A multi-agent system for querying heterogeneous data sources with ontologies

Paolo Dongilli, Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris *

Faculty of Computer Science
Free University of Bozen/Bolzano
Piazza Domenicani 3, 39100 Bozen/Bolzano, Italy
lastname@inf.unibz.it

Abstract. In this paper we present the SEWASIE system, a multi-level agent-based architecture for querying heterogeneous data sources integrated by means of ontologies. Main features of this system are: two level data integration scheme, a query tool that supports the user in formulating a precise query, integrated tools for negotiation and information monitoring, and an agent infrastructure that provides a unifying framework for the architecture. In this work we focus on the querying process, from the user interfacer to the query answering mechanism. We show how the use of ontologies integrates the user interface with the underlying agent architecture.

1 Introduction

The SEWASIE (SEmantic Webs and AgentS in Integrated Economies) project (IST-2001-34825) is an European research projects that aims at designing and implementing an advanced search engine enabling intelligent access to heterogeneous data sources on the web, in a rich semantic (ontological) framework. Thus, it provides the basis for precise and effective access to information, and efficient web based communications. The goal of the architecture is to support a flexible set of actors enabling data providers, intermediaries, and data seekers to meet and exchange both information and meta-information, the ultimate goal being the ability to support large scale communities and economies. The project started on May 2002, and will take place for three years.SEWASIE is a collaboration between the Università degli Studi di Modena e Reggio Emilia, the Università degli Studi di Roma "La Sapienza", the Rheinisch Westfaelische Technische Hochschule Aachen, and the Free University of Bozen/Bolzano, with CNA SERVIZI Modena s.c.a.r.l, Thinking Networks AG, IBM Italia SPA, and Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung eingetragener Verein as industrial partners.

In this paper we present the design and development of the system architecture, describing the main components and their functionality. In particular, we concentrate on the process of query answering, starting from the query tool that assists users in building queries and moving through different types of agents that collaborate in collecting the information for the answer. Agent technology provides a unifying framework for all these modules. This is done in sections 2-4. In section 5 we introduce some characteristiscs of current system deployment, and in section 6 we compare SEWASIE with some related systems.

* This work has been partially supported by the EU projects Sewasie, KnowledgeWeb, and Interop.

2 SEWASIE Architecture

The SEWASIE system architecture [1] satisfying the aforementioned ideas and desiderata is shown in figure 1. *Brokering agents* (BAs) are the peers responsible for maintaining a view of the knowledge handled by the network. This view is maintained in *ontology mappings*, that are composed by the information on the specific content of the SEWASIE Information Nodes (SINodes) which are under the direct control of the BA, and also by the information on the content of other BAs. Thus, BAs must provides means to publish the locally held information within the network.

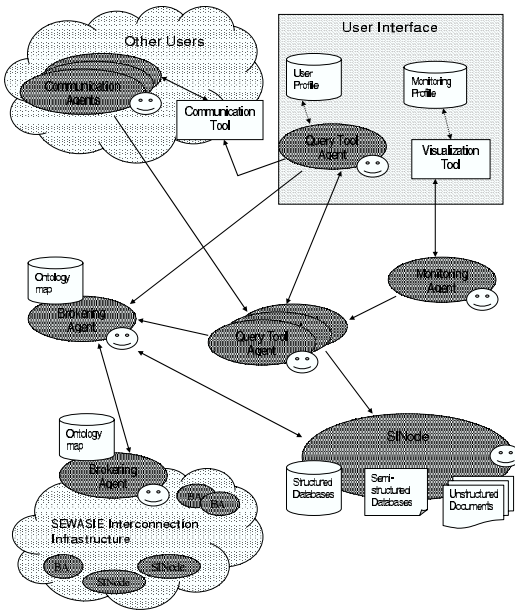

Fig. 1. The SEWASIE system architecture.

Query agents (QAs) are the carriers of the user query from the user interface to the SINodes, and have the task of solving a query by interacting with the BAs network. Once a BA is contacted, it informs the QA a) which SINodes under its control contain relevant information for the query, and b) which other BAs may be further contacted. Therefore, the QA translates the query according to the ontology mappings of the BA, and directly ask the SINodes for collection partial results. Also, it decides whether to continue the search with the other BAs. Once this process is finished, all partial results are integrated into a final answer for the user.

