Configuring Intelligent Mediators using Ontologies

Gennaro Bruno, Christine Collet, and Genoveva Vargas-Solar

LSR-IMAG Laboratory
681, rue de la passerelle
38402, St. Martin d'Hères
France
{Gennaro.Bruno, Christine.Collet, Genoveva.Vargas-Solar}@imag.fr
http://www-lsr.imag.fr/

Abstract. This paper presents a new intelligent mediators configuration approach which exploits high expressive description logics to represent metadata, and reasoning tasks in order to build more flexible mediation systems. A user specifies a *needs expression* in terms of (i) an interesting view over a given application domain, (ii) sources preferences and (iii) architectural requirements. A well-adapted mediator, is automatically configured according to these needs through a reasoning-based configuration process. A configured mediator can therefore be adapted in order to build knowledge-based mediation systems with an arbitrary architecture.

1 Context and Motivations

Mediation systems [1] were introduced to provide an integrated view over distributed and heterogeneous data sources for accessing them in a transparent way. During these last years, their role has constantly evolved. Several mediation approaches, providing different modeling and implementing solutions, have been proposed.

In order to provide query expression and metadata management with more semantics, a particular kind of data integration approach, commonly called *knowledge-based* mediation system, has been proposed. Differently from classical mediation systems, they use high expressive knowledge representation formalisms, i.e., description logics, as basis for data integration. This allows to have a more precise semantic representation of application domains, and to improve classical mediation tasks with inference capabilities.

This work focuses on knowledge-based mediation systems. For this reason, we analyzed many existing approaches according to several aspects such as the integration approach, data model and associated query language, and more particularly the mediation system architecture. According to this latter aspect, knowledge-based mediation systems can be mainly divided into two main categories (cf. Figure 1):

– *centralized mediation systems* [2, 3, 5, 7, 8, 6] are based on a *domain ontology* acting as an integrated view over a set of distributed and heterogeneous data sources. A user formulates a query over the domain ontology. Then the query is rewritten into a set of local expressions over local sources, which are consequently accessed in a transparent way. The mediator represents the single access point to the system, and local sources are directly accessible from it.
– *distributed mediation systems* [4, 9] aim to integrate a very large number of distributed data sources. This makes the construction of an integrated view over them a very difficult task to achieve. Therefore, query processing becomes a distributed task and the centralized mediator is replaced by a net of cooperative components commonly called *peers*. Each peer provides a *local ontology* modeling one or more underlying local sources. A user formulates a query over a peer. If locally retrieved data does not fulfill user expectatives, the query is forwarded to some neighbors peers[1] for execution in order to retrieve mode data.

To our knowledge, no mediation system, being able to adapt to both applicative contexts, exists today.

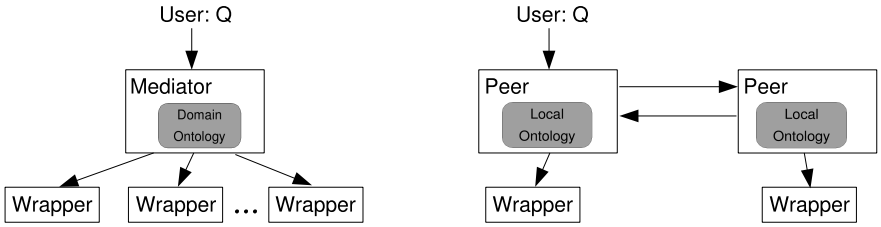

Fig. 1. Existing approaches.

This paper focuses on the intelligent mediator configuration process of ADEMS[2] and on the role of mediators within a knowledge-based mediation system. The architecture of ADEMS has been previously presented in [10], therefore, the paper gives no details on the mediator internal architecture and on query expression and processing.

ADEMS exploits the high expressive description logic $\mathcal{SHIQ}(\mathcal{D})$ [12] to represent metadata, and exploits reasoning tasks in order to automatically configure well-adapted mediators. A user specifies a *needs expression* in terms of (i) the interesting view over a given application domain, (ii) sources preferences and, (iii) architectural requirements. ADEMS configures a well-adapted mediator according to these needs, being able to adapt centralized as well as distributed

[1] Neighbor peers are those ones that a peer can directly access. Differently from centralized approach, not all resources are directly accessible from a peer (mediator).
[2] ADEMS, an *ADaptable and Extensible Mediation Service*.

architectures. A configured mediator manages metadata as knowledge within a set of ontologies, and exploits inference in order to semantically improve the query processing task. Nevertheless, for a lack of space, this paper mainly focuses on the mediator configuration process, and on the role of mediators within a knowledge-based mediation system. No details on the mediator internal architecture and functions are given.

The remainder of this document is organized as follows. Section 2 presents the ADEMS approach. It describes the general architecture of a mediation system, its components, i.e. a set of mediators and sources, and the way they interact. Then it introduces the mediator configuration process. Section 3 illustrates the needs expression structure and shows how its ontological representation allows to better represent the semantics of metadata. Section 4 shows how a needs expression is analyzed and a mediator is configured accordingly, by exploiting reasoning tasks. Metadata involved in such a process is also illustrated. Section 5 discusses on implementation issues and experimental validations. Finally, Section 6 concludes the paper.

2 Approach

In our approach, a mediation system consists of a net of interconnected mediators giving access to a set of heterogeneous and distributed local sources (cf. Figure 2). Each mediator corresponds to a user access point to the mediation system. No assumptions about the system topology are done. The system architecture is not fixed a priori and it is adaptable to different applicative contexts/user requirements.

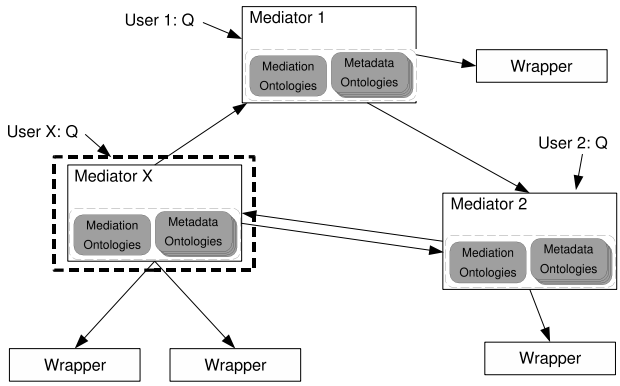

Fig. 2. ADEMS mediation system.

