Ontology-Based Data Integration Methods: A Framework for Comparison

Agustina Buccella[*], Alejandra Cechich[*] and Nieves R. Brisaboa[†]

Abstract

A data integration system provides a uniform interface to distributed and heterogeneous sources. These sources can be databases as well as unstructured information such as files, HTML pages, etc. One of the most important problems within data integration is the semantic heterogeneity, which analyzes the meaning of terms included in the different information sources. This survey describes seven systems and three proposals for ontology-based data integration. An important feature is that all of them use, in some way, ontologies as the way to solve problems about semantic heterogeneity. In this paper, we show similarities and differences among the systems by providing a framework for comparison and classification.

Keywords: Data Integration, Ontology, Semantic Heterogeneity.

1. Introduction

Data integration is concerned with unifying data that share some common semantics but originate from unrelated sources. Necessarily, when we work on data integration, we must take into account a more important and complex concept called "heterogeneity".

Heterogeneity might be classified into four categories: (1) *structural heterogeneity*, involving different data models; (2) *syntactical heterogeneity*, involving different languages and data representations; (3) *systemic heterogeneity*, involving hardware and operating systems; and (4) *semantics heterogeneity*, involving different concepts and their interpretations. Generally speaking, the semantic heterogeneity deals with three types of concepts [14]: the *semantically equivalent concepts*, the *semantically unrelated concepts*, and the *semantically related concepts*. In the first case – semantically equivalent concepts – a model uses different terms to refer the same concept, e.g. synonymous, or some properties are modelled differently by different systems, for example the concept length may be "meter" in one system and "mile" in one another. In the second case – semantically unrelated concepts – the same term may be used by different systems to denote completely different concepts; and in the last case – semantically related concepts – different classifications may be performed, for example one system classifies "person" as "male" and "female" and other system as "student" and "professor".

There are several methods created to address the problem of dealing with different concepts and interpretations. In general, the approaches might be divided into two different branches: approaches using ontologies and approaches without using ontologies (for example using metadata [10,32]). In this paper we will focus on the use of ontologies because of their advantages when using for data integration. Among them:

(1) the vocabulary provided by the ontology serves as a stable conceptual interface to the databases and is independent of the database schemas,
(2) the language used by the ontology is expressive enough to address the complexity of queries typical of decision-support applications,
(3) knowledge represented by the ontology is sufficiently comprehensive to support translation of all the relevant information sources into its common frame of reference, and
(4) the ontology supports consistent management and recognition of inconsistent data.

The term *ontology* was introduced by Gruber [20] as an "explicit specification of a conceptualization". A *conceptualization*, in this definition, refers to an abstract model of how people commonly think about a real thing in the world; and *explicit specification* means that concepts and relationships of an abstract model receive explicit names and definitions.

[*] Universidad Nacional del Comahue, Departamento de Ciencias de la Computación, Neuquén, Argentina, 8300. Email: abuccel,acechich@uncoma.edu.ar

[†] Universidade de A. Coruña, Departamento de Computación, A. Coruña, España, 15071, Email: brisaboa@udc.es

An ontology gives the name and the description of the domain specific entities by using predicates that represent relationships between these entities. The ontology provides a vocabulary to represent and communicate domain knowledge along with a set of relationships containing the vocabulary's terms at a conceptual level. Therefore, because of its potential to describe the semantic of information sources and to solve the heterogeneity problems, an ontology might be used for data integration tasks.

Some surveys on ontology-based systems for data integration can be found in literature. For example, in [38] authors provide a survey specially focusing on the ontology use, mappings and tools, but without providing a comparison of other elements, such as query resolution or architectural issues. Another different survey can be found in [12], but only languages to represent ontologies are compared in this case.

In this paper, we show the state-of-the-art in ontology-based systems for data integration by describing seven systems – SIMS [1,2,3], OBSERVER [30,31], DOME [13,14], KRAFT [19,33,34], Carnot [28,39], InfoSleuth [8,16,41] and COIN [17,18,35] – and three new proposals. We define a conceptual framework for comparison, which takes into account the most important aspects used to achieve data integration. For example, all of the systems produce ontologies in different ways. Some of them define only one ontology while others define multiple ones. In all cases, the ontology describes the application domain by mapping the information sources to other ontologies. Another important aspect is the ontology language. In general, the systems use languages based on Description Logics, although some Web–based languages have recently emerged. In this paper, we will only indicate the language used to represent ontologies without further explanation. For a description and analysis of ontology languages, we refer the reader to [12].

A framework is proposed for evaluating the systems based on the DESMET [24] approach for Feature Analysis, which will be explained in the next section. The paper is organized as follow: Section 2 describes the conceptual framework and introduces the ontology-based systems in terms of the framework's elements. Section 3 presents the comparison of the systems and the proposals using the DESMET evaluation method. Conclusions are discussed in the final section.

2. A Framework for Comparison of Data Integration Approaches

In order to compare the different approaches, we use the DESMET method [24] due to its widespread use in evaluation techniques. The DESMET method is intended to help an evaluator in a particular organisation or academic institution to plan and execute an evaluation exercise that is unbiased and reliable (e.g. maximises the chance of identifying the best method/tool).

DESMET uses the term Feature Analysis to describe a qualitative evaluation. Feature Analysis is based on the identification of the requirements that users have for a particular task and the mapping of those requirements to features that a method/tool aimed at supporting that task should possess. In the context of software methods/tools, the users might be the software managers, developers or maintainers.
An evaluator then assesses how well the identified features are provided by a number of alternative methods/tools. The main processes involved in carrying out a feature analysis are:

1. Select a set of candidate method/tools to evaluate.
2. Decide the required properties or features of the item being evaluated.
3. Prioritise those properties or features with respect to the requirements of the method/tool users.
4. Decide the level of confidence that is required in the results and therefore select the level of rigour required of the feature analysis.
5. Agree on a scoring/ranking system that can be applied to all the features.
6. Allocate the responsibilities for carrying out the actual feature evaluation.
7. Carry out the evaluation to determine how well the products being evaluated meet the criteria that have been set.
8. Analyse and interpret the results.
9. Present the results to the appropriate decision-makers.

A common framework is necessary to make any sort of comparative evaluation of methods and tools. Feature analysis allows the framework to be expressed in terms of a set of common (mandatory and/or desirable) properties, qualities, attributes, characteristics or features for each type of method or tool. After defining a framework, a method or tool has to be judged as to how much support it actually appears to provide for a feature, i.e., to what degree support is present. Once each method or tool has been "scored" for each feature in the framework using some common judgement scale, the results for the methods or tools have to be compared to decide their relative order of merit.

There are two types of features:

1. Simple features that are either present or absent. These are assessed by a simple YES/NO nominal scale.
2. Compound features where the degree of support offered by the method/tool must be measured or judged on an ordinal scale.

Each simple feature can be accompanied by a degree of *importanc*e assessment. Each compound feature must be accompanied by an assessment of its importance and an appropriate judgment scale associated with the degree of existence or *conformance* to a particular feature or characteristic. The importance of a feature can be assessed by considering whether it is mandatory or only desirable. There may be many gradations of "desirability", but in this paper we only use three of them: *Mandatory* (M), *Highly Desirable* (HD) and *Desirable* (D).

A judgement scale for conformance performs two distinct purposes:

1. It defines what level of support of a particular feature is required.
2. It provides the assessor with a consistent measurement scale against which to "score" the feature of a particular candidate.

The granularity of the scale points depends upon the feature that is being looked at, and the actual requirements. In this paper we will use the generic judgment scale defined in the Table 1 for a particular feature.

Generic Scale Point	Definition of scale point	Scale point mapping
Makes things worse	Cause confusion. The way the feature is implemented makes it difficult to use and/or encouraged incorrect use of the feature.	-1
No support	Fails to recognize it. The feature is not supported nor referred to in the specification manual.	0
Little support	The feature is supported indirectly, for example by the use of other tool features in non-standard combinations.	1
Some support	The feature appears explicitly in the feature list of the tools and specification manual. However, some aspects of feature use are not catered for.	2
Strong support	The feature appears explicitly in the feature list of the tools and specification manual. All aspects of the feature are covered but use of the feature depends on the expertise of the development team.	3
Very strong support	The feature appears explicitly in the feature list of the tools and specification manual. All aspects of the feature are covered and the tools provide tailored dialogue boxes to assist the development team.	4
Full support	The feature appears explicitly in the feature list of the tools and specification manual. All aspects of the feature are covered and the tools provide scenarios to assist the development team such as "wizards".	5

Table 1. Generic judgment scale

Our framework for comparison of data integration approaches (or systems) is described and motivated by the main concepts explained in the introduction. All the systems to be compared will be analyzed from viewpoint of a development team, working in an institution with data integration problems. The framework includes relevant features helping the choice of the appropriate system. As Figure 1 shows, the framework is divided into three main features: *architectur*e, *semantic heterogeneity* and *query resolution*. Each feature contains a set of sub-features or items that further describe the framework. Each of them deals with different aspects of the systems in order to describe them and analyze their advantages and disadvantages. The meaning of each feature is:

(1) Architecture:

- **Information Sources**: This feature refers to the information sources supported by the different systems. A main part of an architecture based on data integration are the information sources involved in the integration, that is, the systems use these sources to retrieve the information and return it to the users. The systems have several architectural components within the architecture in order to access the different information sources. In this paper, the types of information sources are divided into two categories: *the State of the Information Sources (SIS)* and *the Type*

of the Information Systems (TIS). The first category divides the information sources into *dynamic* and *static*. When the information sources are dynamic the systems should create some mechanisms so as to know which information is available at a given moment. The second category indicates the three main types of information systems supported by the systems. They are databases, files and HTML pages.