The life cycle of a QA is initiated by an invocation to its service for solving a query, and it is finalized when delivering the results. Therefore, a single QA is attached to only one query processing. In principle, it seems preferable that QAs reside in a single server node, sending remote messages to BA and SINodes on other server nodes, so mobility is not an issue for QA. Anyway, it might be possible for a query agent to decide that the SEWASIE Server node it is residing in is overloaded, and therefore prefers to move to other Server node in order to improve query answering performance

(load balance). These mobility could be supported and/or implemented by the underlying agent platform.

The SINodes are mediator-based systems[2], each including a global view of the overall information managed within. The managed information sources are heterogeneous collections of structured, semi-structured or unstructured data, e.g. relational databases, XML or HTML documents. SINodes are accessed by QAs in order to obtain data, and also by the managing BAs for building the ontology mappings. In order to create and maintain a global view of its information sources, SINodes require an ontology builder. This component performs in a semi-automatic way the enrichment process to create the SINode *ontology*. In turn, this SINode ontology is also integrated with other similar components into the BAs ontology mappings.

The user interface is a group of modules which work together to offer an integrated, easy to the user interaction with the SEWASIE system. It includes a query tool that guides the user in composing queries. In doing so, it requires the ontology of a *starting brokering agent* which helps in the interface presentation and behavior. Each instance of the query tool includes a Query Tool Agent (QTA) that is responsible to carry out communications with other agents in the SEWASIE system. In general, QTAs are needed to obtain the initial ontology from a BA, and also to create QAs that will solve the queries generated by the user.

Two extra elements of the user interface are the visualization tool and the communication tool. The visualization tool is responsible for monitoring information sources according to user interests which are defined in *monitoring profiles*. To perform this task, the visualization tool generates one *monitoring agent* (MA) for each topic of interest. Each MA contains a fixed internal ontology (so-called domain model) which is linked to higher level SEWASIE ontologies. Agents of this type regularly set up QAs to query the SEWASIE network, filter the results, and fill *monitoring repositories* with observed documents.

The communication tool supports negotiation between the user and other parties. Any query including contact information sets the context to launch the communication tool. This tool create several types of communication agents (CAs) that help in finding and contacting potential business partner, asking for initial offers, and ranking them. The human negotiator can then decide and choose the best offer to begin negotiating, with support from the communication tool.In order to fulfill their tasks, some of these agents periodically create query agents to search for information in the SEWASIE network.

We presented in this section the different types of agents that conform the SEWASIE architecture. In next section we will describe in more detail how they interact in order to solve a query.

3 Query answering process

In this section we provide a brief account of the query answering process. For a more detailed description the reader should refer to [3, 4]. Query management in the Sewasie framework involves different tasks corresponding to the two-level integration scheme. These can be summarized as follows:

global level Given a query expressed in terms of an ontology provided by a brokering agent,

1. Single out the SINodes managed by the BA that are relevant for computing the answer to the query, and reformulate in terms of queries to be posed to the SINodes.
2. Individuate alternative brokering agents that may have links to SINodes containing relevant information for computing the answer. The query is then reformulated in order to obtain single queries which should be forwarded to the appropriate BA.

3. Reconstruct the answer to on the basis of the answers to .

Note that the second step above is recursive, in the sense that answering a query posed to other BAs is done through the very same process we are describing. This means that the overall strategy for query management must deal with the problem of how to stop recursion.

local level Given a query posed in terms of the virtual global view associated to an SINode, retrieve the answer to the query. This task is carried out by the query manager of the SINnode of interest, and is characterized as follows. The first goal of this task is to derive a query plan that is able to correctly access the data sources under the control of that SINode. The second goal of this task is to execute the query, thus computing the corresponding answer.

Let first consider the case in which there is a single BA in the system (or no other BAs are relevant for a given query). In this scenario the key components involved in the query answering process are the BA Ontology (used to compose the query) and the mappings between this ontology and the Global Virtual Views exported by the known SINodes.

Query processing is based on two query reformulation steps. The first step reformulates the query in terms of the SINodes known by the Brokering Agent, and the second step reformulates each of the SINode query obtained in the first step in terms of the data sources known by the SINodes. The actual query processing phases are the following:

1. **Query expansion**: the query posed in terms of the brokering agent ontology is expanded to take into account the explicit and implicit constraints in the brokering agent ontology.
2. **BA Class materialization**: the atoms in the expanded query are materialized by taking into account the mapping from the classes in the global schemas of the SINodes to the classes of the Brokering agent ontology.
3. **Evaluation of the expanded query**: when all the BA ontology classes that are relevant for the query are materialized, the expanded query is submitted to the Query Engine to obtain the answer of the original query.