For doing so, a mediator is modeled a general-purpose reusable mediation component. It becomes a specific ad-hoc component through a *configuration process*, whose goal is to adapt a mediator to a particular user needs definition. The

mediator configuration process consists of building a set of ontologies describing the behavior of a mediator: a (i) *mediation ontology* represents a user-defined view over a domain description and acts as a global schema (integrated or not) over a set of underlying local *resources* (sources as well as other mediators), and (ii) a set of *metadata ontologies* representing all necessary metadata about available resources, the data they manage and the way to access them. Therefore, from a *user X* point of view (cf. dashed box in the Figure 2), a mediation system consists of a *mediator X* accessing a set of resources. A user formulates a query over its mediation ontology and the query is evaluated over the mediator resources in a transparent way. The user perception of the whole mediation system is limited to his/her own configured mediator, representing his/her access point to data. We will show later in this paper how, according to the way a mediator is configured, our approach allows to emulate both centralized and distributed knowledge-based mediation systems, and to enable more complex architectures.

Configuring a mediator in such a way is a difficult and tedious task for a human operator. For this reason, in order to help users with this task, we propose the ADEMS *mediator configuration service* (cf. Figure 3). The service manages

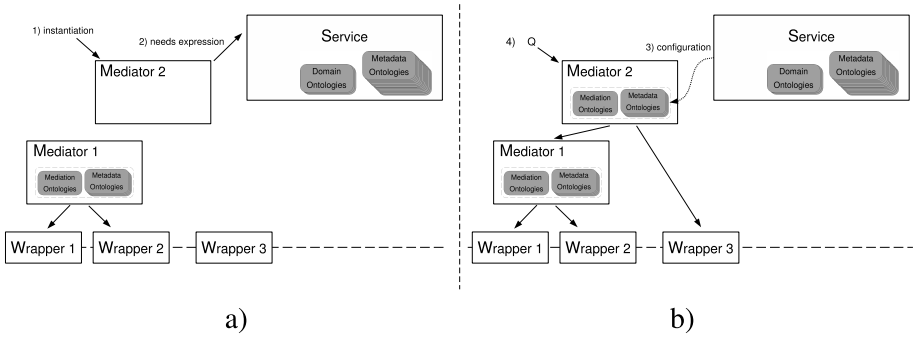

Fig. 3. The ADEMS approach.

all necessary metadata as knowledge within a set of ontologies. For each application domain, the service manages (i) a *domain ontology*, modeling its entities and acting as a shared vocabulary, e.g., bioinformatics, and (ii) a set of *metadata ontologies* representing all necessary metainformation to access available local sources for a domain, e.g., mappings and sources capabilities. A user specifies a *needs expression* (cf. Fig 3-a) in terms of an interesting domain, and a view over it, architectural and source requirements. The service analyzes the user-defined needs expression and exploits reasoning tasks to extract pertinent metadata to configure the mediator (cf. Fig 3-b): the mediation ontology is built as a view over the selected domain ontology, and metadata ontologies as a subset of metadata ontologies within the service.

3 Needs expression

A needs expression is modeled as a complex concept definition within the *needs expression ontology*. This is the most important metadata ontology of the service as it plays a key role during the whole mediator configuration process. This ontology is built around a main central concept Need, representing a whole needs expression. The goal of this ontology is to exploit reasoning tasks to classify needs expressions in order to deduce containment relations, and to exploit this knowledge to discover when a mediator can exploit another mediator as a possible resource. For this reason, metadata in this ontology is manipulated at the intensional level, i.e., classes. Each new needs expression is represented in this ontology as a new subclass of Need. The satisfiability and subsumption verifications can then be exploited, to verify all needs expressions consistency and to classify them.

A needs expression is composed of three main parts, that we call *metadata categories*. Each category represents a set of *metadata aspects*. The concept Need is defined as follows:

$$\begin{aligned} \mathsf{Need} \equiv\ & \exists \mathit{hasConceptSet}.\mathsf{ConceptSet} \sqcap \\ & \exists \mathit{hasArchitecture}.\mathsf{Architecture} \sqcap \\ & \exists \mathit{hasSourcePreference}.\mathsf{SourcePreference} \end{aligned}$$

where the three main categories are: ConceptSet, representing the interesting view the domain; SourcePreference and Architecture, specifying source preferences and architectural requirements respectively.

Given a new needs expression, a new class, representing it, is defined as follows:

$$\begin{aligned} \mathsf{Need\_i} \equiv\ & \exists \mathit{hasConceptSet}.\mathsf{ConceptSet\_i} \sqcap \\ & \exists \mathit{hasArchitecture}.\mathsf{Architecture\_i} \sqcap \\ & \exists \mathit{hasSourcePreference}.\mathsf{SourcePreference\_i} \end{aligned} \tag{1}$$

where, Need_i is deduced to be subclass of Need, and classes ConceptSet_i, Architecture_i and SourcePreference_i represent values for each category. In the remainder of this section, we present details on the three needs expression categories: ConceptSet, Architecture and SourcePreference.

3.1 ConceptSet

A user defines an interesting view over a domain by selecting a set of concepts from the corresponding domain ontology. An algorithm (not shown here) analyzes selected concepts and generates a mediation ontology accordingly. In order to classify needs, in our approach, a whole view over an application domain is represented by a single concept definition called *concept set*. A concept set is defined as follows:

$$\mathsf{ConceptSet\_i} \equiv \mathsf{C}_1 \vee ... \vee \mathsf{C}_n$$
$$\mathsf{ConceptSet\_i} \sqsubseteq \mathsf{ConceptSet}$$

Clearly, this concept must be explicitely defined as a subclass of ConceptSet in order to exploit reasoning capabilities. Given two concept sets ConceptSet_i and ConceptSet_j defined as follows:

$$\mathsf{ConceptSet\_i} \equiv \mathsf{C}_1 \vee ... \vee \mathsf{C}_{n-1} \vee \mathsf{C}_n$$
$$\mathsf{ConceptSet\_j} \equiv \mathsf{C}_1 \vee ... \vee \mathsf{C}_{n-1}$$

the subsumption relation ConceptSet_j $\sqsubseteq$ ConceptSet_i is interpreted as a containment relation between their corresponding views. This motivates the use of a union of concepts to represent this aspect.

By default, when a domain concept, e.g., Person, is added to a concept set, it is considered with all its associated attributes. However, it is possible to explicitely specify a view over a class in order to consider interesting attributes only. For this purpose, our model provides a special property definition *_without* to specify a restriction over an atomic concept. In order to conserve a meaningful subsumption relation between concept sets, this property restriction models attributes that are not taken into account in the view. Here is an example:

$$\begin{aligned}\mathsf{ConceptSet\_i} &\equiv \mathsf{C}_1 \\ \mathsf{ConceptSet\_j} &\equiv \mathsf{C}_1 \wedge \exists\_\textit{without}.\mathsf{ATT}_1 \wedge \\ &\quad ... \wedge \exists\_\textit{without}.\mathsf{ATT}_n\end{aligned}$$

Subsumption verification allows to infer that ConceptSet_j is subclass of ConceptSet_i. Clearly, the subsumption relation between concept sets reflects the containment relation between their respective mediation ontologies.