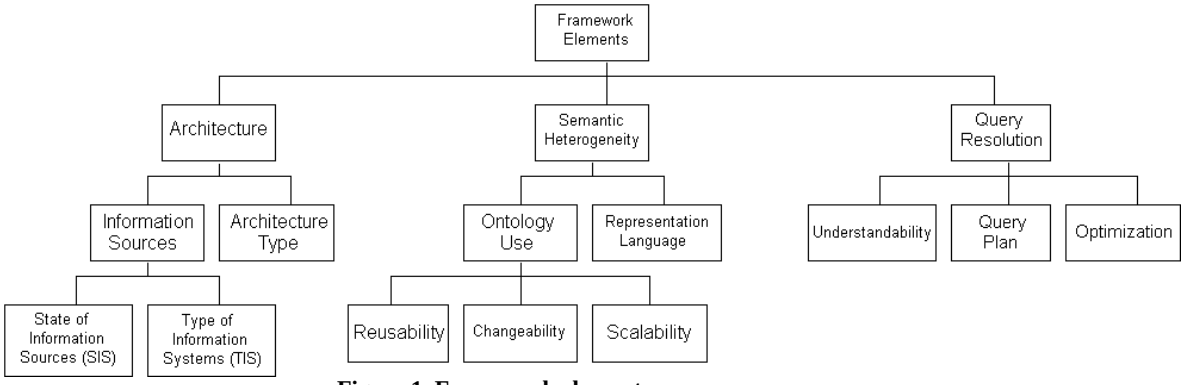

Figure 1. Framework elements

- **Architecture Type**: The software architecture of a system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them [6]. Each system has defined its architectural components based on the user's requirements as well as the need for the working environment. The architecture type can be divided into two types: agent-based and wrapper/mediated-based. Each of them has their own advantages and disadvantages. For example, two main advantages of the agent-based architecture are: (1) agent architectures are designed to allow software processes to communicate knowledge across networks, in high-level communication protocols and (2) agent architectures are highly dynamic and open, allowing agents to locate other agents at run-time, discover the capabilities of other agents, and form cooperative alliances. But this architecture type has also two main disadvantages: (1) the communication among the large number of agents necessary to implement an application may become too expensive and (2) how to understand multiple agents interacting may be difficult, especially if the number of agents is large. On the other hand, the mediator-based architecture has as advantage that all the information needed to achieve the integration is stored in the mediator but this component can make itself appear very complex and difficult to manipulate. Also, performance aspects must be taken into account when mediators are used.

(2) Semantic Heterogeneity:

- **Ontology use**: The first section has described the meaning of the ontologies and how they help solving the data integration problems. The systems should show how the ontologies are used to solve the integration problems and how the ontologies interact with the other components of the architecture. In this feature the system describes its ontological components and the ways to solving the different semantic heterogeneity problems. The development team should find a set of steps or ways to build the ontologies defined in the system. This feature is divided into three sub-features: *reusability*, *changeability* and *scalability*. Reusability refers to the ability of reuse the ontologies, that is, ontologies defined to solve other problems can be used in the system because of either the systems support different ontological languages and/or define local ontologies. Changeability refers to the ability of changing some structures within an information source, without producing substantial changes in the system components. Finally, scalability refers to how easy the integrated system can be extended with new information sources.
  In order to clarify the "Ontology use" feature, Figure 2 shows a diagrammatic representation of part of one ontology used to compare different systems. The arrows in the figure represent properties between two classes. Attributes (or properties relating classes to data types) are represented within class boxes. As we can see, the ontology is modelling an airport's domain.

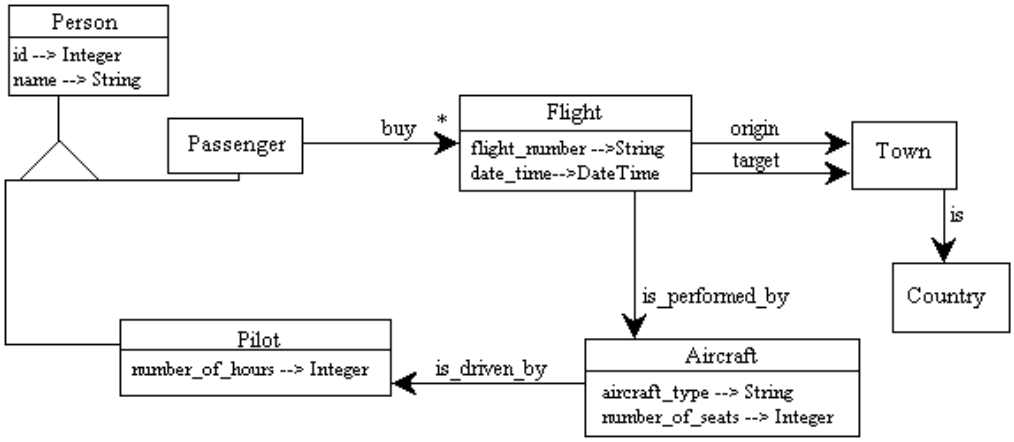

Figure 2. Part of one ontology

Note that Figure 2 only shows a small part of the ontology we have used to illustrate our comparisons.

Now, by applying the sub-features of "Ontology use" to our example, we can see that changeability and scalability impact on some structures by modifying or adding (respectively) the information sources. Depending on the implementation of each system, this change can make the whole ontology be rebuilt or only few modifications be incorporated. Following the example in Figure 2, the next subsections will show how the systems implement the ontologies in order to reach the sub-features.

- **Representation Language**: Each system uses different languages in order to represent the ontologies. Each language has different characteristics with their advantages and disadvantages. The systems should indicate the supported language/s to represent the ontologies and give some support on how the systems use the language/s. For example, either the systems support only one language or support several languages or no restriction of the languages is required. Also, the system should offer some guide to show the language application. Comparison among the languages is out of the scope of this paper. A framework for comparing the expressiveness of the languages can be found in [12].

(3) Query Resolution:

- **Understandability**: In the integrated system users have a unique user interface to access it. Therefore, this user interface should be simple and easy to use. Besides, some guide explaining the interface should be provided. The *understandability* is one of the main quality attribute when user interfaces are thought of. The understandability refers to the capability of the software product to enable the user to understand whether the interface is suitable, and how it can be used for particular tasks and conditions of use. Also, the answer given for the system should be clear and easy for the users, that is, in a language understood by them.
- **Query Plan**: This feature refers to the set of steps needed to achieve a query defined by the users. The system should give a clear description of each step involved showing the interaction among components and ontologies to get a suitable semantic answer. With this feature the development team will understand how the whole system works using the functions of each component.
- **Optimization**: The query optimization can dramatically speed up database query. Many systems have implemented methods to optimize the queries. For example, one method would divide the query into sub-queries, optimize them, and thus generate a faster answer. The comparison among the optimization methods is out of the scope of this paper. We only indicate the optimization methods used by the different systems.

Figure 3 shows the classification into simple and compound features applied to our framework.

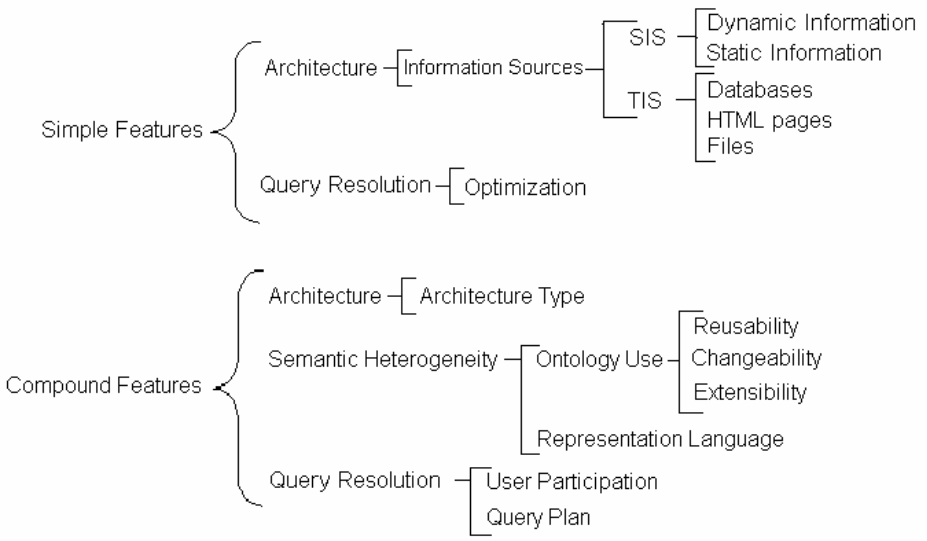

Figure 3. Classification of features between simple and compound

Table 2 shows each feature accompanied by an assessment of its importance. The information sources, architecture type, ontology use and query plan features are mandatory because the minimal needed elements are included within each of these features. When one system does not explain or describe how the ontologies are used, the users or the development team will probably apply them in an incorrect way resulting in an awful system.

Specific Feature	Importance Assessment
Information Sources	M
Architecture Type	M
Ontology Use	M
Representation Language	D
Understandability	HD
Query Plan	M
Optimization	D

Table 2. Importance Assessment for specific features

On the other hand, the representation languages and optimization features are only desirable because the systems may be used properly although these features are incomplete. The same happens with the understandability feature, but in this case the degree of importance is little higher.

In the next sections, we briefly describe seven systems and three proposals, in accordance with the characteristic defined by our framework.

2.1 SIMS (Search In Multiple Sources)

SIMS appeared in 1993 as an information mediator [1,2,3]. The system essentially provides access and integration to multiple sources of information.

Architecture:

- *Information sources*: SIMS was created assuming dynamic information sources, i.e. changing information sources, availability of new information, etc. In general, sources are databases, but SIMS researchers are currently working on including other kinds of information sources, such as HTML pages.
- *Architecture type*: The architecture is based on a *wrapper/mediator*, that is, each information source is accessed through a wrapper. A wrapper, in SIMS, is a module that can translate a data set description into a query, which is submitted to the source. The wrapper also handles communication with the information source. Data returned by the source are taken and sent to the SIMS in some expected format. The SIMS mediator component is used to unify the various available information sources and to provide the terminology for accessing them. The core part of the mediator is the ability to intelligently retrieve and process data.

Semantic Heterogeneity:

- *Ontology use*: A domain model provides the general terminology for a particular application domain. Each information source is incorporated into SIMS by describing the data provided by that source in terms of the domain model. This model is contained in the mediator. SIMS uses a global domain model that also can be called a *global ontology*. The work presented in [38], classifies SIMS as a *single ontology approac*h. In this case, the ontology is composed of the concepts (classes and attributes) that are available from the source, the name of the source that provides the data (also the host, db-name, etc.) and the mappings between them. For example, if two or more sources have attributes with the same values, then they are linked to the same domain concept. A source concept may contain attributes that are not linked to any domain relation.

  Regarding to our example, Figure 2 represents the global ontology for the SIMS system; then any change in the information sources will impact directly on the global ontology. For example, if some information source add information about air-stewardess, the global ontology has to be modified to include this information. Figure 4 shows the changes made to the global ontology in order to represent the new information.