Contrary of the structure among BA and SINodes, there is no hierarchical relation among different BAs. In fact, every brokering agent acts (and cooperates) at the same level. To model the data integration problem underlying the interaction between brokering agents, one should come up with a formal framework which is of different nature with respect to the one adopted in the characterization of SINodes. Whereas in a SINode, data integration is based on the existence of the global virtual view, such a notion does not show up when trying to formalize the interconnection between brokering agents. The formal framework adopted within Sewasie follows the Peer to Peer (P2P) paradigm [5–8]. Roughly speaking, each brokering agent can be considered a peer, and the interconnetion between brokering agents can be seen as mappings between peers.

4 Query Tool

In this section we describe the underpinning technologies and techniques enabling the query user interface. We will start by describing our assumptions on the query language, followed by the query building process. The whole system is supported by formally defined reasoning services, described below in this section.

4.1 Conjunctive queries

Our aim is to be as less restrictive as possible on the requirements for the ontology language. In this way, the same technology can be adopted for different frameworks, while the user is never exposed to the complexity (and peculiarities) of a particular ontology language.

In the SEWASIE context, an ontology is composed by a *set of predicates* (unary, binary), together with a *set of constraints* restricting the set of valid interpretations (i.e. databases) for the predicates. The kind of constraints which can be expressed defines the expressiveness of the ontology language. Note that these assumptions are general enough to take account of widely used modelling formalisms, like UML for example.

The query tool is build around the concept of classes and their properties, so we consider conjunctive queries composed by unary (classes) and binary (attribute and associations) terms. Query expressions are compositional, and their logical structure is not flat but tree shaped; i.e. a node with an arbitrary number of branches connecting to other nodes. This structure corresponds to the natural linguistic concepts of noun phrases with one or more propositional phrases. The latter can contain nested noun phrases themselves.

A query is composed by a list of terms coming from the ontology (classes); e.g. "Supplier" and "Multinational". Branches are constituted by a property (attributes or associations) with its value restriction, which is a query expression itself; e.g. "selling on Italian market", where "selling on" is an association, and "Italian market" is an ontology term.

The body of a query can be considered as a graph in which variables (and constants) are nodes, and binary terms are edges. A query is connected (or acyclic) when for the corresponding graph the same property holds. Given the form of query expressions composed by the interface above introduced, we restrict ourselves to acyclic connect queries. This restriction is dictated by the requirement that non expert user must be comfortable with the language itself.[1] Note that the query language restrictions do not affect the ontology language, where the terms are defined by a different (in our case more expressive) language. The complexity of the ontology language is left completely hidden to the user, who doesn't need to know its details.

To transform any query expression in a conjunctive query we proceed in a recursive fashion starting from the top level, and transforming each branch. A new variable is associated to each node: the list of ontology terms corresponds to the list of unary terms. For each branch, it is then added the binary query term corresponding to the property, and its restriction is recursively expanded in the same way.

Let us consider for example the query "Supplier and Multinational corporation selling on Italian market located in Europe", with the meaning that the supplier is located in Europe. Firstly, a new variable (x_1) is associated to the top level "Supplier and Multinational corporation". Assuming that the top level variable is by default part of the distinguished variables, the conjunctive query becomes

$$\{x_1 \mid \mathrm{Suppl}(x_1), \mathrm{Mult\_corp}(x_1), \ldots\},$$

where the dots mean that there is still part of the query to be expanded. Then we consider the property "selling on", with its value restriction "Italian market": this introduces a new variable $x_{1,1}$. The second branch is expanded in the same way generating the conjunctive query

$$\{x_1 \quad | \quad \mathrm{Suppl}(x_1), \mathrm{Mult\_corp}(x_1), \mathrm{sell\_on}(x_1, x_{1,1}), \mathrm{It\_market}(x_{1,1}), \mathrm{loc\_in}(x_1, x_{1,2}), \mathrm{Eur}(x_{1,2})\}.$$

[1] Our technique can deal with disjunction of conjunctive queries, even with a limited form of negation applied to single terms. See [9, 10] for the technical details.