More complex views may so be defined. Here is an example:

$$\begin{aligned}\mathsf{ConceptSet\_1} &\equiv \mathsf{Person} \vee \mathsf{Car} \vee \mathsf{Job} \\ \mathsf{ConceptSet\_2} &\equiv \mathsf{Person} \vee \mathsf{Car} \\ \mathsf{ConceptSet\_3} &\equiv (\mathsf{Person} \wedge \exists\_\textit{without}.\mathsf{Age}) \vee \mathsf{Car} \\ \mathsf{ConceptSet\_4} &\equiv (\mathsf{Person} \wedge \exists\_\textit{without}.\mathsf{Age} \wedge \\ &\quad \exists\_\textit{without}.\mathsf{Gender}) \vee \mathsf{Car}\end{aligned}$$

Reasoning tasks allow to deduce the following subsumption relation between views:

$$\mathsf{ConceptSet\_4} \sqsubseteq \mathsf{ConceptSet\_3} \sqsubseteq \mathsf{ConceptSet\_2} \sqsubseteq \mathsf{ConceptSet\_1}$$

3.2 Architecture

Architectural aspects are modeled as follows:

$$\begin{aligned}\mathsf{Architecture} \equiv\ & \exists \mathit{hasImportDegree}.\mathsf{ImportDegree} \sqcap \\ & \exists \mathit{hasExportDegree}.\mathsf{ExportDegree} \sqcap \\ & \exists \mathit{hasMatDegree}.\mathsf{MatDegree} \sqcap \\ & \exists \mathit{hasIntDegree}.\mathsf{IntDegree}\end{aligned}$$

where Architecture represents a category containing the four aspects ImportDegree, ExportDegree, MatDegree and IntDegree.

ImportDegree : this dimension defines the possible strategies to adopt when exploiting other running mediators as resources. We refer to this feature as resource *import*. Each strategy is enabled by one of the following values:

- ImportAll: the mediator may import any pertinent resource, independently of its ownership. This value is modeled as an atomic class.
- ImportGroup: specifies the group of trusted users/owners from which mediators can be imported. Any specified group is modeled as a subclass of ImportGroup.
- ImportNone: specifies that no mediator is imported. This value corresponds to an atomic class definition.

According to the semantic of this aspect, in order to guarantee the containment relation between two needs, the following subsumption relations must be stated:

$$\mathsf{ImportNone} \sqsubseteq \mathsf{ImportGroup} \sqsubseteq \mathsf{ImportAll} \tag{2}$$

In order for a mediator M_1 to be potentially imported by M_2, the group of users specified by the M_1's import degree must be contained by the one of M_2. In other words, M_2 cannot import M_1 if M_1's import degree contains at least a user which is not considered in M_2's import degree, as this will violate the M_2 strategy.

These considerations motivate the use of union of classes to model a group of trusted users in the import degree. Here is an example:

$$\mathsf{User\_1} \sqcup \mathsf{User\_2} \sqsubseteq \mathsf{ImportGroup} \tag{3}$$

$$\mathsf{ImportNone} \sqsubseteq \mathsf{User\_1} \sqcup \mathsf{User\_2} \tag{4}$$

Statement (3) specifies a new group composed by User_1 and User_2. This is done by stating the union of classes as a subclass of ImportGroup class. Statement (4) guarantees the containment relation between any group and the empty one.

Figure 4-a shows an example of import groups classification. In this example, eight groups have been specified, and each of them corresponds to a different

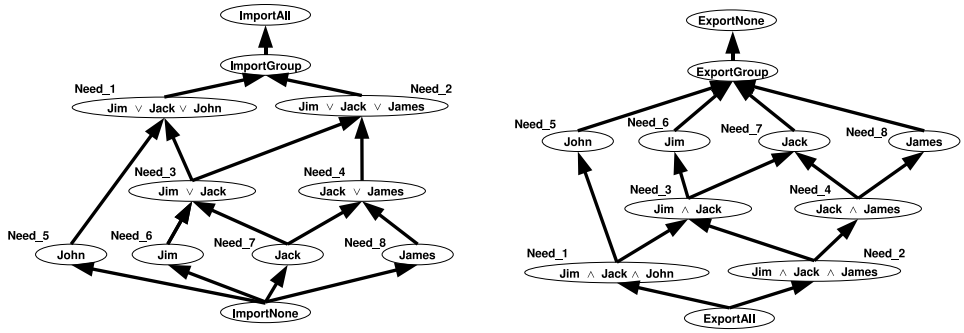

Fig. 4. Import and export degrees classification.

independently defined needs expression. Let us suppose that a user wants to retrieve all resources which may be imported by a mediator configured according the need Need_1. As Need_1 imports resources from users Jim, Jack and John, only needs Need_3, Need_5, Need_6 and Need_7 can be normally be imported. Modeling import groups as union of classes allows to retrieve importable resources for a mediator by asking for synonyms and descendants of the class representing its import degree value.

ExportDegree : this dimension specifies the reusability degree of a new defined mediator in the context of future user needs expressions. We refer to this feature as resource *export*. Three strategies are allowed and represented by the following values:

- ExportNone: specifies that no further defined mediator can reuse this mediator in the future. Its values correspond to an atomic class.
- ExportGroup: specifies the group of users that can reuse the mediator in future configurations. Any specified group is modeled as subclass of ExportGroup.
- ExportAll: specifies that any further defined mediator can reuse this mediator in the future. No user restriction are given.

In order to respect the needs containment according to this aspect, the following subsumption relations must be respected:

$$\mathsf{ExportAll} \sqsubseteq \mathsf{ExportGroup} \sqsubseteq \mathsf{ExportNone} \tag{5}$$

In order for a mediator M_1 to be importable by M_2, the set of users specified by the M_1's export degree must contain the one of M_2. In other words, M_2 cannot import M_1 and, at the same time, export itself to other user than the ones allowed for M_1, because this will violate the M_1 strategy. Consequently, M_1's export degree must be subclass of the M_2's one, what justifies the use of a

conjuction of classes to represent the export degree value. A group is defined as the conjuction of classes representing user names. Here is an example:

$$\mathsf{User\_1} \sqcap \mathsf{User\_2} \sqsubseteq \mathsf{ExportGroup} \tag{6}$$

$$\mathsf{ExportAll} \sqsubseteq \mathsf{User\_1} \sqcap \mathsf{User\_2} \tag{7}$$

Statement (6) specifies a new group composed by User_1 and User_2. This is done by stating the conjunction of classes as a subclass of ExportGroup class. Statement (7) guarantees the containment relation between any group and the whole set of users.

