by a unique name and represents some user. We do not impose any particular format for agent identifiers, but assume that system has its own namespace and name mapping mechanism.

2. An agent does not know all the agents of the system. A group of its partners (friends) is a subset of peers and their identifiers are known to the agent.

3. To join the system an agent has to introduce itself to a special agent - the broker. Process of registration consists in checking agent reliability (certification) and conveying to it a list of its partners.

4. To abandon the system an agent ought to let the broker know its action. This information is then passed by the broker to the adequate set of agents. Due to the system openness agents can join the system and abandon it, the appropriate partner lists should then be modified accordingly.

5. Each agent has a local collection of data and a schema of this data.

6. To posses new information an agent can generate queries to its partners. Conversely, asked by trustworthy peer an agent tries to answer the query, also by means of asking its own friends.

7. In the system agents can communicate by sending messages among each other and by using peer identifiers.

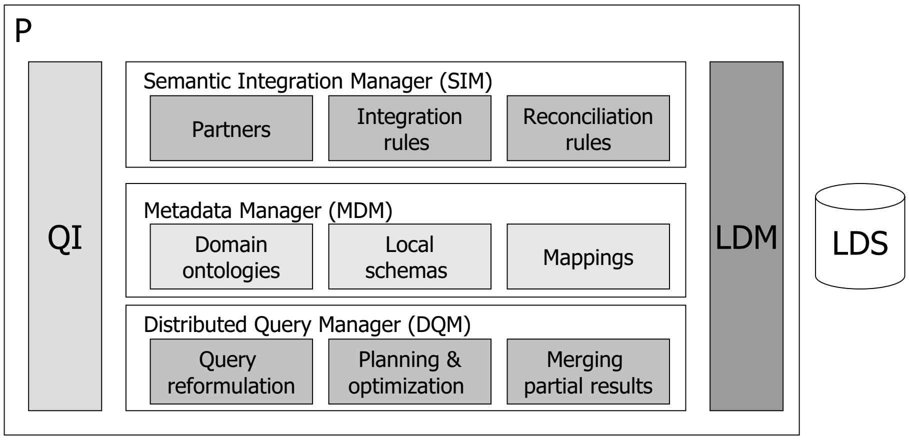

Figure 2: Architecture of a peer in SIX-P2P

In figure 2 the general architecture of a peer (P) in the system is depicted. Each peer has its own local data store (LDS) managed by a local management system (LDM). There is a query interface (QI) for accepting queries and returning answers during interactions with other peers. An important component of data integration system, a distributed query manager (DQM) is responsible for planning execution of a received query using P's own LDS and propagating the query to its partners. Partial results are merged and returned to the enquiring user. The metadata necessary to understand the query and to plan its execution are managed by metadata manager (MDM). Information about partners as well as rules defining integration strategy and reconciliation actions are managed by semantic integration manager (SIM).

There is a distinguished agent - broker in the SIX-P2P system (Fig 3). The broker is responsible for registration and certifications of the peers (R&C). This agent fulfils also some operations over all metadata existing in the system. The global metadata manager (GMM) can reason over schema mappings inferring compositions of mappings, inversions, or inconsistencies. It can map local ontologies and/or supports creation of the global ontology. For this aim global repositories of schemas, mappings and ontologies may be maintained.

The scenario of semantic data exchange between peers in P2P environment is shown in figure 4:

– Agent A wants to send query to B, so asks B for the schema S_B of its source data.

