An ontology based visual tool for query formulation support

Tiziana Catarci[1], Tania Di Mascio[2], Enrico Franconi[3], Giuseppe Santucci[1], and Sergio Tessaris[3]

[1]University of Roma "La Sapienza", Italy
<*lastname*>@dis.uniroma1.it
[2]University of L'Aquila, Italy
tania@ing.univaq.it
[3]Free University of Bozen-Bolzano, Italy
<*lastname*>@inf.unibz.it

Abstract

The SEWASIE (SEmantic Webs and AgentS in Integrated Economies) project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The overall project development strictly follows a user-centred design methodology. Users have been involved from the very beginning and are constantly participating in the design and testing of the system, in order to obtain a first example of visual interface user-dedicated for an ontology-based search engine. In this paper we describe the end-user interface component for query composition. We will describe the user-centred design of the interface, the underlying logic based technologies, and the formal principles of support and interaction. A full worked out example is included in the paper.

1 Introduction

In this paper we describe the principles of the design and development of an intelligent query interface, done in the context of the SEWASIE (SEmantic Webs and AgentS in Integrated Economies) European IST project. The SEWASIE project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The query interface is meant to support a user in formulating a precise query – which best captures her/his

information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The final purpose of the tool is to generate a conjunctive query (or Select-Project-Join query) ready to be executed by some evaluation engine associated to the information system.

The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The ontology defines a vocabulary which is richer that the logical schema of the underlying data, and it is meant to be closer to the user's rich vocabulary. The user can use the ontology's vocabulary to formulate the query, and she/he is guided by such a richer vocabulary in order to understand how to express her/his information needs more precisely, given the knowledge of the system. This latter task – called *intensional navigation* – is the most innovative functional aspect of our proposal. Intensional navigation can help a less skilled user during the initial step of query formulation, thus solving the critical aspect that only very skilled users have available sufficient knowledge about the schema of the stored information and, therefore, are able to formulate significant queries.

Queries can be specified through an iterative refinement process supported by the ontology through intentional navigation. The user may specify her/his request using generic terms; after the query classification, which makes explicit the meaning with respect to the ontology and the specificity of the query itself and of the subterms composing the query, the user may refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the information system, but by giving an explicit meaning to a query and to its subparts through classification.

The overall project development strictly follows a user-centred design methodology. Users have been involved from the very beginning and are constantly participating in the design and testing of the system, in order to obtain a first example of visual interface user-dedicated for an ontology-based search engine. In particular, they are giving key hints to build the user-interface, which is a crucial component of the overall SEWASIE architecture. So, in this paper we will give particular emphasis to the steps in user-centred design we have followed in this project.

The paper is organised as follows. At the beginning, a thorough analysis of the user centred interface design issues is carried on. The core part of the paper describes the functionality and the interface of the query tool. Then, the underlying technologies are introduced, from the point of view of the query expressiveness, the ontology support, and the natural language

verbalisation issues. Finally, a fully worked out example and the related works are presented.

2 User centred interface design

The key role of the user in the development process of interactive systems has been widely recognized (the term user-centered has been coined to denote recent design modalities which are oriented in such a direction [Adler and Winograd, 1992]). Generally speaking, user-centered design is an approach to interactive system development that focuses specifically on making systems usable [ISO-13407, 1996]. Its major advantages can be obtained only letting the user constantly participating in the design process from its first inception onwards. Also, the complete benefits of user-centered design come from calculating the total life-cycle costs of the product including conception, development, implementation, support, use and maintenance.

In order to be user-centered, the development of any interactive system, has to be carried out by emphasizing the following three points:

- the identification of the users and their needs;
- the usage of this information to develop a system which, through a suitable interface, meets the users' needs;
- the usability evaluation and validation tests of the system.

According to the methodology principles [Catarci *et al.*, 2003], the SEWASIE users have been involved in the design and development process from the first phase, namely the specification of the context of use, through both direct interviews and a questionnaire.

2.1 The CNA context analysis

The CNA was founded in 1946. Nowadays it counts more than five hundred thousand participants. It works at different levels: European, National, Regional, Provincial, and Municipal. CNA is an association of enterprises whose ultimate goals can be summarized as follows: to foster the unity of the artisan sector, to obtain political independence, to make clear the role of the members within the association, and to develop the services offered to the companies. The Modena CNA works at Provincial level. It offers different services through its own employees and its associated specialised companies (among them SIAER, INTERPRETA, FIDIMPRESA, SMA, SIXTEMA).

Managers from the textile sector and the mechanical sector associated to CNA have originated the needs for a sophisticated system like the one proposed by the Sewasie project. The principal goal of these users is to create a virtual enterprise (i.e., a set of small and medium enterprises that together provide the services of a larger enterprise) able to deal with bigger projects. Building a communication network among participants belonging to specific sectors (i.e., textile and mechanical) requires to set up an ontology describing as much as possible such sectors. A particular role will be played by the INTERPRETA company, which has been established by CNA Services of Bologna, CNA Services of Modena, and SIAER. Its principal activity is to produce clear documentations for CNA participants that explain all new legal laws. These services require, daily, repeated searches; to improve them it is important to create a specific ontology that describes the universe of laws.

Starting from our interaction with end-users we distinguish four end-user classes: Employees of Provincial and Municipal offices, Textile Customers and Participants, Mechanical Customers and Participants, and Employees of INTEPRETA, according to a particular sector of the SEWASIE ontology (e.g. CNA textile customers use the textile domain).

Main requirements for the query-definition interface are:

- simplicity of use (restrict query language),
- visual approach,
- interactive visualisation of the ontology terms, in order to build specific queries by direct interaction with the graphical interface,
- queries should be generalizable and specializable,
- fewer answers (by means of a precise query language),
- result should be structured, according to the end-user demand.

2.2 The design solution

The design of the user interface has been directed by the necessity to fulfil the requirements emerged from the user analysis. This involved the layout and functionalities of the user interface, as well as the kind of queries the user is able to build (i.e. the query language).