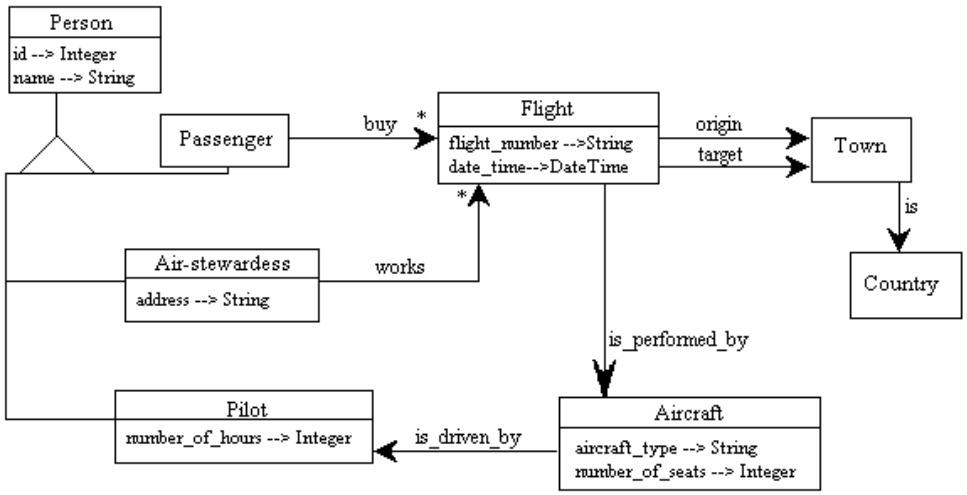

Figure 4. Changes in the global ontology

In this simple example, we add two new elements: the *Air-stewardess* class, as a subclass of the *Person* class, and a property between the *Air-stewardess* class and the *Flight* class. Note that this ontology will be very large, since it will contain all the information in the sources. Therefore, any change or addition on the source's structure will be complex to be implemented in the global ontology.

- *Language*: The domain model (ontology) is described in the Loom language [27]. In Loom, objects in the world are grouped into classes and subclasses. One class can have any number of subclasses, but SIMS does not allow multiple hierarchies. We can define relationships between a subclass and its superclass by explicitly describing the constraints on the subclass. We can also establish relationships between classes. Finally, each class or subclass has a set of attributes.

Query:

- *User participation*: Users make a query in terms of the global ontology without knowing the terms or language used by the underlying information sources. Therefore, queries are written in a high-level language (Loom or a subset of SQL) that actually is the domain model terminology. Queries do not contain information indicating which sources are relevant to their execution or where the sources are located.
- *Query plan*: The first step to answer a query is transforming it into another query expressed in terms of concepts that correspond to information sources. In order to achieve this goal, the system applies four reformulation operations. The *Select-Information-Source* operation maps the concepts in the ontology to the concepts in the information sources, the *Generalize-Concept* operation uses the superclass concept of the ontology when a specific required concept is not in the information sources, the *Specialize-Concept* operation uses the subclass concept in the same way that in the

previous case, and the *Decompose-Relation* operation replaces a particular ontology's relation with the indicated in the information source. The second step consists of generating a query plan and processing the data. To do this, the system divides the query into subqueries trying each of them separately. Once an execution plan has been produced, it is sent to the optimization system.

- *Optimizatio*n: The part of the system that makes the optimization is called *Semantic Query Optimization* (SQO). It optimizes a single subquery and, after that, the entire query plan. SIMS use SQO [21] that can speed up database query answering by using knowledge intensive reformulation.

2.2 OBSERVER (Ontology Based System Enhanced with Relationship for Vocabulary hEterogeneity Resolution)

OBSERVER is an approach that proposes managing multiple information sources through ontologies [30,31].

Architecture:

- *Information sources*: Like SIMS, OBSERVER was created assuming dynamic information sources, but any type of information source such as HTML pages, databases or files, is supported in this case. OBSERVER uses the concept of *data repository,* which might be seen as a set of entity types and attributes. Each repository has a specific data organization and may or may not have a data manager. The different data sources of a repository might be distributed.
- *Architecture typ*e: The architecture is based on *wrappers, ontology servers* and an *IRM* (Inter-ontology Relationship Manager). Here, a wrapper is a module, which understands a specific data organization and knows how to retrieve data from the underlying repository hiding this specific data organization. With Internet sites as data repositories, a wrapper emulates the behaviour of a user, who is accessing data from a Web site. The architecture has several nodes, in which a set of data repositories may have their own wrappers. The Ontology Servers and the IRM will be explained below.

Semantic Heterogeneity:

- *Ontology us*e: In [38], OBSERVER is classified as a *multiple ontology approac*h. OBSERVER defines a model for dealing with multiple ontologies avoiding problems about integrating global ontologies. The different ontologies (user ontologies) can be described using different vocabularies depending on the user's needs. In OBSERVER, every node has only one *Ontology Server*, which is a module that provides information about ontologies located on the node as well as about their underlying data repositories. The system allows encapsulating any direct interaction with user ontologies and also any access to data repositories. Another important component of the OBSERVER system is the *IRM* shared repository. It can be seen as a *catalog of semantics* of the system used to solve the "vocabulary problem" (heterogeneous vocabularies used to describe the same information). OBSERVER also deals with several kinds of interoperable relationships (synonym, hyponym, hypernym, overlap, disjoint and covering).
  In our system, each information source is represented by one ontology; thus a modification or addition of information to some source will only impact on the related ontology and on the IRM. For example, if we add information about air-stewardess, the same changes have to be made to the ontology obtaining the model shown in Figure 4 as a result. But this user-ontology will be smaller than the global ontology in the SIMS system, since it will contain only the information of one source. Therefore, any change or addition on the source's structure will be easier to be implemented.
- *Languag*e: A user may use any language to represent an ontology, but in general, a language based on DL (Description Logics) such as CLASSIC [9] or Loom [27] is used.

Query:

- *User participatio*n: There are two tasks in which a user is involved. During the task *Select User Ontology,* users can browse terms and its definitions, both in DL and in a natural language, in order to choose the terms that exactly fit the semantics of the query. The browse is performed by navigating the different ontologies. During the *Edit Query* task, after selecting the user ontology, the user chooses terms from the ontology to build the constraints and projections that comprise the query. Both tasks together are called *Query Construction.*
- *Query pla*n: The element of the architecture that makes the query resolution is called *Query Processo*r. There are three main steps to answering a query. First of all, the *Query Construction* is performed and it is followed by the *Access to Underlying Data,* and the *Controlled Query Expansion to New Ontologies* steps. The second step translates the user query, expressed in terms of the user ontology, into different queries to the underlying data repositories. To do this, the *Query Processor*

calls the *Ontology Server* that uses some *Mapping Information* to translate the user query, linking the ontology and the underlying data elements. The latest step is executed when the user wants to obtain more relevant data. In this case, the user can choose other ontologies to visit. Then, the original query is translated from terms of the user ontology into terms of another ontology. Obviously the translation is not always exact and the user may define a limit for the *Loss of Information* (loss in precision) allowed.

- *Optimizatio*n: A query is divided into subexpressions that involve only one data repository, although a term in an ontology can be related to several repositories. The cost estimation is concerned with the concept of *Loss of Information* previously mentioned. This concept appears when a user wants to expand the query to other ontologies obtaining more relevant data. Two types of plans can be built in this case: *Plans without loss of information* (when a complete translation of the user query can be achieved, for example by using synonymous) and *Plans with loss of information* (when a complete translation cannot be achieved, for example by using hyponym and hypernym). The user can indicate a desired limit of loss of information (a percentage). The system provides a theoretical basis for estimating information loss measures.

2.3 DOME (Domain Ontology Management Environment)

DOME [13,14] is still under development. It is focused on ontology development by using software reverse engineering techniques.

Architecture:

- *Information sources*: DOME only deals with structured databases and their application programs where the data sources are legacy information systems. However, DOME researchers are currently working on including semi-structured data sources such as Web pages as well.
- *Architecture typ*e: The most important architectural components are wrappers, a set of tools for extracting and defining ontologies and mappings between them (grouped into the Engineering Client component), the Mapping Server, and the Ontology Server. Here, a wrapper performs a translation of queries expressed in the DOME query syntax to queries expressed in the syntax of the data sources.

Semantic Heterogeneity:

- *Ontology us*e: The better approach to describe the DOME system is the *multiple ontology approach* [38]. But regarding to the *Client Engineering* approach, it is also possible to combine a *top-down* and a *bottom-up* ontology development approaches. In this case, the top-down approach identifies key concepts by consulting corporate data standards, information models or generic ontologies such as Cyc or Word Net. This process results in *shared ontologies*, which are used as the common terminology between a number of different systems. The bottom-up approach starts with the underlying data sources and uses ontology extraction tools, such as XRA [42], which are applied to database schemas and application programs to produce initial ontologies. They are further refined into two separate ontologies: *resource ontologies*, which define the terminology used by specific information resources; and *application ontologies*, which define the terminology of a user-group or client-application. Once the ontologies have been defined, they are stored in the Ontology Server. The rest of the ontology engineering task consists of defining a mapping between the resource and the shared ontologies by a particular application. As we previously mentioned, DOME uses XRA [42] as a tool to generate ontologies. XRA is an ontology extraction tool that uses a software reverse engineering approach to extract an initial ontology from given data sources and their application programs.

  In DOME system, like in OBSERVER, Figure 2 only represents an ontology (resource ontology) with semantic information of one source. The modification or addition of a source will change this ontology, and also the mappings between this resource ontology and application and shared ontologies. For example, Figure 5 shows part of one shared ontology. As we can see, it contains information about air-stewardess but it is represented using a different concept (synonym) and a relationship. Therefore to finish the process of adding information, the following mappings have to be added:

  Flight_Assistant ⇔ Air-stewardess
  Flight_Assistant.assists ⇔ Air-stewardess.works

  Some other mappings already stored in the system are:

Airplane_Pilot ⇔ Pilot
Airplane ⇔ Aircraft
Airplane.airplane_type ⇔ Aircraft.aircraft_type
Person ⇔ Person
……..