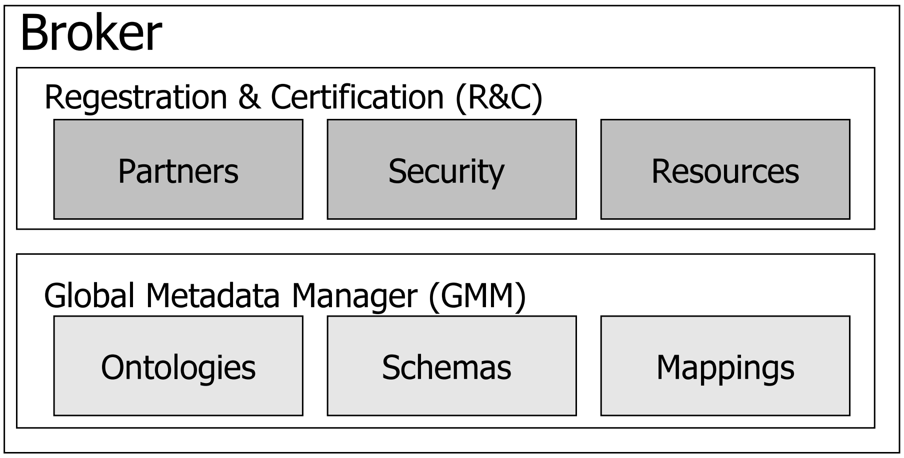

Figure 3: Architecture of the broker in SIX-P2P

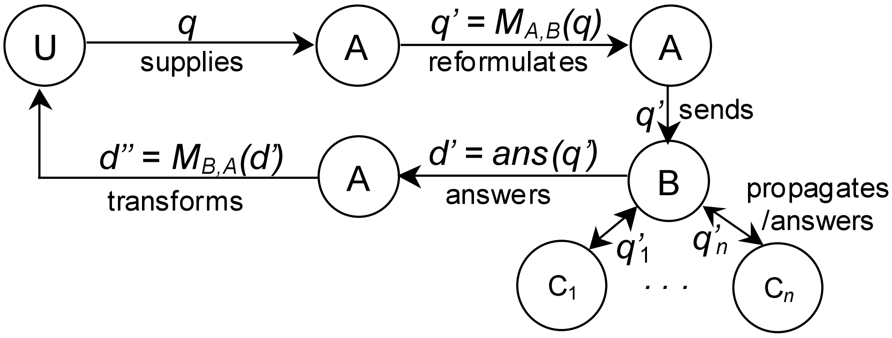

Figure 4: Sending and answering queries in P2P environments

– A creates (possibly with the help of the user) schema mapping $M_{B,A}$ to establish correspondences between schema S_B of an agent B and its own schema S_A. The user U can formulate a query q in terms of schema S_A (the target schema). Partial answer to q is obtained from local data stored in the peer of A, and some answer is also expected from B, so the query must be sent on to B.

– Because q is formulated in terms of S_A, it must be reformulated to a query q' over S_B. After reformulation, q' is sent to B.

- Similarly, agent B reformulates q' into $q'_1, ..., q'_n$ and propagates them to its partners $C_1, ..., C_n$, respectively.
- After receiving the answers from its partners, B merges them into data structured under S_B, and then answers query q'. The answer $d' = ans(q')$ is built on local data of an agent B and on replies of all its partners.
- In the last step d' and partial answers of other partners extend data of an agent A and the final reply $d'' = ans(q)$ is constructed.

4 Specification of data integration tasks

To specify the SIX-P2P system we apply the declarative programming paradigm and logic programming language Prolog because it provides a high level of abstraction for knowledge representation and processing. Prolog is used not only as a specification notation but is also aimed at development of expressive models. Moreover, Prolog is a rapid prototyping language with metaprogramming techniques, pattern matching and backtracking search. All the features are suitable to tackle the querying of integrated data and reasoning with semantics, problems that are the main objective of the Web and peer-to-peer systems.

In the paper we take advantage of a simple extension of Prolog, so called LogicPeer, which is presented in [Loke 2006], which is aimed at declarative programming of integrated peer-to-peer and Web based systems. In this computation model it is possible for an agent to direct a query (Prolog goal) to be evaluated at a specific peer. We can use goals: `PeerID * goal` where `PeerID` is an agent identifier. The special name `self` is reserved for a local peer.

We also employ the underlying protocol Gnutella as the mechanism for propagating Prolog goals among peers. Together with special control tags the protocol prevents loops between peers and supports optimization techniques of goal evaluation. It is worth to notice that a peer working as a propagator can convert a query into another query. With relation to replies, they are directed back along the same path as the query.