This transformation is bidirectional, so that a connected acyclic conjunctive query can be represented as a query expression by dropping the variable names. This means that, given the ontology terms, the interface can be used to generate any acyclic connected conjunctive query.

4.2 Query building

Initially the user is presented with a choice of different query scenarios which provide a meaningful starting point for the query construction. The interface guides the user in the construction of a query by means of a diagrammatic interface, which enables the generation of precise and unambiguous query expressions.

The focus paradigm is central to the interface user experience: manipulation of the query is always restricted to a well defined, and visually delimited, subpart of the whole query (the *focus*). The compositional nature of the query language induces a natural navigation mechanism for moving the focus across the query expression (nodes of the corresponding tree). A constant feedback of the focus is provided on the interface by means of the kind of operations which are allowed. The system suggests only the operations which are "compatible" with the current query expression; in the sense that do not cause the query to be unsatisfiable. This is verified against the formal model (the ontology) describing the data sources.

One of the main requirements for the interface is that it must be accessed by any HTML browser, even in presence of restrictive firewalls. This constraints its design, which overall appearance is shown in Figure 2. The interface is composed by three functional elements. The first one (top part) shows the tree structure of the query being composed, and the current focus. The second one is the query manipulation pane (bottom part) providing tools to specialise the query. Finally, a query result pane containing a table representing the result structure. The first two components are used to compose the query, while the third one is used to specify the data which should to be retrieved from the data sources. Because of lack of space, in this paper we concentrate on the query building part. Therefore we wont discuss the query result pane, which allows the user to define the columns of a table which is going to organise the data from the query result.

Since a query is a tree, the focus corresponds to a selected sub-tree. Each sub-tree univocally identifies a different variable corresponding to a node. Therefore, the focus is always on variables, and moving the focus corresponds to selecting a different variable. Modifying a query sub-part means operating on the corresponding sub-tree modifying the corresponding query tree. There are different possible operations, corresponding to elements of the query interface.

Substitution by navigation corresponds to substitute the whole sub-tree with the chosen ontology term. The result would be that the resulting sub-tree would be composed by a single node, without any branch, whose unary term is the given ontology term.

In the *refinement by compatible terms*, the selected terms are simply added to the root node as unary query terms. The system suggests terms from the ontology whose overlap with the focus can be non-empty (the "compatibility" requirement).. For example, "Student" would be among the compatible terms for the focus "Employee", but "Textile" would not.

The *property extension* enables the user to add attributes (e.g. "Employee whose name is") or associations (e.g. "Industry with sector"), and it corresponds to the creation of a new branch of the query tree. This operation introduces a new variable (i.e. node) with the corresponding restriction. When an attribute is selected, and a constant (or an expression) is specified, then this is added as restriction for the value of the variable.

Reasoning services w.r.t. the ontology are used by the system to drive the query interface. In particular, they are used to discover the terms and properties (with their restrictions) which are

Fig. 2. Query building interface.

proposed to the user to manipulate the query. In fact, the terms and the properties proposed by the system depend on the overall query expression, not only on the focus. This means that subparts of the query expression, taken in isolation, would generate different suggestions w.r.t. those in their actual context in the query.

We do not impose general restrictions on the expressiveness of the ontology language; however, we require the availability of two *decidable* reasoning services: satisfiability of a *conjunctive query*, and containment test of two conjunctive queries, both w.r.t. the constraints. Because of space restrictions, the details of reasonining services are not included, and more information can be found in [11].

If the query language includes the *empty* query (i.e. a query whose extension is always empty), then query containment is enough (a query is satisfiable iff it is not contained in the empty query). As above described, the query building interface represents the available operations on the query w.r.t. the current focus; i.e. the variable which is currently selected. Therefore, we need a way of describing a conjunctive query from the point of view of a single variable. The expression describing such a viewpoint is still a conjunctive query; which we call *focused*. This new query is equal to the original one, with the exception of the distinguished (i.e. free) variables: the only distinguished

variable of the focused query is the variable representing the focus. In the following we represent as q^x the query q focused on the variable x. For example, the query

$$q \equiv \{x_1, x_{1,2} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \\ \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\},$$

focused in the variable $x_{1,1}$ would simply be

$$q^{x_{1,1}} \equiv \{x_{1,1} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \\ \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}.$$

The operations described in this section require two different types of information: *hierarchical* (e.g. substitution by navigation), and on *compatibility* (e.g. refinement and new properties).