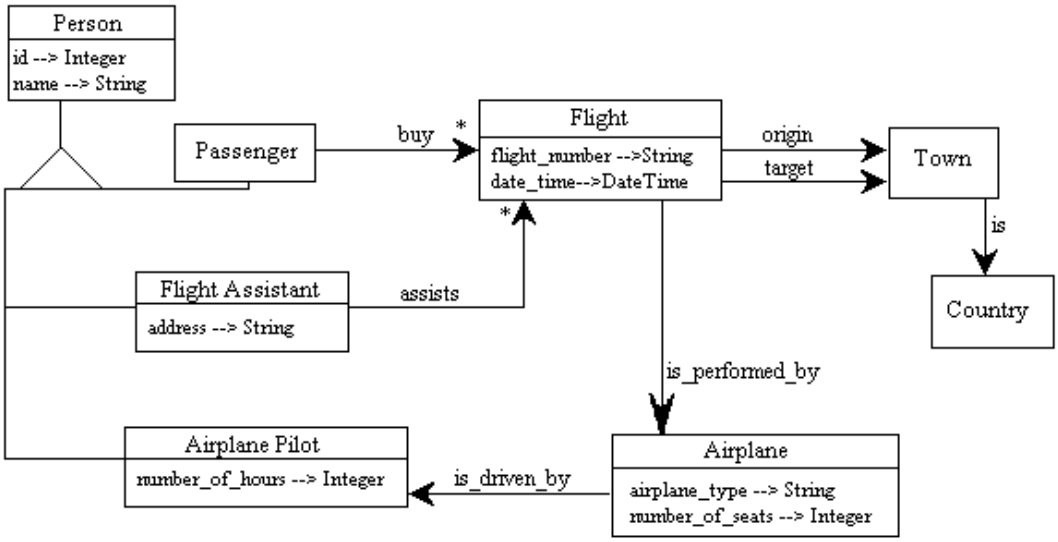

Figure 5. The shared ontology in DOME

- *Language*: The three ontologies previously mentioned are specified using CLASSIC [9], which is used to store ontologies and to make some inferences.

Query:

- *User participation*: DOME has a simple API that allows a user client or an application to query the distributed information space. The user can load and browse specific ontologies to view what information is available from the whole information space. Queries are performed using the terminology defined by an *application ontology* and the returned results are represented using that terminology. However, the details of the different systems – their distribution, structure, syntax or semantics – are actually hidden.
- *Query plan*: The element of the architecture that performs queries is called *Query Engine*. Firstly, the Query Engine needs to decide which resources are relevant to a given query. Then, based on the information about the currently relevant resources, it decomposes the query into subqueries. Results are integrated after been received.
- *Optimization*: There is no cost estimation that shows the best way to achieve a query. As we previously mentioned, the query is divided into subqueries to access the information sources.

2.4 KRAFT (Knowledge Reuse And Fusion/Transformation)

KRAFT [19,33,34] was primarily conceived to support configuration design of applications among multiple organizations with heterogeneous knowledge and data models. It uses the concept of *"Knowledge fusion"* to denote the combination of knowledge from different sources in a dynamic way.

Architecture:

- *Information sources*: Like the other systems, KRAFT was created assuming dynamic information sources. Here, information sources are called *resources* and include both databases and knowledge bases. Resources have constraints that may be stored as metadata of the database, as triggers, etc.
- *Architecture type*: KRAFT has an agent-based architecture, in which all knowledge processing components are realized as software agents. There are four kinds of agents: *wrappers*, *mediators*, *facilitators* and *user agents*. The *wrapper agents* provide a bridge among the legacy system interface, the KRAFT agent interface, and entry-points for user agents. The *user agents* allow end-users to access the information by using some kind of user interface. The *facilitator agents* provide internal routing services for messages within the KRAFT domain. When an agent needs a service, it

asks a facilitator to recommend another agent that provides the service. Finally, the *mediator agents* use the domain knowledge to transform data in order to increase its information content. Examples of data transformation might be integration of data from heterogeneous sources, consistency checking and refinement of knowledge and data, etc.

Semantic Heterogeneity:

- *Ontology use*: The better approach to describe the KRAFT system is the *hybrid ontology approach* [38]. In order to overcome the problems of semantics heterogeneity, KRAFT defines two kinds of ontologies: a *local ontology* and a *shared ontology*. For each knowledge source there is a local ontology. The shared ontology formally defines the terminology of the domain problem. In order to avoid the problems about semantics heterogeneity, that might occur between a local ontology and the shared ontology (*ontology mismatches* [37]), an *ontology mapping* is defined for each knowledge source. It is a partial function that maps terms and expressions defined by a local ontology to terms and expressions of the shared ontology.
  Something similar occurs in the KRAFT system compared with DOME and OBSERVER. The main difference here is that KRAFT contains a shared ontology. When a modification or addition is made in some source, the local ontology (which represents this source) has to be changed, and the mappings between the local and the shared ontology have to be performed. To illustrate this situation we can apply the same example used in the DOME system, because Figure 5 represents one shared ontology in that system. In KRAFT, Figure 5 represents the only shared ontology containing all the information that the system needs to be accessible for users.
- *Language*: The local and shared ontologies may not be represented in the same format, even though they use classical frame-based representation languages.

Query:

- *User participation*: Users access the services of the KRAFT domain via *user agents*. Users specify their requirements as constrains written in any language (KRAFT Constraint Interchange Format, CIF, CoLan languages [7]). So, in order to translate these constraints, the terminology used to perform the queries must be defined in the shared ontologies.
- *Query plan*: The resource constraints must be translated into CIF constraints before the KRAFT network can use it. Communication between agents is made through messages, which include the CIF constraints as part of them. The main function of the facilitator agents is to solve a query called *facilitation*. The goal of this query is to find a resource whose advertised capability matches the requirements derived from the query. The satisfiability criterion depends on the search strategy adopted for the resolution of the query. There are two resolution mechanisms – *the most exact match* and *the set of all approximate matches*.
- *Optimization*: There is no mechanism to provide optimization.

2.5 Carnot and InfoSleuth

The Carnot Project [28,39] was initiated in 1990 with the goal of addressing the problem of logically unifying physically distributed, heterogeneous information. The InfoSleuth project [8,16,41] is an extension of Carnot to make legacy database systems easily accessible via Web.

Architecture:

- *Information sources*: InfoSleuth was created to add the concepts of dynamic information sources to Carnot. It was designed to operate on a static environment, where the information sources never change.
- *Architecture type*: Like KRAFT, these systems have an agent-based architecture to provide interoperation among autonomous systems. There are five kinds of agents in InfoSleuth. The *user agents* assist the user in formulating queries over an ontology, and in displaying the results of queries in a sensitive manner to the user context. The *broker agents* determine the set of relevant resources that can perform the requested service. The *resource agents* translate queries expressed in a common query language into a language understood by the underlying system. These agents usually perform a mapping between the ontology and the local data. The *task execution agents* are designed to deal with dynamic, incomplete and uncertain knowledge. These agents use information provided by a *broker agent* to route requests to the specific *resource agents*. Finally, the fifth type – *ontology agents* – will be explained in the following point.

Semantic Heterogeneity:

- *Ontology use*: Ontologies in InfoSleuth system are used to capture database schemas, conceptual models and aspects of its agent architecture. Here, there are two main tasks to accomplish. The first

one is concerned with *describing the information sources* as the system may have multiple ontologies describing data. The second task aims at *specifying the agent infrastructure*, i.e. the context in which agents operate, its relevant information and relationships, etc. InfoSleuth implements a model with three layers to represent ontologies: the *Frame Layer*, which allows to create and query new meta-models, the *Meta-Model Layer*, in which objects are instances of frame layer objects, and the *Ontology Layer*, in which objects are instances of meta-model objects. Therefore, the most appropriate approach to describe the InfoSleuth system is the *multiple ontology approach* [38]. Otherwise, Carnot is implemented using the *single ontology approach* due to the system provides a global view of the all integrated resources.

Regarding to our example, the same case applied in the SIMS system can be applied in the Carnot system because it also define a global ontology (or view) about the whole domain. In the case of InfoSleuth, the three-layer model is represented by Figure 6, in which the Airport-Ontology box in the Ontology layer is the ontology described in Figure 2.

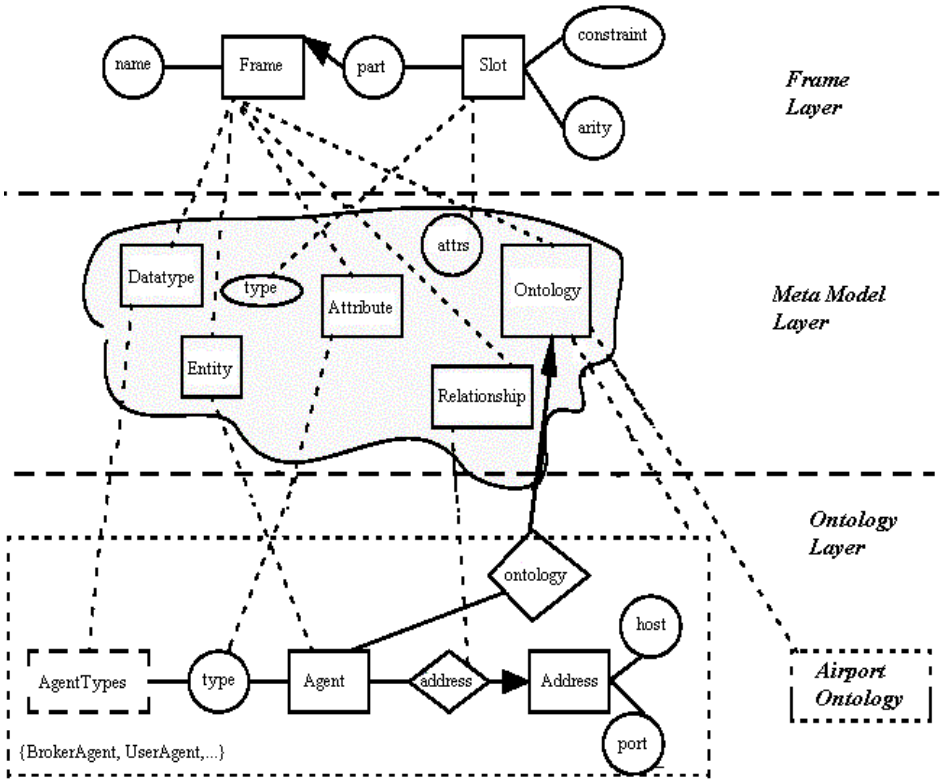

Figure 6. The three-layer model in InfoSleuth

Any change in the subjacent source generates the modification of the Airport Ontology. This case is similar to the application of OBSERVER because each source will have its own ontology, and therefore changes will be easier to be implemented.

- *Language*: There is no specific language required for the system. The ontologies might be represented, for example, by using logic programming languages such as LDL [43].