The query language is one of the main issue, since it must be simple enough to be easily understood and managed by a non specialistic user

(simplicity of use). But at the same time it must be expressive enough to capture specific queries able to retrieve only the required elements. For this reason the query language adopted is a restriction of conjunctive queries (see [Chandra and Merlin, 1977; Catarci *et al.*, 1997]). The expressiveness required is then shifted behind the scene by an extensive use of the ontology designed by using more powerful languages. This is an acceptable compromise, since it is expected that the ontology is design by domain experts comfortable with the complexity of the ontology language.

Another difficulty for the end user consists in choosing an appropriate starting point for the query; i.e. an initial query expression which can be refined to suit her/his request. To this purpose we devised a scenario based mechanism, in which the user is proposed with different starting points (scenarios). Each scenario corresponds to a typical query the user wants to pose (e.g. finding a supplier for services or products). The complete set of scenarios will be finalized during the forthcoming evaluation of the prototype (see Section 5).

Other requirements mentioned in Section 2.1 are reflected in the structure and functionalities of the actual interface. These are described in detail in Section 3.

3 Query interface

In this section we present the functionality and layout of the query interface. A complete example of the interaction of the user with the interface is then presented in Section 6 to clarify its functionalities.

The user is guided in the construction of the query by a diagrammatic interface, which enables the generation of precise and unambiguous query expressions (see Section 4.1 for a precise definition of the query language). Moreover interface presentation and behaviour are entirely guided by the ontology. This is achieved by leveraging the correct and complete automatic reasoning on the ontology language (see Section 4).

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.

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. The node is composed by a list (conjunction) of

ontology terms; e.g. "Supplier and Multinational corporation", where "Supplier" and "Multinational corporation" are two distinct terms. Each branch is constituted by a property (attribute or relation) with its value restriction, which is a conjunctive query itself; e.g. "selling on Italian market", where "selling on" is a relation, and "Italian market" is an ontology term (a sub-query composed by a single term).

Roughly speaking, the query interface supports the user in the composition of these query trees. Each tree has a direct (and unique) translation into conjunctive queries, which can be sent to the query manager (see Section 4) to retrieve the actual result from the data sources.

Although the focus navigation is completely under the control of the user, the system facilitates the movements by selecting the appropriate subparts of the query. Moreover, a constant feedback of the focus is provided by means of a diagrammatic representation of its context. The interface enables the user to compose the query by appropriate selections on the diagrammatic representation of the focus and its context. Moreover, the only allowed operations are those that do not cause the query to be unsatisfiable (i.e. not returning any value).

The key points that characterise the query interface can be summarised in the following list:

1. the user chooses a scenario from a predefined set to initialise the query, this scenario provides both an initial focus and a result structure, which can be refined by means of the interface;

2. the user is guided in the process of building the query by a diagrammatic representation of the focus and its context;

3. generated queries are unambiguous, w.r.t. both the meaning of the terms in the query and their relationships;

4. the system provides a constant feedback of the query by means of a natural language representation of the query itself;

5. the user can add any query subpart to be included in the result.

3.1 Query building interface

The interface is composed by three functional elements (see Figure 1 and Figure 2):

1. a text box showing a natural language representation of the query being composed, and the currently selected query subpart (top part of Figure 1 and Figure 2);

2. a query manipulation pane containing a diagram representing the focus and its terminological context, together with tools to specialise the query (the pane shown in Figure 1);

3. a query result pane containing a table representing the result structure (the pane shown in Figure 2).

The first two components are used to compose the query, while the third one to specify the objects which should to be retrieved from the data sources.

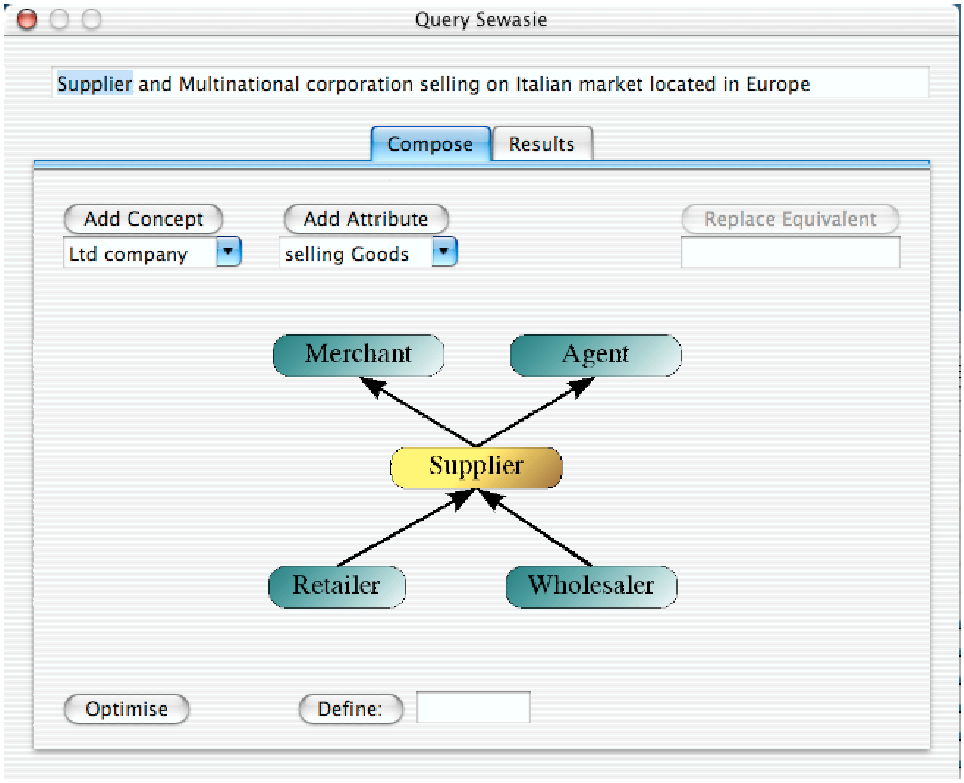

Figure 1: Query building interface.