Let us consider the substitution by navigation with the more specific terms (the cases with more general and equivalent terms are analogous). Given the focused query q^x, we are interested to the unary atomic terms T s.t. the query $\{y \mid T(y)\}$ is contained in q^x and it is most general (i.e. there is no other query of that form contained in q^x, and containing $\{y \mid T(y)\}$).

Refinement by compatible terms and the addition of a new property to the query require the list of terms "compatible" with the given query. In terms of conjunctive queries, this corresponds to add a new term to the query. The term to be added should "join" with the query by means of the focused variable, and must be compatible in the sense that the resulting query should be satisfiable. This leads to the use of satisfiability reasoning service to check which predicates in the ontology are compatible with the current focus. With unary terms this check corresponds simply to the addition of the term $T(x)$ to the focused query q^x, and verify that the resulting query is satisfiable.

The addition of a property requires the discovery of both a binary term and its restriction: the terms to be added are of the form $\{R(x, y), T(y)\}$ if the focused variable is x. As for the refinement by compatible terms, the system should check all the different binary predicates from the ontology for their compatibility. This is practically performed by verifying the satisfiability of the query $q^x \bowtie \{R(x, y)\}$, for all atomic binary predicates R in the signature and where y is a variable not appearing in q.[2] Once a binary predicate R is found to be compatible with the focused query, the restriction is selected as the most general unary predicate T such that the query $q^x \bowtie \{R(x, y), T(y)\}$ is satisfiable.

4.3 Query verbalisation

The system always presents the user with a natural language transliteration of the conjunctive query. This is performed in an automatic way by using meta information associated with the ontology terms, both classes and properties. The verbalisation of the ontology terms must be provided in advance by the ontology engineers. For the verbalisation we use an approach similar to the one adopted by the Object Role Modelling framework (ORM, see [12, 13]).

Each class name in the ontology has associated a short noun phrase (usually one or two words), which represents the term in a natural language fashion. For example, to the class *PStudent* is associated "Postgraduate student" for English and "fortgeschrittener Kursteilnehmer" for German. The user will see only the associated sentence, while *PStudent* is just used in the internal ontology representation.

[2] Here $\bowtie$ represents a natural join.

For (binary) associations the ontology engineer has to provide two different verbalisations for the two directions. For example, let assume that the ontology states that the association *occ_room* links the two classes *PStudent* and *Room*. Then the engineer associates to the association the verbalisation "occupies" for the direction from *PStudent* to *Room*, and the verbalisation "is occupied by" for the other direction.

Attributes need one direction only, since they are never used from the point of view of the basic data type. In this case, the engineer is only required to provide the attribute verbalisation from the point of view of the class.

It is important to stress that, although natural language is used as feedback to represent the query, this is used in generation mode only. Since the user does not write queries directly, there is no need to parse any natural language sentence or to resolve linguistic ambiguities.

5 System deployment

So far we have presented the functional architecture of the system. We now want to shortly describe its deploying architecture. The SEWASIE system is intended to operate in networked environments where heterogeneity and distribution of information arise. Peers expose their ontologies on the network and software agents act as a glue among the different peers. Peers are recognised as being part of the SEWASIE system as long as they register their ontologies by a brokering agent. From a deployment view point, what is distributed is the multi-agent system. As the scope of the SEWASIE project is to focus on the application of software agents and not in providing a general toolkit for building multi-agent systems, the choice was to use existing tools and practices. The key features we were looking for were:

- a high-level language in order to focus on application programming;
- portability in order to allow for multiple platforms to become part of the SEWASIE system in a transparent way;
- FIPA compliant in order to be aligned with the current standards in the agent technology;
- support and maintenance in order to meet deployment needs.

Currently, the number of alternative agent toolkits is quite good [14–22]. Our choice felt on the Java Agent DEvelopment (JADE) developed by TILab [21]. JADE is currently one of the most evolving toolkits and is an open source projects where both professionals and researchers take part. JADE is written in Java and exploits Java RMI for managing software distribution in the environment.