Figure 4-b shows an example of export groups classification. In this example, eight groups have been specified, and each of them corresponds to a different independently defined needs expression. Let us suppose that a user wants to retrieve all resources which may be imported by a mediator configured according the need Need_3. As Need_3 exports to users Jim and Jack, only needs Need_1 and Need_2 can be normally be imported. Similarly to the import degree aspect, modeling an export group as a conjunction of classes allows to retrieve importable resources for a mediator by asking for synonyms and descendants of the class representing its export degree value.

Notice that, differently from the ImportDegree, the ExportDegree must necessarily contain the name of the user defining the need. This is motivated by the fact that a user always exports to himself his own resources. However, a user does not necessary import resources from himself.

MatDegree : this dimension specifies the materialization strategy to adopt within a mediation system. This dimension may take the following two values, modeled as atomic classes:

- Materialized: specifies that the materialization is allowed in all mediators composing the mediation system. Retrieved data can be materialized at the mediator level as well as into its imported resources. When data freshness is not a priority, this allows for a faster data retrieval.
- NoMaterialized: specifies that no materialization is allowed. This corresponds to a fully virtual approach and only non-materializing resources will be imported.

In order to respect the needs containment, the following subsumption relation is stated:

$$\mathsf{NoMaterialized} \sqsubseteq \mathsf{Materialized}$$

stating that a materialized mediator can import materialized as well as non-materialized resources, but not viceversa.

IntDegree : this dimension specifies the integration degree for the mediator, which may essentially take two possible values:

- Integrated: specifies that the mediator provides the user with an integrated global representation over underlying resources. This latter corresponds to the mediation ontology which is built as a subgraph of the domain ontology. A mediator configured in such a way can import integrated as well as non-integrated resources. This strategy is typical adopted in approaches providing a transparent access to underlying local sources, e.g. centralized mediation systems.
- Non-integrated: specifies that the mediator provides the user with a non-integrated global representation. The mediation ontology consists of a set of local ontologies. This feature can be interesting for expert users, which prefer to deal with local sources, or in highly distributed mediation systems, where no integrated global representation is required. In this approach, only non-integrated resources can be imported. This feature enables peer-to-peer mediation systems.

In order to respect the need containment, the following subsumption relation is stated:

$$\text{Non-integrated} \sqsubseteq \text{Integrated}$$

stating that an integrated mediator can import non-integrated as well as integrated resources, but not viceversa.

Given a new needs expression definition Need_i, a new class Architecture_i is defined. Here is an example:

$$\begin{aligned}\text{Architecture\_i} \equiv\ & \exists \textit{hasImportDegree}.(\text{Jim} \ \sqcup \ \text{Jack}) \sqcap \\ & \exists \textit{hasExportDegree}.\text{Jim} \sqcap \\ & \exists \textit{hasMatDegree}.\text{Materialized} \sqcap \\ & \exists \textit{hasIntDegree}.\text{Integrated}\end{aligned}$$

stating that the corresponding mediator imports previously defined mediators from users *Jim* and *Jack*, but can be reused only by *Jim*. Materialization is allowed and the mediator provides an integrated view over underlying resources.

3.3 Source Preference

Source preferences are modeled in the needs expression ontology as follows:

$$\begin{aligned}\text{SourcePreference} \equiv\ & \exists \textit{hasQuality}.\text{Quality} \sqcap \\ & \exists \textit{hasAvailability}.\text{Availability} \sqcap \\ & \exists \textit{hasCost}.\text{Cost}\end{aligned}$$

where Quality, Availability and Cost represent three metadata sub-categories.Notice that these aspects are given for explanation purpose. We do not propose in this paper a source annotation model. Our goal is to show how to apply ontologies and reasoning tasks to model such kind of metadata and deduce knowledge about it. Consequently, other pertinent aspects and categories could be used instead, e.g. local source capabilities. In our approach this can be easily done by simply modifying the needs ontology.

Quality : this metadata category specifies the required quality level for local sources. The class Quality is defined as follows:

$$\begin{aligned}\mathsf{Quality} \equiv\ & \exists sourceQuality._Z \wedge \exists dataQuality._Z \wedge \\ & \exists dataFreshness._Z\end{aligned}$$

where *sourceQuality*, *dataQuality* and *dataFreshness* are integer attributes modeling some possible quality criteria.

Availability : it describes the required availability level for pertinent local sources. The class Availability is defined as follows:

$$\begin{aligned}\mathsf{AlwaysAvailable} &\sqsubseteq \mathsf{Availability} \\ \mathsf{TemporarilyAvailable} &\equiv \mathsf{Availability} \wedge \\ &\quad \exists begin\_d._Z \wedge \exists end\_d._Z \wedge \\ &\quad \exists begin\_h._Z \wedge \exists end\_h._Z\end{aligned}$$

A selected source may be always available or in a period only. In our example, the period is specified by four integer attributes. *begin_d* and *end_d* represent respectively the first day and the last day of the period. *begin_h* and *end_h* represent the initial and final hour of service in the day. This simple source availability model could be easily improved by specifying more complex class definitions and/or constraints.

Cost : this class is used to represent the cost of accessing a data source. It is defined as follows:

$$\begin{aligned}\mathsf{Cost} \equiv\ & \exists fixedCost._Z \wedge \exists variableCost._Z \wedge \\ & \exists connectionSpeed._Z\end{aligned}$$

where the three attributes represent respectively fixed access cost, variable access cost, e.g., euros per hour, and the connection speed for accessing the data source.

When a new needs expression Need_i is defined, a new class SourcePreference_i representing source preferences is specified. Here is an example:

$$
\begin{aligned}
\mathsf{SourcePreference\_i} \equiv\ & \exists \mathsf{hasQuality}.(\exists sourceQuality.\mathbf{min_3} \wedge \\
& \exists dataQuality.\mathbf{min_1} \wedge \\
& \exists dataFreshness.\mathbf{max_5}) \wedge \\
& \exists \mathsf{hasAvailability}.(\exists begin\_d.\mathbf{max_1} \wedge \\
& \exists end\_d.\mathbf{min_{31}} \wedge \\
& \exists begin\_h.\mathbf{max_0} \wedge \\
& \exists end\_h.\mathbf{min_{23}}) \wedge \\
& \exists \mathsf{hasCost}.(\exists fixedCost.\mathbf{equal_0} \wedge \\
& \exists variableCost.\mathbf{equal_0} \wedge \\
& \exists connectionSpeed.\mathbf{min_{10}})
\end{aligned}
$$

where the user requires the source quality to be at least 3, data quality to be at least 1 and data freshness no more than 5 days. Sources must be available 24 hours a day, during the month of January. They must be accessible for free and provide a connection speed which corresponds at least to 10Mbps.