Query:

- *User participation*: Users interact with the system through a *user agent* specifying requests and queries over given ontologies. In this way, a *user agent* obtains information from an *ontology agent* about the common ontological models known by the system. A user agent uses this information to support its associated user in selecting an ontology and formulating a set of queries.
- *Query plan*: When a user performs a query, the *user agent* asks to the *broker agent* for the location of an applicable *execution agent*. When the *execution agent* receives the request, it asks to the *broker agent* for the location of the *ontology agent* and for an appropriate ontology. Then, the *broker agent* is asked for the currently *resource agent*. Depending on the resource availability, the *broker agent* may return a different set of *resource agents* at different time. Then, based on this set, the *execution*

agent decomposes the query and routes it. Each resource agent, after translating the query, returns the results to the *execution agen*t. After receiving the results, the execution agent integrates and returns these results to the *user agen*t, which in turn returns them to the user. On the other hand, Carnot system uses its *query graph generator module* to generate a query graph. This graph is expanded by a *semantics module* including sources with relevant information. Finally, an optimal plan is generated and executed.

- *Optimizatio*n: The query processing strategy in Carnot is based on a dataflow execution model similar to the strategy used in the ORION distributed object-oriented database system [40].

2.6 COIN (Context Interchange)

The COIN Project [17,18,35] was initiated in 1991 with the goal of achieving semantics interoperability among heterogeneous information sources.

Architecture:

- *Information sources*: The COIN system provides access to both traditional data sources such as databases and semi-structured information sources such as Web sites.
- *Architecture typ*e: The system has a mediator-based architecture. The main elements of this architecture are *wrappers, context axioms, elevation axioms, a domain model, context mediators, an optimizer* and *a executione*r. Like other systems, here a *wrapper* is a module that understands a specific data organization and knows how to retrieve data from the underlying repository hiding that data organization. The other architectural elements will be explained below.

Semantic Heterogeneity:

- *Ontology us*e: COIN introduces a new definition for describing things in the world. It states that *the truth of a statement can only be understood with reference to a given contex*t. Here, the notion of *context* is useful to model statements in conflicting heterogeneous databases. Using this definition, COIN creates a framework that constitutes a formal and logical specification of the COIN system components. The framework has three components, which have been previously introduced in the architectural analysis: a *domain mode*l, *elevation axioms* and *context axiom*s. Each information source has a set of elevation axioms and a set of context axioms, and both converge in a unique domain model. Therefore, the most appropriate approach to describe the COIN system is the *hybrid ontology approach* [38]. The *domain model* is a collection of source's primitive types, such as strings or integers, and semantics types that define the application domain corresponding to the integrated data sources. Primitive objects and semantics objects are instances of primitive types and semantics types respectively. Semantics objects may have properties called *modifiers* that serve as annotations to make the data's semantics of different contexts explicit. Also, semantics objects may have different values in different contexts. The *elevation axioms* act as a mapping between attributes in the source and semantics types in the *domain mode*l. Also, they codify the integrity constraints of the sources, which are useful for producing optimal queries. The *context axioms* define alternative interpretations of the semantics objects in different contexts. Every source is associated with exactly one context, but several sources may share the same context. These axioms are divided into two different groups: (1) axioms that define the semantics of data at the source in terms of values assigned to *modifiers*, which correspond to semantics objects, and (2) axioms that define how values of a given semantics object are transformed between different contexts *(conversion function*s).
  As we have described, COIN has a different representation of an ontology but it is based on an hybrid ontology approach. Therefore COIN can be compared with KRAFT or InfoSleuth, in which a change in one source will generate changes in some components to represent the new mappings. In COIN, elevation and context axioms have to be defined to access the new information.
- *Languag*e: The COIN framework is specified as a deductive and object-oriented data model of the family of F-logics [23].

Query:

- *User participatio*n: Users formulate a query based on an ontology without specifying a priori what information sources are relevant.
- *Query pla*n: The query plan involves the last three elements of the architecture: a *context mediator*, which rewrites the query as a *mediated query;* the *optimizer*, which transforms the mediated query as an optimized plan; and the *executioner* that executes it. The *context mediator* is in charge of the identification and resolution of potential semantics conflicts induced by a query. The automatic conflict detection and reconciliation are both possible by using general knowledge of the underlying application domain, along with the informational content and implicit assumptions associated with

the sources. These bodies of declarative knowledge are represented as elements of the framework *(domain mode*l, a set of *elevation axioms* and a set of *context axiom*s). The result of the mediation is a mediated query. To retrieve the data from the different information sources, the *optimizer* transforms the mediated query into a query execution plan, which is optimized taking into account the topology of the network of sources and their capabilities. Then the *executioner* executes the plan to retrieve the data from the various sources. Results are composed as a message and sent to the user. The mediation is performed by an abductive procedure, which produces a reformulation of the initial query in terms of the component sources. The procedure itself is inspired by the abductive logic programming framework [22].

- *Optimizatio*n: The application of abductive reasoning to query rewriting is a useful tool to achieve efficiency. It can be used to formally combine and to implement features of semantics query optimization (SQO) [11].

2.7. Proposal 1 by Arruda, Baptista and Lima [4]

This proposal presents an architecture and design of a web-based query system in which users, using an ontology, can specify their queries and submit them to the underlying data sources.

Architecture:

- *Information sources*: The approach proposes a mediator-based environment for data integration of structured and semi-structured data on the web. The heterogeneous data sources can be either structured data such relational or object databases, or semistructured data such HTML pages, text documents and XML [5]. The semistructured data are treated as XML documents and therefore it is required that the semistructured data repositories export their data into XML documents.
- *Architecture typ*e: The architecture is a mediator-based data integration system, which uses an ontology as its common schema. The main components of the architecture are: the *search engine*, the *mediator and query rewriting* and the *wrappers.* The first component is responsible for receiving the query from the user; for identifying the user chosen ontology; for iterating with the mediator and query rewriting component to structure and enrich semantically the query and for distributing the query among the wrappers (as in structured data) and calling appropriate XML query engines (as in semistructured data). The second component, the mediator and query rewriting, is responsible for maintaining the integrated schema view (the ontologies); for structuring and rewriting the queries according to the ontology and integrating all distributed results in XML format for a posterior document transformation using a publisher framework. Finally the *wrappers* are specific for each underlying structured source and translate the generic application queries into source specific queries. Also, they translate the results from these queries into XML format and send them back to the mediator module.

Semantic Heterogeneity:

- *Ontology us*e: This proposal uses XML documents as information sources. XML provides only a syntactical view of the underlying data. In order to allow for a precise querying and semantic interpretation, the XML documents should be complemented with a conceptual model that adequately describes the semantics of its tags. The proposal uses the Ontologies to fulfil this semantic gap enabling a precise semantic expression for querying XML documents. The ontologies are used as a common schema.

  In this proposal applies the same example as in the SIMS system, because it also define a global ontology (or global schema) containing information about the integrated sources.
- *Languag*e: The common schema is described in DAML+OIL [15] language and supports several ontologies, related or not. The DAML+OIL language is a semantic markup language for Web resources and is being developed as an extension to XML and the RDF [26].

Query:

- *User participatio*n: Users can access the system through a Web device (PDAs, mobile phones, browsers, etc), they browse it graphically and compose their queries using the specified ontology. The interface is developed for web browsers. A tree is automatically built from the DAML ontology elements. Also, the fields for the input query are built according to the ontology attributes for each element.
- *Query pla*n: The proposal has six steps in order to solve a query:

  (1) The users can access the system through a Web device browsing it graphically and compose their queries using the specified ontology.

(2) The mediator and query rewriting component receives the query and parses it in accordance to the conceptual ontological terms. The query then is rewritten using additional information of the ontology, keeping the semantics of the original query.
(3) The search engine then maps this query (which is generic a priori) into queries for both types of source: queries to semistructured data (XML documents) and to structured data (traditional databases). The query is transferred to the XML query engine, in the case of semistructured data, and in the case of structured data, the query is transferred to the wrapper of each underlying source.
(4) The XML Query Engine accesses all documents structured according to the user chosen ontology and related ones.
(5) Each wrapper maps the generic query into the appropriate query language of the local repository. Hence, it makes the due mapping between the internal data source schema and the external schema (based on the ontology). This mapping is done according to a previous conceptual correspondence between ontology and scheme. The results of these queries are then modelled into XML documents and delivered to the mediator and query rewriting component.
(6) The mediator module receives the results from the two kinds of sources in XML format, integrates them and calls an XML transformer (XSLT, for example) to the specific web device.

- *Optimization*: There is no mechanism to provide optimization.

2.8. Proposal 2 by Medcraft, Schiel and Baptista [29]

This proposal presents an architecture for semantic integration using mobile agents and a global ontology. The global ontology is used to represent the semantic information of the sources, and the mobile agents are used to process queries.

Architecture:

- *Information sources*: The objective of this proposal is to provide a uniform interface to heterogeneous, pre-existing relational databases, in a way that users can formulate queries (global queries) that are transported between these different databases by a mobile agent to produce a single consolidated response.
- *Architecture type*: The proposal has an agent-based architecture, known as DIA (Data Integration using Agents), for semantic integration of a federated database using mobile agents and ontologies. The architecture has the user interface developed with web browsers and the ontology includes the equivalent or similar concepts in the source databases. Also, the architecture proposes the use of a travelling pattern and a task pattern [25]. Travelling patterns deal with various aspects of managing the movement of mobile agents. These patterns allow us to enforce encapsulation of mobility management, which enhances reuse and simplifies mobile agent design. An extended *Itinerary* travelling pattern is implemented because it maintains a list of destinations, always knows where to go next and defines special cases such as what to do if a destination is temporally unavailable. To enable a host to be part of the mobile agent destination list, every destination needs to have an agency running a local stationary agent, which cooperates with the mobile agent in the local query adaptation process. Task patterns are concerned with the breakdown of tasks and how these tasks are delegated to one or more agents. The proposal also uses the *Master-Slave* pattern, which allows a master agent to delegate a task to a slave agent. The slave agent moves along its itinerary, performs the specified task and returns the result to its master agent.

Semantic Heterogeneity:

- *Ontology use*: The proposal identifies the objects in the databases that represent equivalent or similar concepts, that is, concepts semantically related. Heterogeneities between the databases appear as semantic conflicts. They are detected and solved at the moment a new database enters the federation. Once identified the equivalent or similar concepts in the new database, they are defined as global concepts in the existing ontology. This ontology describes the concepts as classes and properties, and defines the relationships between each other.