3.1.1 Text box

The first component consists in a text box representing the query expression in a natural language fashion. The user selects subparts of the query (the *selection*) for further refinement. The selection defines the current focus, which will be represented in the diagrams described in the following sections. The selected subexpression can be modified (refined or extended) by means of the diagrams, and the modification is reflected on the query expression itself. Moreover, any operation described in the following sections operates only on the selection.

The text box component contains a linearized representation of the query tree being built, so its structure is hidden. Let us consider, for example, the query "Supplier and Multinational corporation selling on Italian market located in Europe". In this query we have three different nodes: "Supplier and Multinational corporation", "Italian market", and "Europe". Two different properties are "selling on", and "located in". It is clear that "selling on" is a property of the top node "Supplier and Multinational corporation"; but the property "located in", having "Europe" as value restriction, can be either a branch of "Supplier and Multinational corporation" or "Italian market".

Although the query expression does not provide accounts of its structure, the system is aware of the nesting (and so is the user). In fact, if the property in the example is attached to the top level, then it has been added while the focus was on "Supplier and Multinational corporation" rather than "Italian market". The system provides the feedback on the nesting by means of navigation in the query expression when the user is interested in selecting a subpart of the query. When a node is selected, then the system automatically selects the whole subtree rooted in the node selected by the user.

In the context of our example, let us assume that the user selects "Italian market" for further refinement. If the property "located in Europe" is attached to selected term (i.e. it is the market which is located in Europe), the property is automatically selected as well. If, on the other hand, the same property was attached to "Supplier", then the selection is not extended to the property.

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.

3.1.2 Query manipulation pane

The elements in the pane represent the current selection, and the operations allowed in its context. It is organised as a diagram showing the taxonomical context of the selection (the central part), and tools enabling the user to build the query expression.

The pane enables the user to refine the selection, by generalisation/specialisation or by specifying additional properties (attributes or relationships) and terms. Each change performed by means of selections in the diagrams is reflected on the textual representation of the query in the text box.

The central part of the interface is occupied by the diagram allowing what we call *substitution by navigation*; i.e. the possibility of substituting the selected portion of the query with a more specific or more general term.

The focus (the central part in the diagrams) shows the main term of the selection (e.g. "Supplier" or "Italian market" w.r.t. the example of Section 3.1.1). We call this term the *head* of the query, and it is automatically selected by the system according to the query itself and the current scenario.

Consider for example the query showed in Figure 1 with the selection on the first term ("Supplier") selected. The term itself is showed as the focus of the diagram. The terms "Merchant" and "Agent" are more general term in the ontology, while "Retailer" and "Distributor" are more specific.

By selecting one of these terms, the user substitutes the whole selection with the given term. The purpose of the substitution group is twofold: it enables the replacement of the focus and it shows the position of the selection w.r.t. the terms in the ontology.

It can be the case that in the ontology there are terms which are equivalent to the selected part. In this case the user is offered to replace the selection with the equivalent term by the activation of the `Replace Equivalent` button.

Note that the shown ontology terms depend on the overall query expression, not only on the selection. In fact, the main purpose of the diagram is presenting meaningful possible changes applicable to the selection. In any case these changes do not make the overall query unsatisfiable (i.e. empty).

Sub-queries can be associated to new names, and this process corresponds to the definition of new named views. These newly introduced names can be used to shorten the query expression, or as a simple mechanism to extend the user ontology. Once a sub-query has been selected (highlighted) a specific button (`Define`) in the interface enables the user to name it. Once the portion of the query has been renamed, the name appears in the query expression instead of the complex renamed part.

The `Add Concept` pop-up menu allows the user to refine the query by *compatible terms*. These are terms in the ontology whose overlap with the focus can be non-empty. These ontology terms can be added to the head of the selection to enable the user a more precise refinement of the result. For example, "Student" is among the compatible terms for the focus "Employee", but "Textile" is not. Note that the appropriate use of the disjointness constraints in the ontology is crucial to avoid the overload of the compatible terms (see Section 4.3). In fact "Textile" and "Student" can overlap unless their disjointness is stated in (or implied by) the ontology.

By means of the `Add Attribute` pop-up menu the user can add properties to the selection in order to restrict the scope of the focus of the query; i.e. adding *relationships* (e.g. "Industry with sector"), and/or *attributes* (e.g. "Employee whose name is"). The difference between the two kind of property lies on the class representing the range of the property. In the attribute case is a basic data type like "String" or "Integer", while for relationship it is a generic class.

Relationships The menu contains the name of the relationships which can be added to the current selection, together with the applicable value restriction. For example, if the selection is "Textile Industry" then the diagram includes the relationship "with sector" with the restriction "Textile". While for the selection "Industry", for the same relationship, the system would show the restriction "Sector" instead of "Textile".

Name and value restrictions for each relationship are verbalised using meta information associated to the terms in the ontology. For example, the relationship "with sector" with the restriction "Textile" is shown as "belonging to the textile sector".

Attributes Attributes are similar to relationships, but their restrictions are basic data types (e.g. strings or integers). When an attribute is selected, the user is requested to optionally enter an expression compatible with the data type to restrict the query. This expression can be a single value (e.g. the string "Modena"), or more complex expressions (e.g. a numerical range).

Specifying an expression for an attribute will restrict the result to objects having values for the attribute satisfying the expression.

The button `Optimise` is used to request the system to rewrite the query using a more concise representation. The kind of modifications include the

substitution of part of the query with equivalent terms, and lexical rewriting of the natural language representation of the query expression.