A JADE multi-agent system (or *platform*) is a logical space that can be distributed over diverse physical hosts. Each host participating to the platform has its own Java Virtual Machine (JVM) running. Each JVM is an agent container, i.e. a runtime environment that allows agents to concurrently execute. In order to boot the platform, a main container has to be created. The main container hosts the services necessary to support agent life cycle, migration and communicaiton. Technically, the main container activates the RMI registry that JADE uses to allow containers and agents to reside on multiple hosts. Containers eventually residing on remote hosts can be added to the platform at runtime. No matter where containers are located, the agent platform is seen as a uniform logical space, where all containers can be reached simply knowing their name. Recently, JADE introduced the support of security for multi-agent systems. Security for agents is seen as an extension of the Java security model and in particular of the JAAS interface. Besides the JADE security extension we have exploited tuneling techniques in order to address security issues related to network configurations. This has been necessary to deploy the system in firewalled environments.

6 Related Work

Several agent-based information retrieval systems are known. In order to compare to similar systems, we now emphasize SEWASIE main characteristics:

- two-level data integration scheme: strongly tied local nodes are integrated into SINodes; BAs provide globally integrated ontologies by means of weaker mappings.
- query management: query building assisted by a query tool, query rewriting in the two levels of data integration following local ontologies using sound and complete algorithms.
- additional tools: negotiation and monitoring tools integrated in the same agent architecture.

Altogether these points make the SEWASIE system unique among the agent-based information retrieval systems.

Some systems are strong on data integration. CARROT II [23] is an agent-based architecture for distributed information retrieval and document collection management. It consists of an arbitrary number of agents providing search services over local document collections or information sources. They contain metadata describing their local document store which are sent to other agents that act as brokers. Like in SEWASIE, these metadata have an unstructured form, without a central control. But there are anyway several differences with the SEWASIE architecture. First, data integration is done in only one level. In this sense, CARROT II agents play the role of a brokering agent and an SINode at the same time. Second, there is no support for the user in creating the query. Metadata information is not reflected in the process of query building. Finally, the most important difference is that agents in this system only produce a routing of the query to relevant information sources, no query rewriting is done in this step. In SEWASIE the query is reformulated following brokering agent's ontology before asking SINodes, which contain the information sources. Several other information retrieval systems are known with routing agents, like HARVEST [24], CORI [25] and InfoSleuth [26].

Other systems, like TSIMMIS [27], include some rewriting rules against predefined query patterns. There are several steps of query processing also in the MISSION project [28]. In these cases, data integration technology is not present, or in TSIMMIS limited to automatic generation of wrappers [29] and mediators [30] from web pages. In SEWASIE, the data integration techniques [31] adopted by SINodes apply not only to unstructured, or semi-structured data sources, but also to relational databases.

7 Conclusions

This paper presented the work done, within the perspective of querying data sources, in the SEWASIE project. We showed how the agent-based architecture has been tied up with an ontology based approach to provide the users a transparent access to heterogeneous data sources.

The SEWASIE system provides an ease of access to the data without requiring an in depth knowledge of the sources. This is achieved by leveraging both the agent based collaboration between the different components of the system, and the heavy use of ontologies as a means of abstraction.

Focusing on the Query Tool, this paper has presented the first well-founded intelligent user interface for query formulation support in the context of ontology-based query processing. Our work has been done in a rigorous way both at the level of interface design and at the level of ontology-based support with latest generation logic-based ontology languages such as description

logics, DAML+OIL and OWL. However, there are open problems and refinements which have still to be considered in our future work.

Another important aspect to be worked out is the understanding of the effective methodologies for query formulation in the framework of this tool, a task that needs a strong cooperation of the users in its validation. This is going in parallel with the interface user evaluation.

The other crucial aspect is the efficiency and the scalability of the ontology reasoning for queries. We are currently experimenting the tool with various ontologies in order to identify possible bottlenecks.

The agent framework has a few details which are going to be finalised during the last year of the project. In particular, the use of multiple Brokering Agents, and the deployment of some of the minor components of the overall architecture.