  In our example, Figure 2 represents the global ontology. When one change in the source is made, this proposal finds the equivalent concept of the global ontology, and modifies the matching table to store this new relationship accordingly. The global ontology proposed here represents the information about the domain. Each information source through the matching table maps its information with the information represented in the global ontology. Therefore, if added information (in this case about air-stewardess) is represented in the global ontology, only the matching table has to be modified.

- *Language*: The ontology is represented by the DAML+OIL [15] language.

Query:

- *User participation*: Users submit queries expressed over the ontology. The interface was developed for web browsers. A tree is automatically built from the DAML+OIL ontology classes and the fields for the input query are also built according to the ontology properties for each class.
- *Query plan*: Users trigger the query process by preparing a global query, based on terms from an ontology. When the user submits the query, the application notifies a stationary agent running inside an agency. This stationary agent (master agent) creates a mobile agent (slave agent) for the migration through the databases. Applying the Itinerary pattern explained in the architecture feature, before its creation, a list of destinations is set for the new mobile agent. With the query and knowing its itinerary, the migration process starts. As the mobile agent reaches a destination, it contacts the local stationary agent, passing it the global query. This agent knows the ontology on which the global query terms are based on, and also knows the local database schema. Therefore, with a local matching mechanism, it is able to convert the global query to a corresponding local one. The database query is performed and the result set presented to the visiting mobile agent is showed in XML syntax. In case the mobile agent is not at the first destination, before moving to the next node, the mobile agent performs an integration of the local result with the result collected previously. This reduces the size of the carried XML, and means that when the mobile agent returns to its origin, no extra result integration or analysis will be performed. When the mobile agent completes its itinerary, it migrates back to its origin, returns the final result to its master agent and removes itself. The master agent presents the collected data to the user.
- *Optimization*: The proposal realizes an optimized itinerary using two types of itineraries: *static* and *dynamic*. A static itinerary is an itinerary that can be established in advance by the stationary master agent. Depending on information available about the local databases, the master agent analyses the global query and selects those local databases that must be visited to process the query. In the simplest case, the itinerary is "visit all nodes". In a dynamic itinerary, the mobile agent can decide, depending on the partial result, if the query is already answered and it can return, or not. This dynamic capability depends on a previous classification of the global query.

2.9. Proposal 3 by Tzitzikas, Spyratos and Constantopoulos [36]

This proposal presents an approach for providing uniform access to multiple information sources through mediators and ontologies. It proposes a new component, named articulation, to improve the cost of adding or modifying the sources.

Architecture:

- *Information sources*: This approach proposes a model for providing integrated and unified access to multiple information sources of the type of Web catalogs. In the environment of the Web there are many examples of such sources. Specifically, the general purpose catalogs of the Web, such as Yahoo! or Open Directory (http://dmoz.org), the domain specific catalogs/gateways (e.g. for medicine, physics, tourism), as well as the personal bookmarks of the Web browsers can be considered as examples of such sources.
- *Architecture type*: The system has a mediator-based architecture. For each source are defined *ontologies* plus a *database* storing objects that are of interest to its users. Each object in the database of a source is indexed under one or more terms of the ontology of that source. Another architecture component is the *mediator*. The mediator is a secondary source that can bridge the heterogeneities that may exist between two or more sources and provides a unified access to those sources. Besides, the mediator has a number of *articulations* to the sources. An articulation to a source is a set of relationships between the terms of the mediator and the terms of that source. These relationships are defined by the designer of the mediator at design time and are stored at the mediator.

Semantic Heterogeneity:

- *Ontology use*: As we have said above each source has an *ontology*, that is, a structured set of names, or *terms*, that are familiar to the users of the source. In particular the ontologies that consider in this proposal consist of a set of terms structured by a subsumption relation. An ontology is a pair $(T, \preceq)$ where T is a *terminology*, i.e. a set of names, or *terms*, and $\preceq$ is a *subsumption* relation over T, i.e. a reflexive and transitive relation over T. In addition to its ontology, each source has a stored *interpretation I* of its terminology, i.e. a function $I : T \rightarrow 2^{Obj}$ that associates each term of T with a

set of objects. The symbol 2^{Obj} is to denote the power set of *Obj*. Also, the mediator has an ontology $(T, \preceq)$ that reflects the needs of its potential users but does not maintain a database of objects. Instead, the mediator has a number of *articulations* to the sources. The relationships included in the articulations are defined by the designer of the mediator at design time and are stored at the mediator. To solve heterogeneity problems among the sources, this proposal does not merge the ontologies, that is, if the same term appears within two sources, they do not assume as equivalent because they may have different interpretations (meanings). Two terms are considered equivalent only if they can be shown to be equivalent using the articulations of each source.

As the most appropriate approach to describe this proposal is the *hybrid ontology approach* [38], the modification or addition of a change in the sources only generates the modification of the local ontology (which represents this source) and the addition of the mappings between the ontology of the mediator and the local ontology within the respective articulation. If we consider that the ontology of the mediator is shown in Figure 5, then the mappings added in this example are the mappings that we have to add in the articulation.

- *Language*: To represent the ontologies there is no language proposed by this approach.

Query:

- *User participation*: Users submit queries expressed over the ontology of the mediator. Upon receiving a user query, the mediator has to query the underlying sources. A user who looks for objects of interest can browse the ontology of the source until he reaches the desired terms, or he can query the source by submitting a boolean expression of terms.
- *Query plan*: Users formulate queries over the ontology of the mediator and it is the task of the mediator to choose the sources to be queried, and the query to be sent to each source. Then it is again the mediator that appropriately combines the results returned by the sources in order to produce the final answer. Specifically, the mediator uses the articulations in order to translate queries over its own ontology to queries over the ontologies of the articulated sources. The sources can provide two types of answers to a given query, namely, a *sure* answer or a *possible* answer (the first type of answer being appropriate for users that focus on precision, while the second for users that focus on recall). Moreover a user query to the mediator admits two types of translation, namely, *lower* or *upper* translation (again, the first type of translation being appropriate for users that focus on precision, while the second for users that focus on recall). The combination of these two types of answer and of translation generates four interpretations: lower-sure, lower-possible, upper-sure and upper-possible. Therefore, the mediator can answer queries submitted by its users based on any of these four interpretations. Also, the operation modes of the mediator either can be decided (and fixed) by the mediator designer at design time, or they may be indicated by the mediator users at query time.
- *Optimization*: There is no mechanism to provide optimization.

3. Discussion

Table 3, and Figure 7 and 8 present the results of the systems evaluation for each feature, i.e. architecture, semantic heterogeneity and query resolution; using the judgment scale explained in Section 2.

Table 3 shows the resultant assessment of the Architecture feature. It has simple features assessed by a simple YES/NO scale. As we can see in the first column, SIS (dynamic information), some systems are assessed by a NO value, that is, these systems do not have implemented mechanisms to know which information is available at a given moment. The second column, TIS, assesses the types of information systems supported by the systems. All systems, except the Proposal 3, support the integration of databases in the federation. But only some of them support semistructured data such as HTML pages and files. The last column, Architecture Type, does not have an assessment because more information is necessary in order to evaluate the architecture properties. We divide the architecture into two main approaches: agent-based approaches and mediator-based approaches and classify the systems into these approaches. For example, the KRAFT system uses an agent-based architecture to provide extensibility and adaptability in a dynamic distributed environment. KRAFT also focuses on the interchange of data and constraints among agents in the system. Another example is the Proposal 2 working on a mobile agent architecture to improve the process of retrieving and consolidating data from the heterogeneous databases. Otherwise, a system like COIN uses a mediator-based architecture to detect semantic conflicts among heterogeneous systems by comparing context axioms corresponding to the sources involved.

	Architecture				
	Information Sources				
	SIS	TIS			
	Dynamic Information	Databases	HTML pages	Files	Architecture Type
SIMS	Yes	Yes	No	No	mediator-based
OBSERVER	Yes	Yes	Yes	Yes	wrappers, ontology server and IRM
DOME	No	Yes	No	No	wrappers, ontology server and mapping server
KRAFT	Yes	Yes	No	No	agent-based
Carnot	No	Yes	No	No	agent-based
InfoSleuth	Yes	Yes	Yes	Yes	agent-based
COIN	Yes	Yes	Yes	Yes	mediator-based
Proposal 1	No	Yes	Yes	Yes	mediator-based
Proposal 2	Yes	Yes	No	No	agent-based
Proposal 3	No	No	Yes	No	mediator-based

Table 3. The results for Architecture feature

Figure 7 shows the results of assessing the ten systems for the semantic heterogeneity feature by using a graphical representation. All of the sub-features are compound features and they are assessed by the judgment scale for conformance defined in Table 2.

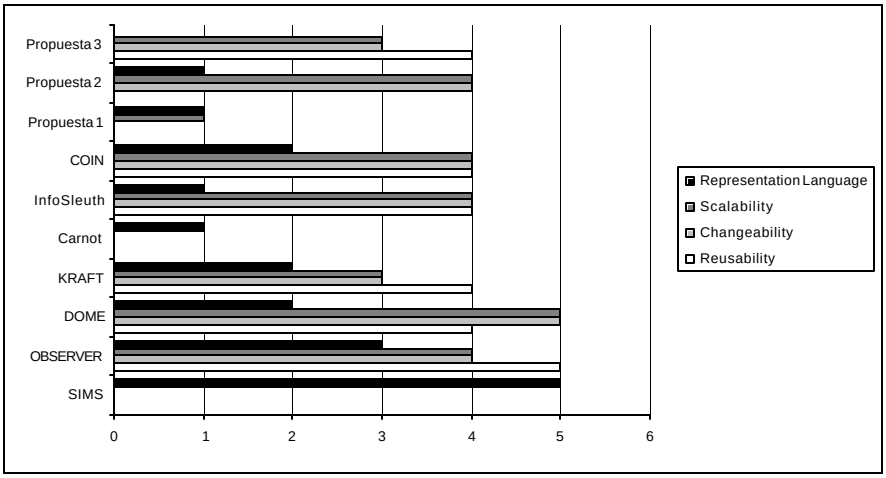

Figure 7. A graphical representation for Semantic heterogeneity feature

We will first analyze the ontology use feature (reusability, changeability, scalability). As we can see SIMS and Carnot systems have a zero (0) in the three sub-features. In the first sub-feature the zero means that both ontologies are not reusable because both define a global ontology or a single ontology approach in order to integrate the data. The same happens with the changeability sub-feature, because no support is given by these systems to bear changes in the information sources. The global ontology must be rebuilt when one source changes. Finally, scalability, is not supported because again by adding a new information source the rebuilding of the global ontology is generated. The problem of dealing with the global ontology approach is that we must manage a global integrated ontology, which involves administration, maintenance, consistency and efficiency problems that are very hard to solve. For example, SIMS requires constructing a general domain model that encompasses the relevant parts of the database schemas. Due to each database model is related to this general domain model, the integration problem is shifted from how to build a single integrated model to how to map the domain and the information source models. Also, the Proposal 1 has a similar assessment because a global ontology is built and changes in one source or the addition of a new source generates a modification in this global ontology.