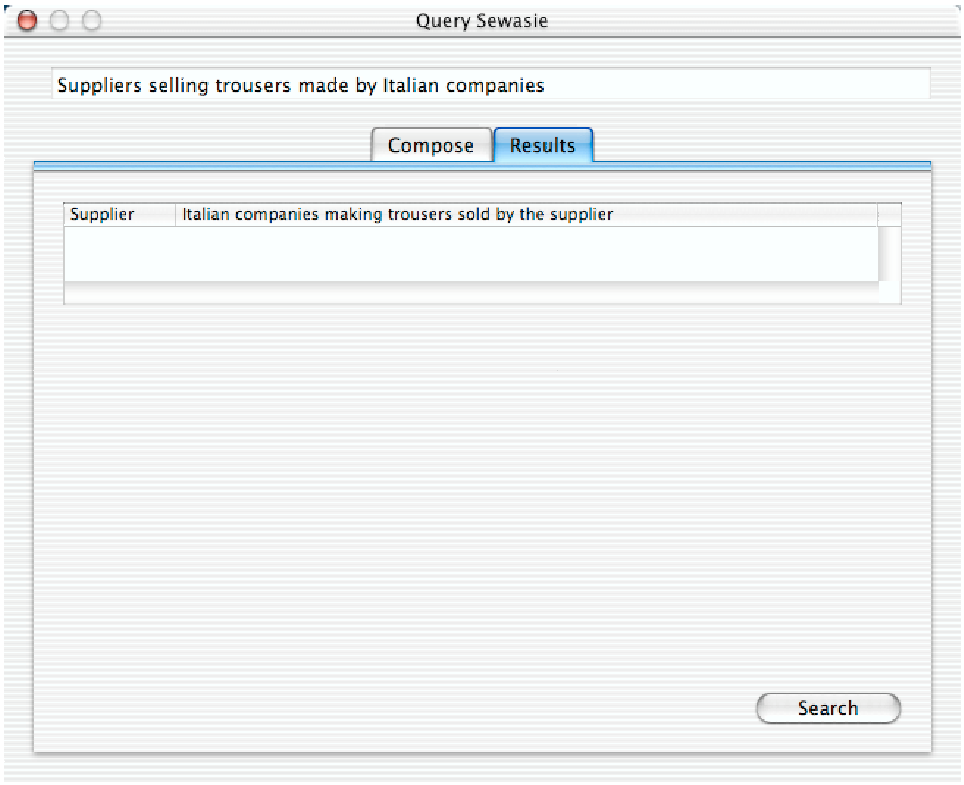

Figure 2: Query result pane.

3.1.3 Query Result pane

The third part of the interface contains the schema for the query result (see Figure 2). By means of this component the user can specify which elements of the query expression must be included in the result, as well as the structure of the table containing the result.

Initially, the table structure is specified by the chosen scenario, but the user can arbitrarily modify it by adding or removing columns. Any selected property (attribute/relationship) or sub-query in the text box can be included in the result by an opportune selection operation. This operation adds a new column to the table, and this is suitably labelled in order to univocally identify the query part it corresponds to.

As mentioned above, queries are tree structured, so each column is uniquely identified by the path from the top level. Note that the query can be arbitrarily nested, so paths can be composed by more than one property. For example, let us consider the query "Suppliers selling trousers made by Italian companies". The query has three levels: the top level is the company supplying trousers, which are the second level connected by the relation "which sell". The third level are the "Italian companies", which are connected to "trousers" by the relation "made by". If the user requires the companies as well as the suppliers in the result, the result table will contain two columns. The first is going to be labelled by "Supplier", while the second by the path "Italian companies making trousers sold by the supplier".

Note that a substitution operation performed in the query manipulation pane may affect part of the columns in the result structure. This can happen when, for instance, the whole selection is substituted by a single term. Therefore, when parts of the selection are selected as output result the system must decide whether they should be kept in the result structure or removed as well.

4 Underlying technologies

4.1 Conjunctive queries

The system enables the user to compose connected acyclic conjunctive queries, using unary (classes) and binary (property) terms. As already mentioned, this restriction is dictated by the requirement that the generic user must be comfortable with the language itself. Therefore the query manager is capable of answering to positive queries containing terms from the ontology.[1]

Note that the query language restrictions do not affect the ontology language, where the terms are defined by different (in our case more expressive) language. The ontology language contains constraints as disjuntiveness and covering. This gives the user the power of these construct, while the complexity of their understanding is left to the ontology engineers.

4.1.1 Conjunctive queries

In our framework a conjunctive query is an expression

$$\{x_1, \ldots, x_k \mid T_1(y_1), \ldots, T_n(y_n), R_1(z_1, w_1), \ldots, R_\ell(z_\ell, w_\ell)\}$$

[1]Note that in general query terms in the ontology not necessarily correspond directly to terms in the actual schema of the data sources; therefore, the query manager is responsible for the rewriting of the queries.

where the letters x, y, z, w denote variables or basic data constants (numbers, strings, etc.), T and R unary and binary terms respectively. Variables $x_1, \ldots, x_k$ are called *distinguished*, and represent the values which are going to be returned by the query. Remaining variables connect (join) the terms in the body of the query (i.e. $T_1(y_1), \ldots, T_n(y_n), R_1(z_1, w_1), \ldots, R_\ell(z_\ell, w_\ell)$), and are considered as existential quantified. Terms are conjoined (i.e. they all have to be satisfied), and the order in which they appear does not matter.

The body of the 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.

The result of a query is the set of k-tuples whose values, substituted to the distinguished variables, make the body of the query satisfied in the integrated view of the data sources. For example, a query to retrieve the suppliers selling on the Italian market would be

$$\{x \mid \text{Supplier}(x), \text{selling\_on}(x, y), \text{Italian\_market}(y)\}.$$

4.1.2 Query expressions and conjunctive queries

As introduced in Section 3.1, query expressions are tree structured. Under this assumption, we do not need to explicitly use variable names since the paths from the root univocally individuate each variable.