4 Mediator configuration

Given a valid needs expression, represented within the needs ontology, a mediator can be configured accordingly. Therefore, a needs expression is first analyzed in order to identify pertinent metadata by exploiting reasoning tasks. Then the corresponding set of ontologies, i.e., mediator configuration, is generated and made available in a web repository for download. This allows a mediator to access the repository in order to download its configuration at runtime. Thanks to this approach, no mediator recompilation is needed, and the configuration can be updated at any time.

The remainder of this section focuses on the three main configuration steps which are: *architecture identification*, *mediators importation*, *candidate source selection*. For a lack of space, no details on the *mediator ontologies generation* are given.

4.1 Architecture identification

The first step of the mediator configuration consists of identifying the architecture of a mediation system. For doing so, firstly, the *integration degree* is analyzed. According to its value, two main mediation system categories may be identified:

- *integrated mediation systems*: the mediation ontology is built as a view over the domain ontology. The domain ontology acts as a shared domain vocabulary. Centralized knowledge-based mediation systems are included in this category.

– *non-integrated mediation systems*: are characterized by a mediation ontology consisting of a non-integrated view over a set of local sources ontologies. It gives access to underlying sources by applying their local vocabulary. Distributed knowledge-based mediation systems are in this category.

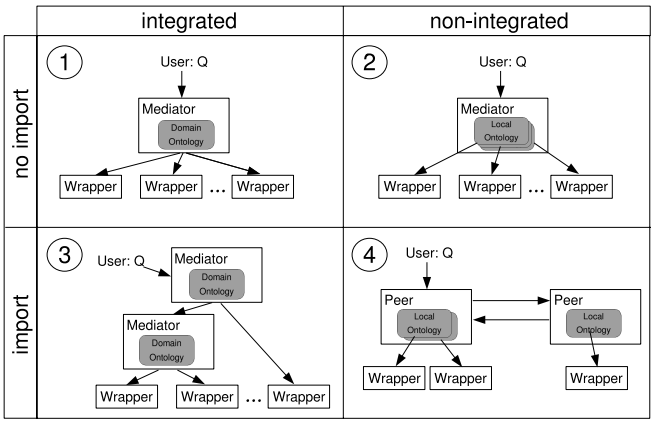

Fig. 5. Integration and import degrees.

Secondly, the *import degree* aspect is analyzed. According to its value, combined with the integration degree aspect, four main mediation systems architectures can be identified (cf. Figure 5). If mediators importation is not enabled (`ImportNone`), mediation systems are composed of only one mediator accessing underlying local sources. When an integrated mediation ontology is defined (cf. configuration `1` in the Figure), the resulting mediation system allows to emulate a centralized architecture.If the mediation ontology consists of a non-integrated view over local sources (cf. configuration `2`), the mediation system represents a kind of *multidatabase system* [13].

If mediators importation is enabled, the mediation system may be composed of more than one mediator. When a configured mediator provides an integrated mediation ontology (cf. configuration `3`), it may import both integrated and non-integrated mediators as resources. This allows to build more complex architecture such as hierachical mediation in [11]. However, when a non-integrated mediation ontology is adopted, imported mediators act as neighbour peers into a distributed mediation system (cf. configuration `4`). This allows to emulate existing knowledge-based distributed mediation systems.

Thirdly, after having identified all resources of the new configured mediator, the way the mediator accesses them is analyzed. Two possible methods may be applied to access imported resources [3]:

[3] Remember that resources are local sources and imported mediators.

– *rewriting* is needed when a mediation ontology `X` and a resource ontology `Y` (mediator or wrapper) adopt two different vocabularies. In this case a query expressed over the mediation ontology `X` (or a part of it) is rewritten according to the target vocabulary `Y` by exploiting *inter-ontology* semantic correspondences.
– *dispatching* is applied when a mediation ontology `X` and a resource ontology `Y` adopt the same vocabulary. In this case no rewriting is needed and a query (or a part of it) can be dispatched to the underlying resource for execution.

Independently of the mediation system architecture, a configured mediator is always characterized by:

– a mediation ontology (integrated or not), acting as a global schema, and
– a set of resources accessible by rewriting and/or dispatching.

Therefore, a mediator can easily be adapted to several mediation system architectures. Figure 6 gives some architecture examples (rewriting is represented by a continous arrow, while dispatching by a dashed one). For instance, in a centralized mediation system, a mediator `X` is configured so that all of its resources (i) correspond to wrapped sources and (ii) are accessed by rewriting. This is due to the fact that local sources are described by their own ontologies, independently of the mediator vocabulary. On the other hand, building a distributed mediation system means to cope with both mediators and wrappers as resources. The mediator (or peer) `X` accesses its underlying local sources by dispatching queries. This is due to the fact that each mediator in the distributed mediation system applies the local vocabulary to describe underlying data. Therefore, accessing other mediators requires a rewriting task through inter-ontology correspondences. More complex architecture can be configured, e.g. hierachical architecture allows to exploit previously defined mediators as resources by dispatching (partial) queries to them.

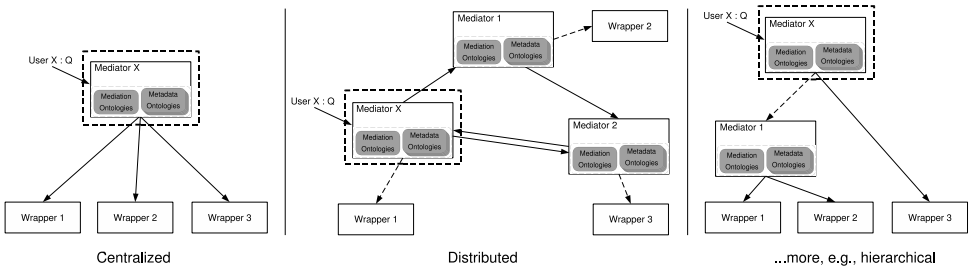

Fig. 6. Mediation system architectures.

4.2 Mediators importation

This task consists of identifying which running mediators can be imported as resources by a new configured mediator. This task is performed by exploiting

the concepts classification in the needs ontology. Let the need N_i be composed of n aspects, whose values are represented by classes $\{\mathsf{D}_{i1}, ..., \mathsf{D}_{in}\}$. Let a mediator M_i be configured according to the needs expression N_i. Given two needs N_1 and N_2, if:

- $\forall_{j=\{1..n\}}\ \mathsf{D}_{1j} \equiv \mathsf{D}_{2j}$, then it is deduced that $\mathsf{N}_1 \equiv \mathsf{N}_2$. This means that M_1 and M_2 have equivalent characteristics and consequently M_1 can be reused by M_2 as a resource and viceversa.
- $\forall_{j=\{1..n\}}\ \mathsf{D}_{1j} \sqsubseteq \mathsf{D}_{2j}$, then it is deduced that $\mathsf{N}_1 \sqsubseteq \mathsf{N}_2$. ADEMS interprets such a relation as a *needs containment*, which means that a mediator M_1 can be reused by a mediator M_2 as a pertinent resource.