An agent's local data are represented as a collection of Prolog facts and rules. Some facts store details of architecture (e.g., partners, mappings) and rules define evaluation of locally initiated goals or queries received from other peers. So agents' actions are specified by rules.

There are two types of agents in SIX-P2P system: peers (Fig. 2) and brokers (Fig. 3). In terms of multi-agent systems an environment of each peer consists of its partners (with their schemas and data), requesting peers and a broker, whereas the broker sees all the peers (their identifiers). Each peer can perform at least three actions directed to a broker, namely introduction of an agent into the system (`introduce/3`), leaving the system by an agent (`leave/2`) and letting a broker know that an agent has modified its schema (`modify_schema/1`). Introduction of the `Agent` to the `Broker` consists of registration, which is an action executed by the `Broker`. In the specification a broker identifier prefixes the goal `register(Agent, Parts)`, and the action results in replying by the `Broker` a list of `Agent`'s partners. The list is stored by the `Agent` as a part of its metadata (see architecture in figure 2).

```
introduce(Agent, Broker, Parts) :-
  Broker * register(Agent, Parts), % registration is done by the broker
  assert(partners(Parts)).         % list of partners is stored as a fact
```

The broker remembers in Prolog-like database all registered agents in the system (fact `agents/1`)and all partners of these agents (broker's facts `partners/2`). It uses its own internal criteria to select partners of the new agent (action `choose/3`).

```
register(Agent, Partners):-
  agents(Agents),
  append(Agents, [Agent], Agents1),   % a new agent is added
  retract(agents(Agents)),            % the old set of agents is deleted
  assert(agents(Agents1)),            % the new set of agents is stored
  choose(Agent, Agents, Partners),    % partners are selected
  assert(partners(Agent, Partners)).  % partners of an agent are stored
```

An action of leaving the system by the `Agent` is very simple and contains only a message to the `Broker`.

```
leave(Agent, Broker) :-
  Broker * log_out(Agent).            % log_out is done by the broker
```

If an agent wants to log out, the broker informs about it all agents which may cooperate with this agent. Internal broker's data (i.e. the list of all registered agents, the list of agent's partners) is also updated.

```
log_out(Agent):-
  agents(Agents),
  remove(Agent, Agents, Agents1),     % the logged out agent is deleted
  retract(agents(Agents)),            % the old set of agents is deleted
  assert(agents(Agents1)),            % the new set of agents is stored
  set_of(A, (partners(A, Partners),   % agents which cooperate with
  member(Agent, Partners)), Agents2), % the deleted agent are selected
  l_inform_all(Agents2, Agent),       % and informed
  retract(partners(Agent, _)),        % the logged out agent is deleted
  p_remove(Agent).                    % from partners facts
```

The broker checks all lists of partners and sends messages `logged_out/1` to all agents cooperating with the removed one.

```
l_inform_all([ ], _).
l_inform_all([A | As], Agent):-
  A * logged_out(Agent),             % a message is sent to the agent A
  l_inform_all(As, Agent).
```

The logged out agent has to be also deleted from broker's database. To perform this action all the `partners` facts are scanned and updated by the `Broker`.

```
p_remove(Agent):-
  partners(A, Partners),
  member(Agent, Partners),        % cooperators of an agent are selected
  remove(Agent, Partners, Partners1),
  retract(partners(A, Partners)),  % partners facts of the logged out
  assert(partners(A, Partners1)),  % agent are updated
  fail.
p_remove(_).
```

When the Agent determines to restructure its data it has to modify its local schema (modify_schema/3) locally and to inform via Broker other peers to which the Agent is a partner. The local modification done by itself (modify/2) completes when a new schema replaces the old one in a store (Local schemas in Fig. 2) and new mappings (create_maps/1) would be defined.

```
modify_schema(Agent, Broker, Snew) :-
  modify(Snew),                    % agent modifies schema locally
  Broker * modified_schema(Agent). % broker is informed about modification
modify(Snew) :-
  retract(schema(self, Sold),      % the old schema is removed
  assert(schema(self, Snew),       % the new schema is stored
  create_maps(Maps).               % new mappings are created
```