To each node in the query expression is associated a list of ontology terms (i.e. unary query terms); in addition, each departing branch is labeled by a property (i.e. binary query terms) connecting other nodes. 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". Note that the top level variable is by its nature part of the distinguished variables. Then, the conjunctive query becomes

$$\{x_1 \mid \text{Supplier}(x_1), \text{Multinational\_corporation}(x_1), \ldots\},$$

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 ($x_{1,1}$) and the query is expanded as

$$\{x_1 \mid \text{Supplier}(x), \text{Multinational\_corporation}(x_1), \\ \text{selling\_on}(x_1, x_{1,1}), \text{Italian\_market}(x_{1,1}), \ldots\}.$$

Finally, the other branch is expanded in the same way and the corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x), \text{Multinational\_corporation}(x_1), \\ \text{selling\_on}(x_1, x_{1,1}), \text{Italian\_market}(x_{1,1}), \\ \text{located\_in}(x_1, x_{1,2}), \text{Europe}(x_{1,2})\}.$$

If the user selected other properties to be included in the output (e.g. the "located in"), then the corresponding variable would be distinguished as well:

$$\{x_1, x_{1,2} \mid \text{Supplier}(x), \text{Multinational\_corporation}(x_{1,1}), \\ \text{selling\_on}(x_1, x_{1,1}), \text{Italian\_market}(x_{1,1}), \\ \text{located\_in}(x_1, x_{1,2}), \text{Europe}(x_{1,2})\}.$$

This transformation can be easily reversed, so a tree shaped conjunctive query can be represented as a query expression (in the sense of Section 3) by dropping the variable names. In fact, the system is using this inverse transformation since the internal representation of queries is conjunctive queries. Here we present the transformation in this way because it makes the syntax restriction on conjunctive queries easier to understand. The next section shows the system perspective over the described query building process. The verbalisation mechanism which enables the system to show the queries in a natural language fashion is then introduced in Section 4.2.

4.1.3 Query building

As described in Section 3, query building is performed by means of the query manipulation diagram which enables the user to modify the selected part of the query. With the formalisms introduced in the previous sections we are now able to give a precise meaning to the selection of sub-parts of the query on the text box.

Since a query is a tree, only sub-trees can be selected. Moreover, the system helps the user in this selection process by moving across the levels

of the tree. The movement is rendered to the user by an appropriate highlighting of query sub-parts in the text box. It is easy to realise that each sub-tree is univocally identified by a node (i.e. the sub-tree rooted in the node itself). Therefore, the focus is always on a node (i.e. a variable) and moving the focus corresponds to selecting a different variable.

The operations introduced in Section 3.1.2 can be described by means of modifications of the query tree as following:

Substitution by navigation corresponds to substitute the whole sub-tree with the chosen ontology term. The result would be a tree composed by a single node, without any branch, whose unary term is the given ontology term. This means that all the variables in its branches are deleted from the query, and this can be a problem if they where marked as distinguished. In this case, according to the type of substitution (super, sub, or equivalent) the sub-tree leading to the corresponding distinguished variable is not deleted.

Refinement by compatible terms: the selected terms are simply added to the root node as unary query terms.

Property extension: adding an attribute or relationship correspond to create a new branch. 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.

Summarising:

- the focus is always on a node of the query tree;
- modifying a query sub-part means operating on the corresponding sub-tree (adding or deleting branches, adding ontology terms to its root, substituting the whole tree with a single ontology term, etc.);
- the operation of dragging a property to the result structure corresponds to add the variable introduced by the property to the list of the distinguished ones.

4.2 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 Modeling (ORM, see [Halpin, 2001; ORM, 2003]).

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

For (binary) relations the ontology engineer has to provide two different verbalisation for the two directions. For example, let assume that the ontology states that the relation *occ_room* links the two classes *PStudent* and *Room*. Then the engineer associates to the relation 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. Therefore, the engineer is only required to provide the attribute verbalisation from the point of view of the class.

4.3 Ontology requirements for the interface

The whole interface is driven by algorithms which make an extensive use of logical reasoning services on top of the ontology. None of the diagrams described in Section 3, and shown in the example of Section 6, are precanned but they are built on the fly by means of appropriate reasoning services.

This means that the effectiveness of the interface depends almost entirely on the quality of the ontology (see [Franconi *et al.*, 2003]). In particular, a poor ontology will result in the interface being cluttered by irrelevant terms and properties. This will confuse the user, which would not receive a good support from the system in the refinement of the query.

The appropriate use of disjointness and covering constraints in the ontology is crucial for the system being able to prune insignificant terms from the query manipulation diagram. Lacks of relevant disjointness constraints (w.r.t. the modelled domain) affects the detection of relevant properties, and compatible terms of the selected sub-query. This leaves the user in the position of relying only on the taxonomical structure of the ontology terms.

To understand the reasons behind the importance of disjointness constraints, the reader should consider that in principle any terms and property is compatible with any other term or property unless otherwise stated

in the ontology. For example, the user would find terms like "Multinational corporation" among the terms compatible with "Student", or attributes like "city" applicable to "Trousers".

Considering the example presented in Section 6, an ontology without disjointness constraint would cause the interface to propose any attribute in the ontology as an applicable property to any term. This is obviously meaningless to the user, which find herself/himself to be forced to guess the right properties without any support.

Although not as crucial as disjointness, covering constraints provide an extremely powerful tool for discovering equivalences and properties using reasoning by cases.[2] Therefore, it enhances the user experience with the interface, as the system is able to pinpoint precise possible refinements for the sub-query.

5 The Query Interface evaluation

In order to evaluate the Sewasie query interface, we will propose to end-users (CNA and Interpreta employees) an usability-test, including the interaction with a simplified version of the query interface prototype.

The Usability-test has been designed following the method proposed in [Dix *et al.*, 2001]. Key-aspects considered in this test are:

- the easiness(intuitiveness) of:
  - to specify a set of task relevant data,
  - to construct the query,
  - to specify the query,
  - to generalize the query,
  - to rename the query,
  - to specify the result structure,
  - to connect to other Sewasie tools;
- the personal agreement of the following aspects:
  - the language/terms used on the screens is/are clear,
  - the interface design elements are well organized on the screen,

[2] When used in conjunction with disjointness.

- there is consistency in the way design elements are used and placed on the various interface screens,
- the system always valuable responds to user operations,
- the system always informs the user about its progress,
- the interface provides adequate shortcuts for common operations,
- the interaction mechanism that are offered by the interface substantially shield one from making errors,
- in the error situation, the interface helps one to recognize the specific error and recover from the error,
- the system has sufficient and useful help and documentation,
- it is easy to exit from a task situation at any level operation at any time,
- it is easy to perceive or remember the role of the interface elements,
- while performing a task, it is easy to remember (or acquire) the aspects that are relevant to the task.