Otherwise, the OBSERVER system, for example, uses multiple ontologies (with different vocabularies) to solve the users needs in a better way, and to alleviate the problems presented by the single approach. In order

to do so, OBSERVER implements an IRM component that enhances the scalability of the query processing strategy by avoiding the need of designing a common global ontology. The reusability is also full supported by OBSERVER because it defines local ontologies that might be defined for other purposes.

KRAFT has the first sub-feature, reusability, scored with a very strong support (4) because it defines an hybrid ontology approach where the local ontologies can be defined independently. The changeability is assessed as a strong support because when a source changes, only the local ontologies and the mapping ontology must be modified by including the changed information. The scalability is also strongly supported because we only include the information of the new source in the same components. Something similar happens with Proposal 3, COIN and InfoSleuth because in the case of Proposal 3 to add a source we have to select a mediator in the network and design an articulation between the selected mediator and the new source. In the case of COIN the domain model does not have to be updated each time a new source is added. Finally, in the InfoSleuth system to add a resource to the system the resource only needs to have an interface to advertise its services and let other agents make use of it immediately.

The two last sub-features in the DOME system have the higher value, i.e. full support(5) because DOME researchers are developing tools in order to extract ontologies from legacy systems, check the ontology consistency, merge ontologies, etc. All these tools generate a highly scalable system.

Otherwise, Proposal 2 has no support for reusability because a global ontology is defined. But the changeability and scalability have very high values because this global ontology does not need to be modified when a source is added, only an ontology-schema matching table is necessary to add a new database to the integrated system.

The support given by the systems to apply some specified language is assessed by the representation language sub-feature. Only SIMS has a full support because it is the only one that has a full description about the language used in the ontology (Loom). Otherwise, OBSERVER does not have a predefined language, that is, the different ontologies (user ontologies) can be described by using different vocabularies depending on the users needs although it shows several examples and uses of CLASSIC language describing how the system work. The other systems have little or no support for the languages. Some of them only refer to the used language but no help or explanation is provided.

Figure 8 shows the results of assessing the ten systems for the query resolution feature by using a graphical representation. All of the sub-features are compound features and they are assessed by the judgment scale for conformance defined in Table 2.

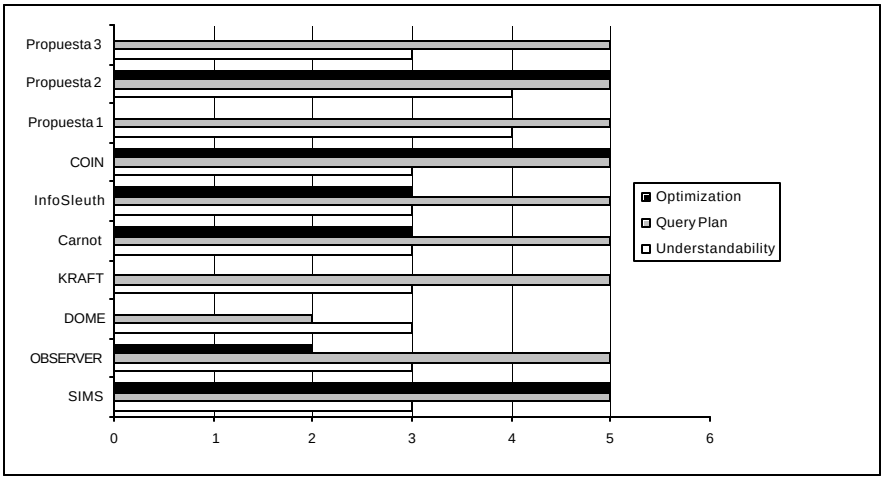

Figure 8. A graphical representation for Query resolution feature

The user participation sub-feature evaluates the degree of understandability that the system provides. In all the systems, users perform a query based on ontologies because of their better understandability. Therefore, as we can see, the first seven systems and the last one have a strong support on this feature. But Proposal 1 and Proposal 2 have a very strong support because the user interface used to make queries is shown and described more clearly. Also, a clear and understandable answer is given by all of them.

The second sub-feature, query plan, evaluates whether the systems clearly describe a set of steps to explain the query process. In general, the query is divided into subqueries in such a way that each of them corresponds to a different information source.

All systems, except DOME have a full description about the query plan describing each step and how the components interact in order to get a consistent answer. The DOME system does not describe the steps to achieve the query, only explain the components of the architecture. It does not show how the components interact in a query resolution.

Regarding to optimization, the last sub-feature, some systems (DOME, KRAFT, Proposal 3 and Proposal 1) do not have any mechanism in order to optimize the query. In the case of OBSERVER, it does not have any mechanism to optimize the query, but uses the concept *loss of information* to describe the situation in which a user wants to expand the query to other ontologies obtaining more relevant data. Indirectly, this characteristic would help the optimization of the query. Other systems like InfoSleuth and Carnot have a similar optimization strategy that ORION system applies. The full support values are chosen for the SIMS, COIN and Proposal 2 systems because all of them have implemented specific mechanisms to optimize the query. For example, SIMS and COIN use SQO to achieve it.

Finally, we should analyze the score sheets and determine which of the systems is the best. Our analysis is based on accumulating the absolute scores. As we have simple features as well as compound ones we need assign a score to the YES and a score to the NO. It is inappropriate to score 1 for providing a simple feature because that will bias your assessment towards systems that provide some support for compound features. Therefore, we assess the YES simple feature with a score 5, and the NO simple feature with a score 0. Besides, we use the following weights to assess the relative importance of features:

- Mandatory: 10
- Highly Desirable: 6
- Desirable: 3

Each feature will be assessed by the product between the assessed value on each feature and the specified weight depending on its importance. For example, in the case of SIMS for *Information Sources* feature, the resultant value is 100 ((5+5)x10, because it is a mandatory feature) out of a possible 200 giving a percentage score of 50%. Another example can be COIN, with a resultant value of 200 (100%) obtained by the product between 20 (5x4, because the four subfeatures are scored with 5) and 10.

Figure 9 graphically shows the percentages for each system within the Information Sources feature. The highest values (100%) are obtained by COIN, InfoSleuth and OBSERVER systems because they fulfil all the subfeatures within the Information Sources feature. The lowest values are achieved by Carnot, Proposal 3 and DOME systems.

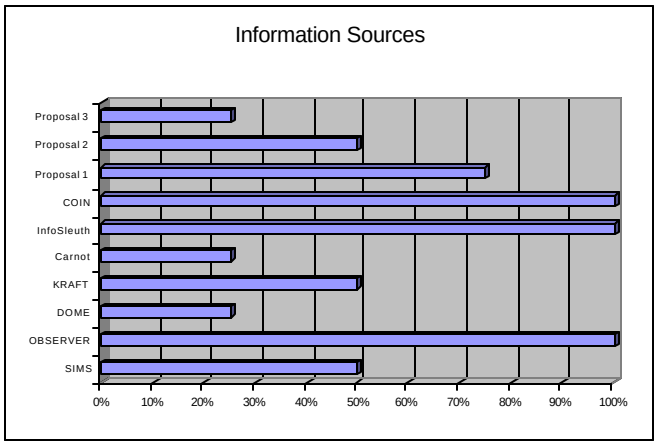

Figure 9. The information sources percentages

Similarly, Figure 10 presents the percentages of the Semantic Heterogeneity feature.

DOME and OBSERVER systems have the highest values while Carnot, Proposal 1 and SIMS systems obtain the lowest values.

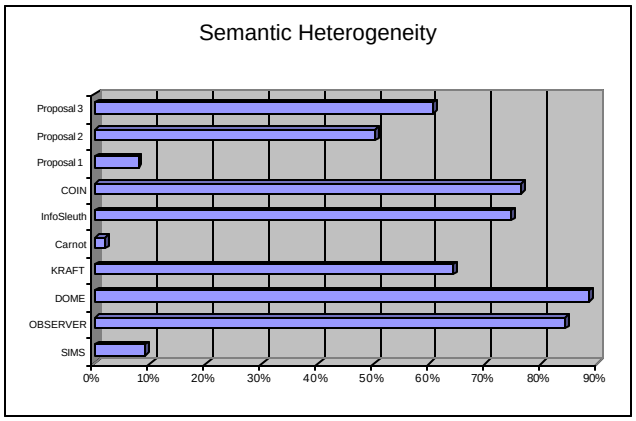

Figure 10. The semantic heterogeneity percentages

Finally, Figure 11 shows the percentages of the Query Resolution feature. All systems have high values but Proposal 2 and SIMS systems are the best.

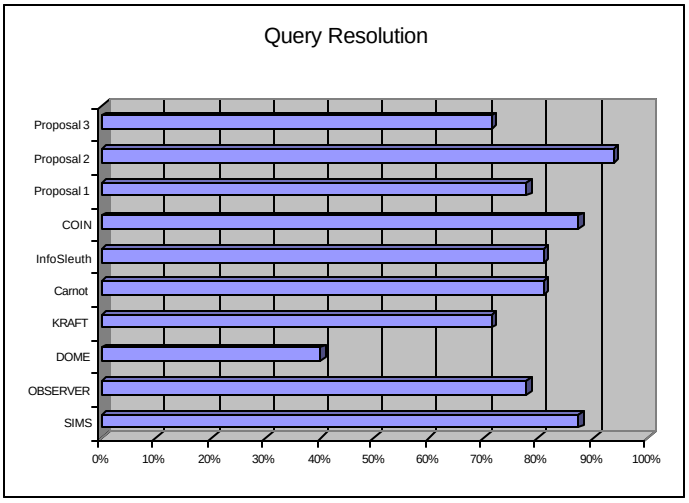

Figure 11. The query resolution percentages

With all of these percentages in mind, we can do the final comparison. Figure 12 presents the total values obtained by the product between the addition of all the assessed features and their respective weights (accordingly with their importance). The higher value allowed joining all features, being 460, that is the 100%.