In the system there is an action initiated by a broker and executed as a consequence of receiving an agent's message about schema modification. The action modified_schema depends on updating of broker's data and sending appropriate messages to all the interested agents in the system.

```
modified_schema(Agent) :-
  set_of(A, (partners(A, Parts),    % agents which cooperate with
  member(Agent, Parts)), Agents2),  % the agent are selected
  m_inform_all(Agents2, Agent),     % and informed
```

The Broker informs about schema modification reported by the Agent passing modified_schema message to cooperators of the Agent.

```
m_inform_all([ ], _).
m_inform_all([A | As], Agent):-
  A * modified_schema (Agent),      % a message is sent to the agent A
  m_inform_all(As, Agent).
```

To completely describe interactions between broker and other peers in the system we need to specify peers' reactions on broker's messages logged_out and modified_schema. The action (logged_out/1), performed by a peer, consists in updating the list of partners,if one of the partners, the Agent, has just logged out from the system.

```
logged_out(Agent):-
  partners(Partners),
  remove(Agent, Partners, Partners1),  % the agent is removed
  retract(partners(Partners)),         % the partners facts are updated
  assert(partners(Partners1)).
```

When a peer receives the message `modified_schema`, in which broker informs about schema modification reported by the `Agent`, then the old `schema` facts and the old mappings are removed from the Agent's metadata. At the same time the `Agent` is asked about its new schema and a new mapping is created.

```
modified_schema(Partner):-
  retract(schema(Partner, _)),        % the old schema is removed
  retract(mapping(Partner, _)),       % the old mappings are removed
  Partner * schema(self, Schemap),    % partner is asked about  schema
  assert(schema(Partner, Schemap)),   % the new schema is stored
  schema(self, Schemaa),
  map(Schemap, Schemaa, Mpa),         % the new mapping is created
  assert(mapping(Partner, Mpa)).      % mapping is stored as a fact
```

Before asking any partner an agent has to prepare suitable mappings between schemas. The `Agent` tries to build mappings for all the partners (`create_map/3`). Since it is not assured that such a mapping exists (is constructed) the special value `null` is chosen to depict the situation. A mapping from the `Partner` is denoted as `Mpa`.

```
create_maps(Maps) :-
  partners(Partners),              % agent's partners
  create_pmaps(Partners, Maps).    % mappings from the partners
create_pmaps([], _).               % all mappings are created
create_pmaps([P|Partners],[ Mpa|Maps]):-
  create_map(P, Mpa),              % mapping from P is created
  create_pmaps(Partners,Maps).
create_map(Partner, Mpa) :-
  schema(self, Schemaa),           % agent's schema
  Partner * schema(self, Schemap), % schema is taken from the partner
  map(Schemap, Schemaa, Mpa),      % the new mapping is created
  assert(mapping(Partner, Mpa)).   % mapping is stored as a fact
```

A process of passing queries and receiving answers (see Fig. 4) is specified as an action (`ask/2`). It has the following steps: asking partners (`ask_partners/2`), merging their answers (`merge/3`) with the local data and answering the query locally (`query/2`).

```
ask(Query, Answer) :-
  ask_partners(Query, Answers),       % partners are queried
  merge(Answers),                     % answers are merging
  query(Query, Answer).               % query is answered locally
```

An action of asking the Agent's partners is proceeded by selection (`select/1`) of qualified partners - those for which mappings are constructed.

```
ask_partners(Query, Answers) :-
  select(QPartners),           % qualified partners are chosen
  ask_qpartners(Query, QPartners, Answers).
                               % subset of qualified partners is queried
select(Qpartners) :-
  set_of(Partner,(mapping(Partner, Mpa), Mpa ? null), QPartners).
                               % only not null mappings are considered
ask_qpartners(_, [ ], _).      % all the partners are asked
ask_qpartners(Query, [P|Ps], [A|As]) :-
  ask_qpartner(Query, P, A),   % partner P is queried
  ask_qpartners(Query, Ps, As).
```