6 Full query answering example

As working example we assume that the user is looking for suppliers of Levis trousers, together with the city in which they are located. Among the different scenarios the system proposes the *suppliers search*, which provides the right starting point to the user for further specialisation.

1. The text box starts with the expression "Supplier". Since the user is supposed to refine the query, this term is selected and shown on the focus in the diagram.

   The corresponding conjunctive query is

   $$\{x_1 \mid \text{Supplier}(x_1)\}.$$

2. The ontology states that "Supplier" participates in the relation "selling", but since the kind of supplier is not specified the only restriction the system can suggests is "Goods". Therefore, among the relationships there is "selling Goods". Selecting this relation results in the text box containing the expression "Supplier selling Goods".

   The corresponding conjunctive query is

   $$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{Goods}(x_{1,1})\}.$$

3. Since the user is interested in a particular kind of goods (trousers), then she/he highlights "Goods" in the text box. The diagram is now showing "Goods" instead of "Suppliers" as focus (Figure 3).

   The conjunctive query does not change since the focus has been only moved to a different sub-query.

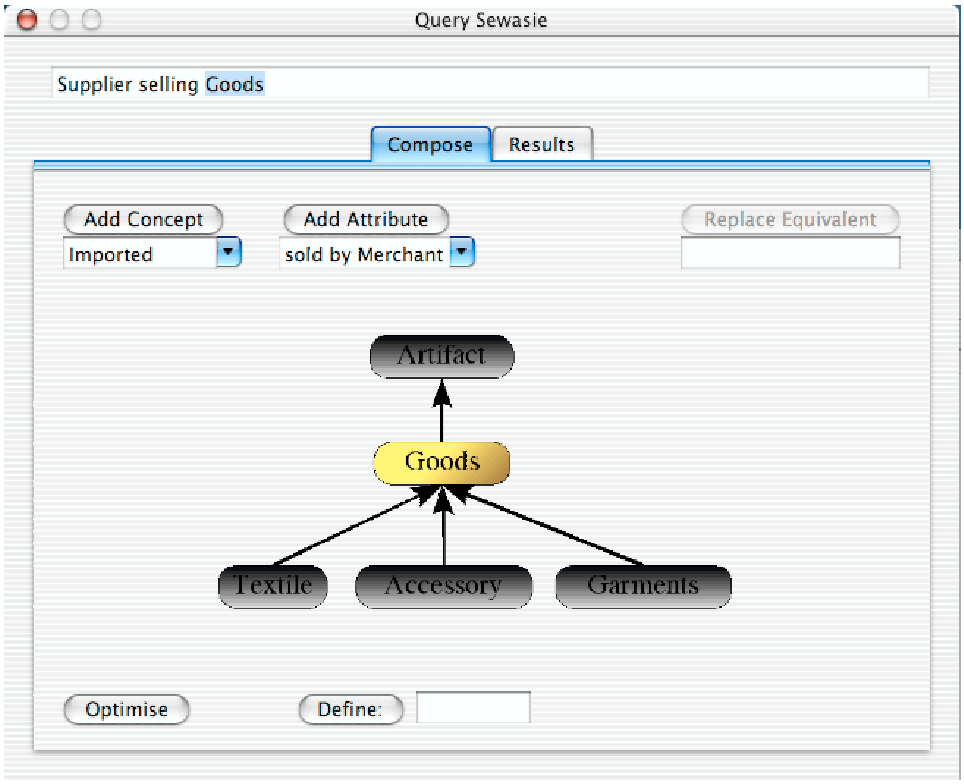

Figure 3: Selection on "Goods".

4. Among the more specific terms, the user finds "Garment" and selects it. Once selected the term "Goods" is substituted by "Garment" on the text box, and the diagram focus moves to "Garment".

   The text box contains "Supplier selling Garment", and the corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{Garment}(x_{1,1})\}.$$

5. The term "Trousers" is among the more specific terms of "Garment", so it is selected. Now the focus is on "Trousers", and the text box contains "Supplier selling Trousers".

   The corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{Trouser}(x_{1,1})\}.$$

6. The relation "of brand Clothing company" becomes now available for selection with the focus on "Trousers". Note that the system can narrow down the value restriction of the relation "brand", since the ontology states that a company that manufactures garments is a "Clothing company".

   The user selects the relationships, and the query in the text box changes in "Supplier selling Trousers of brand Clothing company". The corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{Trouser}(x_{1,1}), \\ \text{brand}(x_{1,1}, x_{1,1,1}), \text{Clothing\_comp}(x_{1,1,1})\}.$$

7. Since the user is interested in a particular brand name, he/she highlights the term "Clothing company" to add an attribute specification. Again the focus on the diagram changes and it is moved on "Clothing company". The diagram is now showing the attributes of "Clothing company", and among these there is "name".

   When the user selects this attribute, the system asks for string expression to be specified. The user types the string "Levis", and the text box shows "Supplier selling Trousers of brand Clothing company which name is Levis".

   The corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{Trouser}(x_{1,1}), \\ \text{brand}(x_{1,1}, x_{1,1,1}), \text{Clothing\_comp}(x_{1,1,1}), \text{name}(x_{1,1,1}, ''Levis'')\}.$$

8. The user wants to shorten the query by introducing a new name "JLevis" defined as "Trousers of brand Clothing company which name is Levis". To this purpose she/he selects this query sub expression, and selects the named view definition function.

The text box now contains "Suppliers selling JLevis", and the focus is on "JLevis" (Figure 4).

The corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{JLevis}(x_{1,1})\},$$

since "JLevis" hides its definition.