In all other cases, N_1 and N_2 cannot be classified, and therefore, nothing can be stated.

Given a set of independently defined needs expressions, they are automatically classified by exploiting reasoning tasks. Let us suppose that two needs expression are defined as follows:

$$\begin{aligned}
&\mathsf{Need_1} \equiv \exists \mathit{hasConceptSet}.\mathsf{ConceptSet_1} \\
&\quad \sqcap \exists \mathit{hasArchitecture}.\mathsf{Architecture_1} \\
&\quad \sqcap \exists \mathit{hasSourcePreference}.\mathsf{SourcePref_1}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Need_2} \equiv \exists \mathit{hasConceptSet}.\mathsf{ConceptSet_2} \\
&\quad \sqcap \exists \mathit{hasArchitecture}.\mathsf{Architecture_2} \\
&\quad \sqcap \exists \mathit{hasSourcePreference}.\mathsf{SourcePref_2}
\end{aligned}$$

$$\mathsf{ConceptSet\_1} \equiv \mathsf{Person} \sqcup \mathsf{Car} \sqcup \mathsf{Job}$$

$$\mathsf{ConceptSet\_2} \equiv \mathsf{Person} \sqcup \mathsf{Car}$$

$$\begin{aligned}
&\mathsf{Architecture_1} \equiv \\
&\quad \exists \mathit{hasImportDegree}.(\mathsf{Jim} \sqcup \mathsf{Jack}) \\
&\quad \sqcap \exists \mathit{hasExportDegree}.\mathsf{Jim} \\
&\quad \sqcap \exists \mathit{hasMatDegree}.\mathsf{FullMaterialized} \\
&\quad \sqcap \exists \mathit{hasIntDegree}.\mathsf{Centralized}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Architecture_2} \equiv \\
&\quad \exists \mathit{hasImportDegree}.(\mathsf{Jim} \sqcup \mathsf{Jack}) \\
&\quad \sqcap \exists \mathit{hasExportDegree}.\mathsf{Jim} \\
&\quad \sqcap \exists \mathit{hasMatDegree}.\mathsf{FullMaterialized} \\
&\quad \sqcap \exists \mathit{hasIntDegree}.\mathsf{Centralized}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{SourcePref_1} \equiv \\
&\quad \exists \mathit{hasQuality}.(\exists \mathit{sourceQuality}.\mathbf{min_3} \\
&\qquad \sqcap \exists \mathit{dataQuality}.\mathbf{min_1} \\
&\qquad \sqcap \exists \mathit{dataFreshness}.\mathbf{max_5}) \\
&\quad \sqcap \exists \mathit{hasAvailability}.\mathbf{AlwaysAvailable} \\
&\quad \sqcap \exists \mathit{hasCost}.(\exists \mathit{fixedCost}.\mathbf{max_{10}} \\
&\qquad \sqcap \exists \mathit{variableCost}.\mathbf{equal_0} \\
&\qquad \sqcap \exists \mathit{connectionSpeed}.\mathbf{min_{10}})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{SourcePref_2} \equiv \\
&\quad \exists \mathit{hasQuality}.(\exists \mathit{sourceQuality}.\mathbf{min_3} \\
&\qquad \sqcap \exists \mathit{dataQuality}.\mathbf{min_3} \\
&\qquad \sqcap \exists \mathit{dataFreshness}.\mathbf{max_2}) \\
&\quad \sqcap \exists \mathit{hasAvailability}.\mathbf{AlwaysAvailable} \\
&\quad \sqcap \exists \mathit{hasCost}.(\exists \mathit{fixedCost}.\mathbf{equal_0} \\
&\qquad \sqcap \exists \mathit{variableCost}.\mathbf{equal_0} \\
&\qquad \sqcap \exists \mathit{connectionSpeed}.\mathbf{min_{10}})
\end{aligned}$$

Subsumption allows to deduce the following relations:

$$\mathsf{ConceptSet\_2} \sqsubseteq \mathsf{ConceptSet\_1} \tag{8}$$

$$\mathsf{Architecture\_2} \equiv \mathsf{Architecture\_1} \tag{9}$$

$$\mathsf{SourcePref\_2} \sqsubseteq \mathsf{SourcePref\_1} \tag{10}$$

The subsumption relation (8) is evident and does not require more explanation. Identical architectural choices (9) give as result an equivalence relation

between classes Architecture_2 and Architecture_1. This equivalence has no impact on the needs classification. In (10) SourcePref_2 is deduced to be subclass of SourcePref_1. This is due to the fact that all of its role restrictions are more restrictive or as much as the ones of SourcePref_2. For instance, the data freshness for Need_2, i.e., data must not be older than 2 days, is more restrictive than the Need_1 one, i.e. days must not be older than 5 years. Therefore the following subsumption relation is deduced:

$$\mathsf{Need\_2} \sqsubseteq \mathsf{Need\_1}$$

Now, let us suppose that architectural requirements for both needs were defined differently:

$$\begin{aligned}
&\mathsf{Architecture_1} \equiv \\
&\quad \exists \mathit{hasImportDegree}.(\mathsf{Jim}) \\
&\wedge \quad \exists \mathit{hasExportDegree}.\mathsf{Jim} \\
&\wedge \quad \exists \mathit{hasMatDegree}.\mathsf{FullMaterialized} \\
&\wedge \quad \exists \mathit{hasIntDegree}.\mathsf{Centralized}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Architecture_2} \equiv \\
&\quad \exists \mathit{hasImportDegree}.(\mathsf{Jim} \vee \mathsf{Jack}) \\
&\wedge \quad \exists \mathit{hasExportDegree}.\mathsf{Jim} \\
&\wedge \quad \exists \mathit{hasMatDegree}.\mathsf{FullMaterialized} \\
&\wedge \quad \exists \mathit{hasIntDegree}.\mathsf{Centralized}
\end{aligned}$$

Given the fact that the mediator M_1 can import resources from Jim, but not from Jack, the following subsumption relations are deduced:

$$\begin{aligned}
\mathsf{ConceptSet_2} &\sqsubseteq \mathsf{ConceptSet_1} \\
\mathsf{Architecture_2} &\sqsupseteq \mathsf{Architecture_1} \\
\mathsf{SourcePref_2} &\sqsubseteq \mathsf{SourcePref_1}
\end{aligned}$$

therefore, Need_2 and Need_1 cannot be classified and the mediator M_2 will not be deduced to be a potential resource for M_1.