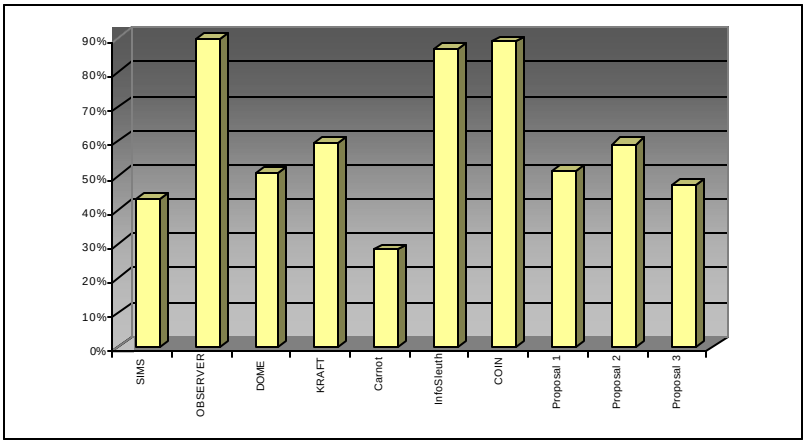

Figure 12. The final percentages

OBSERVER, InfoSleuth and COIN systems have obtained the highest values, between 80-90%, because they have achieved higher scores in many features or subfeatures of the framework. The second range of scores, between 50-60%, has been obtained by Proposal 2, Proposal 1, KRAFT and DOME systems. Finally, Proposal 3, SIMS and Carnot systems have the lower values because some features or subfeatures are not supported.

4. Conclusion

In this paper, we have presented both an analysis and a comparison of seven systems and three proposals that use ontologies to solve the problems involved in data integration. In order to do so, we have created a conceptual framework with three main categories: *architecture*, *semantics heterogeneity* and *query resolution*. Based upon our comparison, we have found some elements in common and also original aspects of the systems.

In order to assess the features defined in our framework the DESMET approach for Feature Analysis has been used. This approach assesses the features with a scale of values according to the mechanism used by the systems to resolve them. OBSERVER, InfoSleuth and COIN systems have obtained the higher values because they have totally achieved many features of the framework.

Our survey intents help for the ontology-based data integration community giving and comparing different aspects of systems which have been used as a reference for further research. But other several aspects have to be analyzed as the valuation of the architecture properties. Also for each approach, a more exhaustive analysis is needed to compare the optimization techniques applied to the query plans.

Some of these systems are still under development, and currently the research community is looking for ways of improving the construction of their ontologies. Therefore, there is still an ongoing concern on how ontology-based data integration should be. We hope our comparison be useful to the increasingly community working today on data integration using ontologies.

References

1. Ambite, J.L., Arens, Y., Ashish, N., Knoblock, C. A. and collaborators. The SIMS Manual 2.0. Technical Report, *University of Southern, California.* December 22, 1997. http://www.isi.edu/sims/papers/sims-manual.ps.
2. Arens, Y., Hsu, C., Knoblock, C. A. Query processing in the SIMS Information Mediator. *Advanced Planning Technology, Austin Tate (Ed.), AAAI Press*, Menlo Park, CA, 61-69, 1996.
3. Arens, Y., Hsu, C., Knoblock, C. A. Retrieving and integrating data from multiple information sources. *International Journal in Intelligent and Cooperative Information Systems*,Vol.2(2), 127-158, 1993.
4. Arruda, L., Baptista, C., Lima, C. MEDIWEB: A Mediator-based environment for data integration on the web. *Databases and Information Systems Integration. ICEIS* , 34-41, 2002.
5. Baru C. Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. In: *W3CWorkshop on Query Language(QL´98)*, Boston, 1998.
6. Bass, L., Clements, P., Kazmar, R. *Software Architecture in Practice.* Addison-Wesley. 1998.
7. Bassiliades, N. and Gray, P. M. D. CoLan: A functional constraint language and its implementation, Data and Knowledge Engineering, 14, 203–249, 1994.
8. Bayardo Jr., R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A. and Woelk, D. InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. Microelectronics and Computer Technology Corporation (MCC).
9. Borgida, A., Brachman, R.J., McGuinness, D.L., and Resnick, L.A. CLASSIC: A structural data model for objects. In Proceedings *ACM SIGMOD-89*, Portland, Oregon, 1989.
10. Busse, S., Kutsche, R. D., Leser, U. & Weber H. (1999). Federated Information Systems: Concepts, Terminology and Architectures. Technical Report. Nr. 99-9, *TU Berlin*.
11. Chakravarthy, U. S., Grant, J., and Minker, J. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.* Vol.15(2), 162–207, 1990.
12. Corcho, O. and Gomez-Perez, A. Evaluating knowledge representation and reasoning capabilities of ontology specification languages. *In Proceedings of the ECAI 2000 Workshop on Applications of Ontologies and Problem-Solving Methods*, Berlin, 2000.
13. Cui, Z., Jones, D. and O'Brien, P. Issues in Ontology-based Information Integration. *IJCAI 2001*– Seattle, USA • August 5 2001.
14. Cui, Z. and O'Brien, P. Domain Ontology Management Environment. *In Proceedings of the 33rd Hawaii International Conference on System Sciences*. 2000.

15. DAML+OIL 2001: Available at:http://www.daml.org/2001/03/daml+oil-index.
16. Deschaine, L., M., Brice, R., S., Nodine, M., H. Use of InfoSleuth to Coordinate Information Acquisition, Tracking and Analysis in Complex Applications. Technical Report *MCC-INSL-008-00*. 2000.
17. Firat, A., Madnick, S., Grosof, B. Knowledge integration to overcome ontological heterogeneity: challenges from financial information systems. *Twenty-Third International Conference on Information Systems*. 2002.
18. Goh, C.H., Bressan, S., Siegel, M. and Madnick, S. E. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems*, Vol. 17(3), 270–293, 1999.
19. Gray, P.M.D, Preece, A., Fiddian, N.J. and colab. KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases, *Proceedings of 8th International Workshop on database and Expert Systems Applications* (DEXA'97). 1997.
20. Gruber, T. A translation approach to portable ontology specifications. *Knowledge Acquisition 1993* –Vol.5(2), 199–220, 1993.
21. Hsu, C. and Knoblock, C. A. *Using Inductive learning to generate rules for semantic query optimization*. In Piatetsky-Shapiro, G., Fayyad, U., Symyth, P. and Uthurusamy, R. editors, Advances in Knowledge Discovery and Data Mining, chapter 17. AAAI Press, 1996.
22. Kakas, A. C., Kowalski, R. A., and Toni, F. .Abductive Logic Programming,. *Journal of Logic and Computation*. Vol.2(6), 719-770, 1993.
23. Kifer, M., Lausen, G., and Wu, J. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, Vol.42(4), 741–843, 1995.
24. Kitchenham, B. DESMET: A method for evaluating Software Engineering methods and tools Technical Report TR96-09. *Department of Computer Science, University of Keele, Staffordshire.* August 1996.
25. Lange, D., Oshima, M. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.
26. Lassila O., R.R. Swick,. Resource Description Framework (RDF) Modeland Syntax Specification. *W3C Recommendation*, February 22, 1999.
27. MacGregor, R. Inside the LOOM classifier. *SIGART bulletin*. Nº 2(3), 70-76, 1991.
28. MCC Carnot Project, http://www.mcc.com/projects/carnot
29. Medcraft, P., Schiel, U., Baptista, P. DIA: Data Integration Using Agents. Databases and Information Systems Integration. *ICEIS*, 79-86, 2003.
30. Mena, E., Kashyap, V., Sheth, A. and Illarramendi, A. Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Kluwer Academic Publishers*, Boston. http://citeseer.nj.nec.com/mena96observer.html, 1-49, 2000.
31. Mena, E., Kashyap, V., Sheth, A. and Illarramendi, A. Managing Multiple Information Sources through Ontologies: Relationship between Vocabulary Heterogeneity and Loss of Information. *In Proceedings of Knowledge Representation Meets Databases (KRDB'96)*, ECAI'96 conference, Budapest, Hungary, 50-52, 1996.
32. Nam, Y. & Wang, A. (2002). Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. In Proceedings *International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems*, Irvine CA, 28-30.
33. Preece, A., Hui, K., Gray, P. KRAFT: Supporting Virtual Organisations through Knowledge Fusion. Artificial Intelligence for Electronic Commerce: *Papers from the AAAI-99 Workshop*, Technical Report WS-99-01, AAAI Press, 33-38, 1999.
34. Preece, A., Hui, K., Gray, A., Jones, D., Cui, Z. The KRAFT Architecture for Knowledge Fusion and Transformation. In *19th SGES International Conference on Knowledgebased Systems and Applied Artificial Intelligence (ES'99)*, Berlin. Springer.
35. Siegel, M. and Madnick, S. E. A metadata approach to resolving semantic conflicts. In Proceedings of the 17th Conference on Very Large Data Bases (Barcelona, Spain, Sept.). *VLDB Endowment 1991*, Berkeley, CA, 133–145, 1991.
36. Tzitzikas, Y., Spyratos, N., Constantopoulos, P. Mediators over Ontology-based Information Sources. *Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'02)*.IEEE 2002.
37. Visser, P. R. S., Jones, D. M., Bench-Capon, T. J. M. and Shave, M. J. R. Assessing heterogeneity by classifying ontology mismatches. *In Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, IOS Press, 148–162, 1998.
38. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. and Hübner, S. Ontology-based Integration of Information - A Survey of Existing Approaches. *In: Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing*, Seattle, WA, 108-117, 2001.
39. Woelk, D., P. Cannata, M. Huhns, W. Shen, and C. Tomlinson. Using Carnot for Enterprise Information Integration. *Second International Conf. Parallel and Distributed Information Systems*. January, 133-136, 1993.
40. Woelk, P., Kim, W. And Lee, W. Query Processing in Distributed ORION. International Conference on Extending Datadase Technology, 169-187, March, 1990.

41. Woelk, D. and C. Tomlinson. The InfoSleuth Project: Intelligent Search Management via Semantic Agents. *MCC Tech. Report*, Sept., 1994. Also available in the Electronic Proceedings of the Second World Wide Web Conference '94 at http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/woelk/woelk.html.
42. Yang,H. Cui, Z. and OBrien, P., Extracting Ontologies from Legacy Systems for Understanding and Re-engineering. *IEEE International Conference on Computer Software and Applications*, 1999.
43. Zaniolo, C. The Logical Data Language (LDL): An Integrated Approach to Logic and Databases. *MCC Technical Report STP-LD-328-91*, 1991.