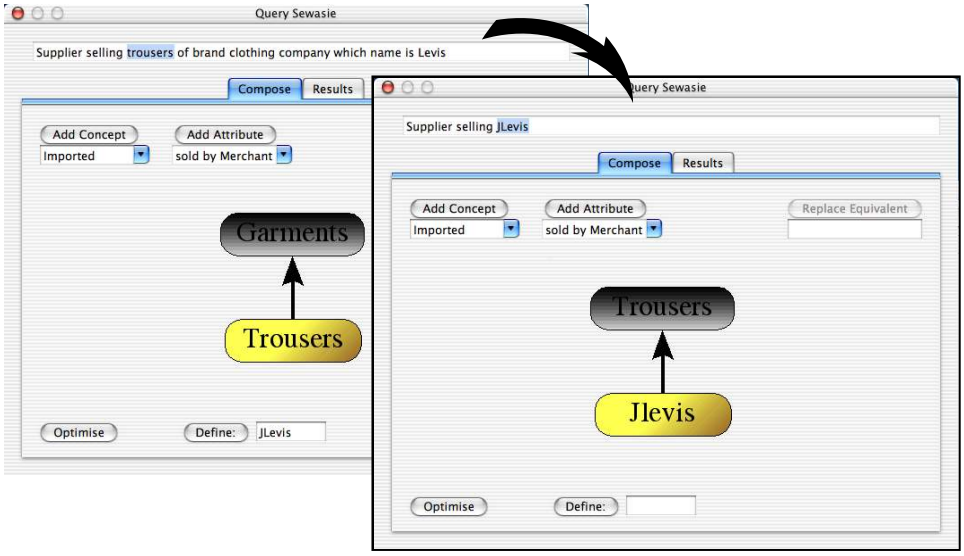

Figure 4: Renaming of sub expressions.

9. The focus term in the diagram changes from "Trousers" to "JLevis", but the rest of the diagram does not change since "JLevis" by definition is semantically equivalent to "Trousers of brand Clothing company which name is Levis". In particular, being "Trousers", "JLevis" has the attribute "waist size". The user selects it and, prompted by the system, adds the integer range [30,35]. The query now becomes "Suppliers selling JLevis with waist size between 30 and 35".

   The corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{JLevis}(x_{1,1}), \\ \text{waist\_size}(x_{1,1}, [30, 35])\}.$$

10. Together with the suppliers the user is interested in the city where each supplier is located. Since this is a property of "Suppliers", the user

highlights it to move the focus. The diagram now shows "Suppliers" as its focus; note that the diagram areas now contain different terms w.r.t. the starting point. This is because the query has been refined; for example, now the system knows that the supplier is a "Clothing company".

Among the relations of "Suppliers" there is "located in Address", which the user selects.

The corresponding conjunctive query is

$$\{x_1 \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{JLevis}(x_{1,1}), \\ \text{waist\_size}(x_{1,1}, [30, 35]), \text{location}(x_1, x_{1,2}), \text{Address}(x_{1,2})\}.$$

11. Since the user wants the city, the she/he highlights "Address" and selects its attribute "city" as part of the result structure.

    The query now is "Suppliers selling JLevis with waist size between 30 and 35 located in Address which city is any". Since the user did not specify the string value for "city", then the system uses the placeholder "any" which means any value. A new column is added to the result structure, and its label is "The city of the address of the location of the supplier".

    The corresponding conjunctive query is

$$\{x_1, x_{1,2,1} \mid \text{Supplier}(x_1), \text{sells}(x_1, x_{1,1}), \text{JLevis}(x_{1,1}), \\ \text{waist\_size}(x_{1,1}, [30, 35]), \text{location}(x_1, x_{1,2}), \\ \text{Address}(x_{1,2}), \text{city}(x_{1,2}, x_{1,2,1})\},$$

    note that now $x_{1,2,1}$ is a distinguished variable.

)

7 Related work

The work proposed in this paper deals with a relatively new problem, namely providing the user with a visual interface to query heterogeneous data sources through an integrated ontology, and a specific literature does not exist yet. Some preliminary work was done by one of the authors and his group [Bresciani and Franconi, 1996; Franconi, 2000; Bresciani *et al.*, 2000; Bresciani and Fontana, 2002]. The widest corpus of related work that we can find is

on so-called Visual Query System. Indeed, our system allows the user to (partly) visually query data sources similarly to what Visual Query Systems (VQSs) [Catarci *et al.*, 1997] do. VQSs are defined as systems for querying databases that use a visual representation to depict the domain of interest and express related requests. VQSs provide both a language to express the queries in a visual format and a variety of functionalities to facilitate user-system interaction. As such, they are oriented towards a wide spectrum of users, especially novices who have limited computer expertise and generally ignore the inner structure of the accessed database. Notable features of VQSs are the adopted *visual representation* of the database and the applicable language operators. The query representation is generally dependent on the database representation, since the way in which the query operands (i.e., data in the database) are presented constrains the query representation. For example, given a query on a relational database, we may state the query in terms of several representations, e.g., filling some fields in tables visualizing the relations, or following paths in a hyper-graph that visualizes the relational schema. In this case the table and the hyper-graph are two possible representations associated with the relational database. Typical visual representation are based on the use of forms, diagrams, icons, or a combination of them in the so-called hybrid representation. The SEWASIE interface adopts a hybrid representation in which diagrams are used to describe the ontology graph, while some icons are used to indicate actions to be performed. Forms are mainly used for displaying the query result.

The second key feature of VQSs refers to the *interaction strategies* provided to retrieve data. Data retrieval through interaction with a VQS is usually accomplished through the following two main activities:

- *Understanding the reality of interest.* The goal of this activity is the precise identification of the fragment of the schema the query refers to. Generally, the schema is much richer than the subset of concepts that are involved in the query. The result of this step is a query subschema, i.e. the static representation of all schema items that are needed to solve the query.

- *Formulating the query.* The query subschema can be manipulated in several ways, according to which query operators are provided. The goal of query formulation is to formally express the operations and operands that eventually make up the query.