A similar effect can be obtained if a disjointness relation between the two needs can be inferred for at least one aspect. Let us suppose that source preferences for both needs were defined as follows:

$$\begin{aligned}
&\mathsf{SourcePref_1} \equiv \\
&\quad \exists \mathit{hasQuality}.(\exists \mathit{sourceQuality}.\mathbf{min_3} \\
&\qquad \wedge \exists \mathit{dataQuality}.\mathbf{min_1} \\
&\qquad \wedge \exists \mathit{dataFreshness}.\mathbf{max_5}) \\
&\wedge \exists \mathit{hasAvailability}.\mathbf{AlwaysAvailable} \\
&\wedge \exists \mathit{hasCost}.(\exists \mathit{fixedCost}.\mathbf{equal_5} \\
&\qquad \wedge \exists \mathit{variableCost}.\mathbf{equal_0} \\
&\qquad \wedge \exists \mathit{connectionSpeed}.\mathbf{min_{10}})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{SourcePref_2} \equiv \\
&\quad \exists \mathit{hasQuality}.(\exists \mathit{sourceQuality}.\mathbf{min_3} \\
&\qquad \wedge \exists \mathit{dataQuality}.\mathbf{min_3} \\
&\qquad \wedge \exists \mathit{dataFreshness}.\mathbf{max_2}) \\
&\wedge \exists \mathit{hasAvailability}.\mathbf{AlwaysAvailable} \\
&\wedge \exists \mathit{hasCost}.(\exists \mathit{fixedCost}.\mathbf{equal_0} \\
&\qquad \wedge \exists \mathit{variableCost}.\mathbf{equal_0} \\
&\qquad \wedge \exists \mathit{connectionSpeed}.\mathbf{min_{10}})
\end{aligned}$$

Due to their incompatible values for the aspect *fixedcost*, SourcePref_2 and SourcePref_1 are deduced to be disjoint classes. Consequently Need_2 and Need_1 cannot be classified.

4.3 Candidate sources selection

A *candidate source* is a source that respects user-defined source preferences in the needs expression, i.e., availability, quality and cost requirements, but it does not necessarily fulfil the view definition, i.e. concept set. We refer as a *pertinent source*, a candidate source which also respects the user-defined view definition.

Metadata about sources is modeled within the *source metadata ontology*, whose structure is quite similar to the source preference category in the needs expression ontology. Let us imagine that the source metadata ontology contains the following two definitions:

$$\begin{aligned}
&\mathsf{Source_1} \equiv \\
&\quad \exists \mathit{hasQuality}.(\exists \mathit{sourceQuality}.\mathbf{equal_3} \\
&\qquad \wedge \exists \mathit{dataQuality}.\mathbf{equal_{10}} \\
&\qquad \wedge \exists \mathit{dataFreshness}.\mathbf{equal_1}) \\
&\wedge \exists \mathit{hasAvailability}.\mathbf{AlwaysAvailable} \\
&\wedge \exists \mathit{hasCost}.(\exists \mathit{fixedCost}.\mathbf{equal_5} \\
&\qquad \wedge \exists \mathit{variableCost}.\mathbf{equal_0} \\
&\qquad \wedge \exists \mathit{connectionSpeed}.\mathbf{equal_{15}}) \\
&\wedge \exists \mathit{hasKeyword}.\mathbf{Person} \\
&\wedge \exists \mathit{hasKeyword}.\mathbf{Car}
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Source_2} \equiv \\
&\quad \exists \mathit{hasQuality}.(\exists \mathit{sourceQuality}.\mathbf{equal_1} \\
&\qquad \wedge \exists \mathit{dataQuality}.\mathbf{equal_1} \\
&\qquad \wedge \exists \mathit{dataFreshness}.\mathbf{equal_5}) \\
&\wedge \exists \mathit{hasAvailability}.\mathbf{AlwaysAvailable} \\
&\wedge \exists \mathit{hasCost}.(\exists \mathit{fixedCost}.\mathbf{equal_0} \\
&\qquad \wedge \exists \mathit{variableCost}.\mathbf{equal_0} \\
&\qquad \wedge \exists \mathit{connectionSpeed}.\mathbf{equal_{10}}) \\
&\wedge \exists \mathit{hasKeyword}.\mathbf{Car} \\
&\wedge \exists \mathit{hasKeyword}.\mathbf{Motorbike}
\end{aligned}$$

The set of candidate sources, is retrieved by executing a reasoning-based query over the source metadata ontology. This query is built by using knowledge about user-defined source preferences in the needs expression.

In *integrated* mediation systems, candidate sources are firstly retrieved by comparing the source preferences with metadata contained within the source metadata ontology. This is done by defining a new class based on the user-defined source preferences. Here is an example:

$$\begin{aligned}
\mathsf{SourcePref_1} \equiv \exists \mathit{hasQuality}.(&\exists \mathit{sourceQuality}.min_3 \wedge \\
&\exists \mathit{dataQuality}.min_1 \wedge \\
&\exists \mathit{dataFreshness}.max_5) \wedge \\
\exists \mathit{hasAvailability}.&\mathsf{AlwaysAvailable} \wedge \\
\exists \mathit{hasCost}.(&\exists \mathit{fixedCost}.\mathbf{max_5} \wedge \\
&\exists \mathit{variableCost}.equal_0 \wedge \\
&\exists \mathit{connectionSpeed}.min_{10})
\end{aligned}$$

The candidate sources are retrieved by asking for synonyms and descendants of this class definition over the source metadata ontology. In our example, a query based on the class SourcePref_1, retrieves the source Source_1 only. Once candidate sources are discovered, pertinent ones are identified by exploiting the concept set information and semantic correspondences. This task is performed during the generation of the mediator configuration ontologies.

In *non-integrated* mediation systems, the pertinence of sources can be verified during the candidate sources retrieval. This is made possible by the presence of

source annotations as a set of *keywords*. Let us illustrate this aspect with an example:

$$\begin{aligned}\mathsf{SourcePref\_1} \equiv\ & \exists \mathit{hasQuality}.(\exists \mathit{sourceQuality}.\mathit{min}_3 \sqcap \\ & \quad \exists \mathit{dataQuality}.\mathit{min}_1 \sqcap \\ & \quad \exists \mathit{dataFreshness}.\mathit{max}_5) \sqcap \\ & \exists \mathit{hasAvailability}.\mathsf{AlwaysAvailable} \sqcap \\ & \exists \mathit{hasCost}.(\exists \mathit{fixedCost}.\mathbf{max_{15}} \sqcap \\ & \quad \exists \mathit{variableCost}.\mathit{equal}_0 \sqcap \\ & \quad \exists \mathit{connectionSpeed}.\mathit{min}_{10}) \\ & \exists \mathit{hasKeyword}.(\mathbf{Person} \sqcup \mathbf{House})\end{aligned}$$

In this example, source preferences establish that the fixed cost must not exceed 15. This would make both sources Source_1 and Source_2 two valid candidate sources. In order to identify pertinent sources, the concept set information is exploited to filter pertinent sources thanks to their keywords annotation. For example, let us suppose that the specified concept set in the needs expression corresponds to (Person $\sqcup$ House). The query SourcePref_1 is modified in order to take into account the concept set information as shown above. This allows to deduce that the only pertinent source is Source_1 (Person $\sqsubseteq$ Person $\sqcup$ House). Clearly this requires a source annotation effort in terms of the domain ontology vocabulary. Anyway, if the keywords information is not available for each source description, or deduced sources do not satisfy the user expectatives, a non-integrated mediation system requires the user to manually select pertinent sources among candidate ones.