To use partner's data in a process of query answering an agent has to rewrite (`rewrite/3`) the original `Query` into more specialized query `Qp`, directed to the suitable partner `P`.

```
ask_qpartner(Query, Partner, Answer) :-
  mapping(Partner, Mpa),
  rewrite(Query, Mpa, Qp),     % query is rewritten due to the mapping
  Partner * ask(Qp, Answer).   % partner answers the query
```

5 Conclusions

We discuss some theoretical problems related to semantic data integration in peer-to-peer systems. We focus ourselves on (1) logical specification of schema mappings between local data repositories managed by a particular peer (agent) and (2) declarative specification of semantic-driven communication in the data integration processes, particularly between agents and between agents and the broker. The considerations are based on the implementation of the SIX-P2P system, currently under development in Poznań University of Technology, where the discussed methods are under verification and evaluation.

References

[Areas et al. 2005] Arenas, M., Libkin, L.: XML Data Exchange: Consistency and Query Answering, *PODS Conference*, 2005, 13–24.

[Bernstein et al. 2002] Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data Management for Peer-to-Peer Computing : A Vision, *WebDB*, 2002, 89–94.

[Bouquet et al. 2004] Bouquet, P., Serafini, L., Zanobini, S.: Peer-to-peer semantic coordination, *Journal of Web Semantics*, **2**(1), 2004, 81–97.

[Calvanese et al. 2004] Calvanese, D., Giacomo, G. D., Lenzerini, M., Rosati, R.: Logical Foundations of Peer-To-Peer Data Integration, *Proc. of the 23rd ACM SIGMOD Symposium on Principles of Database Systems (PODS 2004)*, 2004, 241–251.

[Fagin et al. 2005] Fagin, R., Kolaitis, P. G., Popa, L.: Data exchange: getting to the core, *ACM Trans. Database Syst.*, **30**(1), 2005, 174–210.

[Lenzerini 2002] Lenzerini, M.: Data Integration: A Theoretical Perspective, *PODS*, 2002, 233–246.

[Loke 2006] Loke, S. W.: Declarative programming of integrated peer-to-peer and Web based systems: the case of Prolog, *J. of Systems and Software*, **79**(4), 2006, 523–536.

[Meilicke et al. 2006] Meilicke, C., Stuckenschmidt, H., Tamilin, A.: Improving Automatically Created Mappings Using Logical Reasoning, *Proceedings of the 1st International Workshop on Ontology Matching OM-2006*, CEUR Workshop Proceedings Vol. 225, http://CEUR-WS.org/Vol-225.

[Pankowski 2006] Pankowski, T.: Management of executable schema mappings for XML data exchange, *Database Technologies for Handling XML Information on the Web, EDBT 2006 Workshops*, Lecture Notes in Computer Science **4254**, Springer, 2006, 264–277.

[Pankowski et al. 2007] Pankowski, T., Cybulka, J., Meissner, A.: XML Schema Mappings in the Presence of Key Constraints and Value Dependencies, *ICDT 2007 Workshop EROW'07*, CEUR Workshop Proceedings Vol. 229, CEUR-WS.org/Vol-229, 2007, 1–15.

[Rahm et al. 2001] Rahm, E., Bernstein, P. A.: A survey of approaches to automatic schema matching, *The VLDB Journal*, **10**(4), 2001, 334–350.

[Shvaiko et al. 2006] Shvaiko, P., *et al.*: Dynamic Ontology Matching: A Survey, Techn. Report DIT-06-046, University of Trento, Available on: http://eprints.biblio.unitn.it/ archive/00001040/, 2006.

[Tatarinov et al. 2004] Tatarinov, I., Halevy, A. Y.: Efficient Query Reformulation in Peer-Data Management Systems, *SIGMOD Conference*, 2004, 539–550.

[W3C 2002] XQuery 1.0: An XML Query Language. W3C Working Draft: 2002. www.w3.org/TR/ xquery

[Yu et al. 2004] Yu, C., Popa, L.: Constraint-Based XML Query Rewriting For Data Integration, *SIGMOD Conference*, 2004, 371–382.