Several strategies exist for each activity (see [Catarci *et al.*, 1997] for a complete classification) and some VQSs allow more than one of them for each

activity. This is also the case for the SEWASIE system. Indeed, as for the understanding the reality of interest, several mechanisms are available, or will be available, to filter the information considered significant by the user. The general idea is to follow a so-called *top-down* approach, providing the user with a strategy where general aspects of the reality are first perceived, and then specific details may be viewed. Similarly, there is the idea of exploiting *selective zoom*. In the case of selective zoom, the schema is unique, and the concepts are layered in terms of distance from the focus; the schema can be examined at several levels, so that only objects above a specified distance level are visible. Another well established technique for learning about the information content of a schema is *browsing*. In this case, browsing is essentially a viewing technique aimed at gaining knowledge about the data. The main hypothesis is that the user has only a minor knowledge about the data set and the interaction techniques. Within this hypothesis, the user starts the interaction by examining a concept and its neighborhood (adjacent concepts can be considered as a first level of explanation of the examined concept). Next, a new element is selected by the user from neighboring concepts to be the current one, and its neighborhood is also shown: this process proceeds iteratively.

An alternative approach to refinement and browsing is *schema simplification*. The idea here is to "bring the schema close to the query". This is done by building a user view resulting from aggregations and transformations of concepts of the original schema, and it is achieved in SEWASIE through the renaming mechanism.

Query formulation is the fundamental activity in the process of data retrieval. The SEWASIE query strategy *by navigation* has the characteristic of concentrating on a concept (or a group of concepts) and moving from it in order to reach other concepts of interest, on which further conditions may be specified. A second strategy for query formulation is *by sub-queries*. In this case the query is formulated by composing partial results.

Finally, it is often stressed that VQSs, being highly interactive systems, should be developed strictly following a user-centered design methodology [Norman and Draper, 1986]). Unfortunately, such a methodology does not seem to be completely pursued in actual VQS developments. In particular, scarce attention has been given to the range of users that might interact with a system and how the system should adapt to the different types of users. Also, not many usability experiments have been reported in the VQS literature. Whereas, SEWASIE development is strongly user-centered and deeply rely on user involvement and usability testing.

8 Conclusions

Only recently has research started to have an interest in query processing and information access supported by ontologies. Recent work has come up with advanced reasoning techniques for query evaluation and rewriting using views under the constraints given by the ontology – also called view-based query processing [Ullman, 1997; Calvanese *et al.*, 2000b; 2000a]. This means that the notion of accessing information through the navigation of an ontology modelling the information domain has its formal foundations.

This paper has presented the first well-founded intelligent user interface for query formulation support in the context of ontology-based query processing. This paper hopefully proved that 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 DAML+OIL and OWL.

Many open problems and refinements have still to be considered in our future work.

The system uses the verbalisations described in Section 4.2 to transform the conjunctive query into a natural language expression closer to the user understanding. However, as the examples presented earlier on show, the expressions still look somehow awkward. In the course of the Sewasie project some effort will be dedicated to explore semi-automatic techniques to rephrase the expressions in more succinct ways without loosing their logical structure.

Another important aspect to be worked out is the development of the associated methodologies for query formulation, a task that needs a strong cooperation of the users in its validation. This will go in parallel with the interface user evaluation, which is just starting at the time of writing this paper.

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.

References

[Adler and Winograd, 1992] P.S. Adler and T.A. Winograd. *Usability Turning Technology into Tools.* Oxford Press, 1992.

[Bresciani and Fontana, 2002] Paolo Bresciani and Paolo Fontana. A knowledge-based query system for biological databases. In *Proceedings*

of FQAS 2002, volume 2522 of *Lecture Notes in Computer Science*, pages 86–89. Springer Verlag, 2002.

[Bresciani and Franconi, 1996] P. Bresciani and E. Franconi. Description logics for information access. In *Proceedings of the AI*IA 1996 Workshop on Access, Extraction and Integration of Knowledge*, Napoli, September 1996.

[Bresciani *et al.*, 2000] Paolo Bresciani, Michele Nori, and Nicola Pedot. A knowledge based paradigm for querying databases. In *Database and Expert Systems Application*, volume 1873 of *Lecture Notes in Computer Science*, pages 794–804. Springer Verlag, 2000.

[Calvanese *et al.*, 2000a] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, 2000.

[Calvanese *et al.*, 2000b] D. Calvanese, G. De Giacomo, M. Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of the 15th IEEE Sym. on Logic in Computer Science (LICS 2000)*, 2000.

[Catarci *et al.*, 1997] Tiziana Catarci, Maria Francesca Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[Catarci *et al.*, 2003] Tiziana Catarci, Tania Di Mascio, Enrico Franconi, Giuseppe Santucci, Sergio Tessaris, and Guido Vetere. Evaluation of existing tools, description of the user centered interface design. Technical Report D6.1, SEWASIE Consortium, 2003.

[Chandra and Merlin, 1977] A.K. Chandra and P.M. Merlin. Optimal implementation on conjunctive queries in relational data bases. In *Proc. 9th ACM Symp. on Theory of Computing*, pages 77–90, 1977.

[Dix *et al.*, 2001] A. Dix, J.Finlay, G. Abowd, and R. Beale. *Human Computer Interaction*. Prentice Hall press, 2001. second Edition.

[Franconi *et al.*, 2003] Enrico Franconi, Sergio Tessaris, Tiziana Catarci, Tania Di Mascio, Giuseppe Santucci, and Guido Vetere. Specification of the ontology design tool. Technical Report D6.2, SEWASIE Consortium, 2003.

[Franconi, 2000] Enrico Franconi. Knowledge representation meets digital libraries. In *Proc. of the 1st DELOS (Network of Excellence on Digital Libraries) workshop on "Information Seeking, Searching and Querying in Digital Libraries"*, 2000.

[Halpin, 2001] Terry A. Halpin. Augmenting UML with fact orientation. In *HICSS*, 2001.

[ISO-13407, 1996] ISO-13407. Human centred design processes for interactive systems. International Organisation for Standardisation, ISO/IEC 13407-2, 1996.

[Norman and Draper, 1986] D. Norman and S. Draper. *User Centered System Design.* LEA, Hillsdale, N.J., 1986.

[ORM, 2003] `http://www.orm.net`, 2003.

[Ullman, 1997] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf on Database Theory (ICDT'97)*, pages 19–40, 1997.