5 Implementation and validation

We have implemented prototypes of the service and of the general purpose mediator. They have been implemented by using the Java platform and by exploiting the RACER [14] inference engine. RACER allows for an efficient $\mathcal{SHIQ}(\mathcal{D})$ ontologies management, and permits importing their description in several emerging standard languages such as OWL[4] and DAML+OIL[5]. RACER allows us to guarantee sound and complete reasoning tasks and provides highly optimized algorithms for terminology classification and reasoning on concrete datatypes [15].

Our approach has been validated in the context of computer assisted instruction (CAI) through the configuration of an integrated mediation system named SKIMA [16]. A graphical application, built on top of a configured mediator gives a transparent access to distributed material, e.g. documents. The domain ontology represents concepts and relations regarding the PI (programmed instruction) context, e.g. course, learner, section, document, exercise. Sources are modeled via

[4] W3C-OWL - http://www.w3.org/2004/OWL/
[5] DAML+OIL - http://www.daml.org/

a $\mathcal{SHIQ}(\mathcal{D})$ representation and integrate, via semantic correspondences, data about students and teachers, available courses with associated material and exercises. Students work through the programmed material by themselves at their own speed and after each step test their comprehension by answering questions.

We are currently conducting a validation experience in bioinformatics in the context of the Mediagrid project[17]. Mediagrid provides an infrastructure for giving a transparent access to biological sources. Using such an infrastructure, biologists can correlate expression levels of a gene and observe their evolution using data stored in a set of distributed data sources. Our goal is to exploit the ADEMS service in order to configure knowledge-based mediators being:

- able to exploit the current Mediagrid query evaluation capabilities, and
- adaptable to biologists needs in terms of view of interests over a biological domain ontology, architectural requirements and sources preferences.

6 Conclusions

This paper shows how to use ontologies for describing metadata and configuring mediators in an intelligent way. Given a user needs expression, ADEMS configures a mediator by generating a set of ontologies describing its necessary metadata. Reasoning tasks are also fully exploited for processing queries expressed in terms of the mediation ontology entities. Satisfiability and subsumption checking allow for query consistency verification and containment. Moreover, the knowlege-based query processing allows to enable approximative as well as partial query plans generation.

Being our approach strongly based on reasoning tasks, performances of the mediator configuration process are tightly related to the efficiency of the exploited reasoner. Currently, our approach allows fast metadata classification and mediator configuration, thanks to our metadata model and the use of RACER that provides efficient reasoning algorithms. More tests still have to be done in the presence of a very large amount of metadata. Recent research works, aiming to provide efficient reasoning algorithms on large ontologies, let us believe that future advances of inference engines will enable large-scale knowledge management. Future improvements in this research domain will validate our approach on huge number of data sources providing important volumes of distributed data.

Acknowledgements

We thank Eng. Hector Manuel Perez Urbina for his careful reading. Eng. Perez is the implementor of the SKIMA prototype, and he strongly participated in the implementation of the first ADEMS service prototype.

References

1. Wiederhold, G.: Mediators in the architecture of future information systems. Computer (1992) 25(3):38–49

2. Collet, C., Huhns, M., Shen, W.-M.: Resource integration using a large knowledge base in carnot. Computer (1991) 24(12):55–62
3. Arens, Y., Knoblock, C.-A., Shen, W.-M.: Query reformulation for dynamic information integration. Journal of Intelligent Information Systems - Special Issue on Intelligent Information Integration, (1996) 6(2/3): 99–130
4. Mena, E., Kashyap, V., Sheth, A.-P., Illarramendi, A.: OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. Conference on Cooperative Information Systems, (1996), 14–25
5. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Schema and data integration methodology for DWQ. Technical Report DWQ-UNIROMA-004, Dipartimento di Informatica Sistemistica, Universita‘ di Roma La Sapienza, (1998)
6. Goasdoué, F., Lattes, V., Rousset, M.-C.: The use of CARIN language and algorithms for information integration: The PICSEL system. International Journal of Cooperative Information Systems, (2000), 9(4):383 – 401
7. Baker, P.-G., Brass, A., Bechhofer, S., Goble, C., Paton, N., Stevens, R.: TAMBIS: Transparent access to multiple bioinformatics information sources. Int. Conf. on Intelligent Systems for Molecular Biology, (1998), 25–34
8. Peim, M., Franconi, E., Paton, N., Goble, C.: Query processing with description logic ontologies over object-wrapped databases. In SSDBM ’02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management, (2002), 27–36
9. Adjiman, P., Chatalic, P., Goasdoue, F., Rousset, M.-C., Simon, L.: Somewhere in the semantic web. Technical report, LRI, (2004)
10. Bruno, G., Vargas-Solar, G., Collet, C.: ADEMS, an adaptable and extensible mediation framework: application to biological sources. Electronic Journal e-Gnosis (2004)
11. Li, C., Yerneni, R., Vassalos, V., Garcia-Molina, H., Papakonstantinou, Y., Ullman, J., Valiveti, M.: Capability based mediation in TSIMMIS, (1998), 564–566
12. Horrocks, I., Sattler, U., Tobies, S.: Reasoning with individuals for the description logic shiq. In CADE-17: Proceedings of the 17th International Conference on Automated Deduction, (2000), 482–496
13. Litwin, W., Mark, L., Roussopoulos, N.: Interoperability of multiple autonomous databases, (1990), 22(3):267–293
14. Haarslev, V., Möller, R.: Practical reasoning in racer with a concrete domain for linear inequations. In Proceedings of the International Workshop on Description Logics, (2002), 91–98
15. Haarslev, V., Möller, R.: Description logic systems with concrete domains: Applications for the semantic web. In KRDB, (2003)
16. Perez-Urbina, H., Bruno, G., Vargas-Solar, G.: SKIMA: Semantic Knowledge and Information Management. In Proceedings of the Encuentro Internacional de Computacion, (2005)
17. Collet, C., et al.: Towards a mediation system framework for transparent access to largely distributed sources. In Proceedings of the International Conference on Semantics of a Networked World (sematics for Grid databases), (2004)