References

1. Bergamaschi, S., Fillottrani, P., Gelati, G.: The sewasie multi-agent system. In: Proceedings of the 3rd. International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2004), 19/20 July, New York. (2004)
2. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer **25** (1992) 38–49
3. Lenzerini, M., Majkić, Z., Beneventano, D., Mandreoli, F.: Techniques for query reformulation, query merging, and information reconciliation part a. Technical report, SEWASIE consortium (2003)
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Query reformulation over ontology-based peers. In: Proc. of the 12th Italian Conf. on Database Systems (SEBD 2004). (2004)
5. Bernstein, P., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: A vision. Workshop on the Web and Databases, WebDB (2002)
6. Halevy, A., Ives, Z., Suciu, D., Tatarinov, I.: Schema mediation in peer data management systems. In: Proceedings of the 19th International Conference on Data Engineering (ICDE'03). (2003)
7. Franconi, E., Kuper, G., Lopatenko, A., Serafini, L.: A robust logical and computational characterisation of peer-to-peer database systems. In: International VLDB Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03). (2003)
8. Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Logical foundations of peer-to-peer data integration. In: Proc. of the 23rd ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-2004). (2004) To appear.
9. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98). (1998) 149–158
10. Horrocks, I., Sattler, U., Tessaris, S., Tobies, S.: How to decide query containment under constraints using a description logic. In: Logic for Programming and Automated Reasoning (LPAR 2000). Volume 1955 of Lecture Notes in Computer Science., Springer (2000) 326–343
11. Dongilli, P., Franconi, E., Tessaris, S.: Semantics driven support for query formulation. In: Proceedings of the 2004 International Workshop on Description Logics (DL-04). Volume 104 of CEUR Workshop Proceedings. (2004)
12. Halpin, T.A.: Augmenting UML with fact orientation. In: HICSS. (2001)
13. : http://www.orm.net (2003)
14. : Agent Development Kit. (`www.tryllian.com/technology/product1.html`)
15. : Aisland Project. (`aisland.jxta.org`)
16. : April Agent Platform. (`www.nar.fujitsulabs.com/aap`)
17. : Comtec Agent Platform. (`ias.comtec.co.jp/ap`)

18. : FIPA-OS. (fipa-os.sourceforge.net)
19. : Grasshopper. (www.grasshopper.de)
20. : JACK intelligent agents. (www.agent-software.com)
21. : JADE. (jade.cselt.it)
22. : JAS API. (www.java-agent.org)
23. Klusch, M., Ossowski, S., Shehory, O., eds.: Integrating Distributed Information Sources with CARROT II. In Klusch, M., Ossowski, S., Shehory, O., eds.: Cooperative Information Agents VI, 6th International Workshop, CIA 2002, Madrid, Spain, September 18-20, 2002, Proceedings. Volume 2446 of Lecture Notes in Computer Science., Springer (2002)
24. Bowman, C.M., Danzig, P., Hardy, D.R., Manber, U., Schwartz, M.F.: The Harvest information discovery and access system. Computer Networks and ISDN Systems **28** (1995) 119–125
25. Callan, J.P., Lu, Z., Croft, W.B.: Searching distributed collections with inference networks. In Fox, E.A., Ingwersen, P., Fidel, R., eds.: SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, Washington, USA, July 9-13, 1995 (Special Issue of the SIGIR Forum), ACM Press (1995) 21–28
26. Woelk, D., Tomlinson, C.: Infosleuth: Networked exploitation of information using semantic agents. In: COMPCON Conference. (1995)
27. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.D., Vassalos, V., Widom, J.: The TSIMMIS approach to mediation: Data models and languages. Journal of Intelligent Information Systems **8** (1997) 117–132
28. McClean, S.I., Karali, I., Scotney, B.W., Greer, K., Kapos, G.D., Hong, J., Bell, D.A., Hatzopoulos, M.: Agents for querying distributed statistical databases over the internet. International Journal on Artificial Intelligence Tools **11** (2002) 63–94
29. Hammer, J., McHugh, J., Garcia-Molina, H.: Semistructured data: The tsimmis experience. In: Proceedings of the First East-European Symposium on Advances in Databases and Information Systems (ADBIS'97), St.-Petersburg, September 2-5, 1997. Volume 1: Regular Papers, Nevsky Dialect (1997) 1–8
30. Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object exchange across heterogeneous information sources. In Yu, P.S., Chen, A.L.P., eds.: Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan, IEEE Computer Society (1995) 251–260
31. Bergamaschi, S., Castano, S., Beneventano, D., Vincini, M.: Retrieving and integrating data from multiple sources: the MOMIS approach. Data and Knowledge Engineering **36** (2001) 215–249