J Intell Inf Syst (2006) 26: 7–24
DOI 10.1007/s10844-006-5448-9

Building and querying e-catalog networks using P2P and data summarisation techniques

Hye-young Paik · Noureddine Mouaddib · Boualem Benatallah · Farouk Toumani · Mahbub Hassan

Abstract One of the critical issues in Web-based e-commerce has been how to efficiently and effectively integrate and query heterogeneous, diverse e-catalogs.[1] We propose an integration framework for building and querying catalogs. Our approach is based on a hybrid of peer-to-peer data sharing paradigm and Web-services architecture. Peers in our system serve as domain-specific data integration mediators. Links between peers are established based on the similarity of the domain they represent. The relationships are used for routing queries among peers. As the number of catalogs involved grow larger, the need for filtering irrelevant data sources will become increasingly high. We apply a summarisation technique to summarise the content of catalogs. The summaries are used to pre-selecting data sources that are relevant to a user query.

Keywords E-catalogs · Data integrations · Peer-to-peer · Data summarisation

H.-y. Paik
School of Information Systems, Queensland University of Technology, Brisbane, Australia
e-mail: h.paik@qut.edu.au

N. Mouaddib
LINA/ATLAS-GRIM, Polytech'Nantes, University of Nantes, France
e-mail: mouaddib@lina.univ-nantes.fr

B. Benatallah · M. Hassan
School of Computer Science, University of New South Wales, Sydney, Australia
e-mail: boualem@cse.unsw.edu.au, mahbub@cse.unsw.edu.au

F. Toumani
LIMOS, ISIMA, University Blaise Pascal, France
e-mail: ftoumani@isima.fr

[1] We use terms e-catalog and catalog interchangeably.

1. Introduction

Catalog portals, such as Amazon.com, are becoming a more and more prominent feature of the World Wide Web. Behind every portal, there exists a potentially large number of intricate product catalogs which have to be *integrated and queried*. In most web catalog portals, (i) integration relies on centralised categorisation and indexing of all products, which is not scalable, (ii) querying is often based on information retrieval techniques, with no or limited semantic support, where a keyword is searched against entire catalog content, (iii) the number of data sources to be considered are potentially large, hence there is a need for an approach that would effectively identify a pocket of closely relevant sources for a given query to improve query efficiency.

Consider the following user query: "*Show me a list of laptops, made by Sony, with at least 250 MB memory, 40 GB HDD and more than 1 year warranty*". The typical keyword-based matching used by catalogs is not suited for processing this kind of query. This is because a portal covers various kinds of products, it is almost impossible to build an *advanced query user interface* for every kind of products the portal supports. That is, the query attributes for finding books are different from those for finding DVDs. Inherently, search is limited only to general attributes like "title". The centre of this problem is that the integration is centralised and based on a single categorisation and indexing scheme. To cater for highly dynamic and distributed nature of e-catalogs (e.g., catalogs are added or removed frequently), we adopt divide-and-conquer strategy in designing a framework for building dynamic portals.

In our approach, catalogs catering for similar customer needs are grouped together into *catalog communities* (Bouguettaya et al., 2000; Paik et al., 2004) (eg., community of laptops, books or travels). Catalogs *register* themselves to a community as *members*. A catalog community can be viewed as an ontology of the catalogs belonging to the community members (ie., an integrated schema specific to a product domain). Catalog communities themselves can be linked together as *peers* based on ontology relationships (eg., community of flights can peer with community of accommodations).

Queries are initially processed by a single community, but if the community cannot satisfy all the requirements specified in a query, it looks for other peers who can fulfill the requirements. This process is known as a *collaborative query processing* in our system. The purpose of the collaboration between peers is to identify a set of members that, when put together, can satisfy all constraints required by the query. Such members can be both local or external to the community who first received the query.

It is noted that we described the framework that builds and queries catalog communities in details in Benatallah et al. (2005). This paper focuses on how querying large number of catalog communities can be managed efficiently by employing a summarisation technique.

As the number of members grow large in a catalog community, the number of data sources to be considered in a query processing becomes high. This calls for an approach that could effectively identify a subset of the members that is most likely to provide relevant answers to a given query. The idea is to not only select e-catalogs based on their query capabilities but also based on how likely their content is relevant to answer the given query. We use a content summarisation technique to build a *local summary* of each catalog in a community. The local summaries are integrated to build a community summary which is used to perform content-based selection of catalogs that are most likely relevant to a user query.

In this paper, we present how we construct the integrated summaries of catalog communities and make use of them in selecting relevant data source for a given query. To be self-contained, we briefly describe the framework, the WS-CatalogNet system here as presented in Benatallah et al. (2005). WS-CatalogNet has been designed and implemented in a

web-services framework. Both the catalogs and the communities, and summarisation technique are implemented as web services.

The paper proceeds as follows. In Section 2, we describe how catalog communities are built (i.e., how communities are created, members are registered, peers are linked, etc.), Section 3 describes our collaborative query processing techniques, where peers work together to find catalogs that are relevant to a given query. Then, in Section 4, we introduce the content summarisation technique. Section 5 describes the implementation architecture of the system. Section 6 discusses some related work, followed by concluding remarks in Section 7.

2. Building catalog communities

A community describes its ontology in terms of categories and descriptive attributes. For example, the community `Laptops` may have a category `Laptop`, which is described using attributes such as `Brand`, `CPU`, `Memory`, `HDD`, and `Price` etc. (see Figure 1).

2.1. Community description language

To provide formal semantics in describing the ontology, necessary for precise characterisation of queries over the catalogs, we use a (concept) class description language that belongs to the family of description logics (Baader et al., 2003). The community ontology (also called *community schema*) is described in terms of *classes* (unary predicates) and *attributes* (binary predicates). Class descriptions are denoted by expressions formed by the following constructors:

- class conjunction ($\sqcap$), e.g., the description `Peripherals` $\sqcap$ `SCSI` denotes the class of products which are instances of the classes `Peripherals` and `SCSI` (e.g., a `Hard-Disk`),
- the universal attribute quantification ($\forall R.C$), for example, the description $\forall$`release Date.Date` denotes the class of products for which all the values of the attribute `releaseDate` are instances of the class `Date` (i.e., the data type of the attribute `releaseDate` is `Date`),
- the existential attribute quantification ($\exists R$), e.g., the description $\exists$`Memory` denotes the class of products having at least one value for the attribute `Memory`.

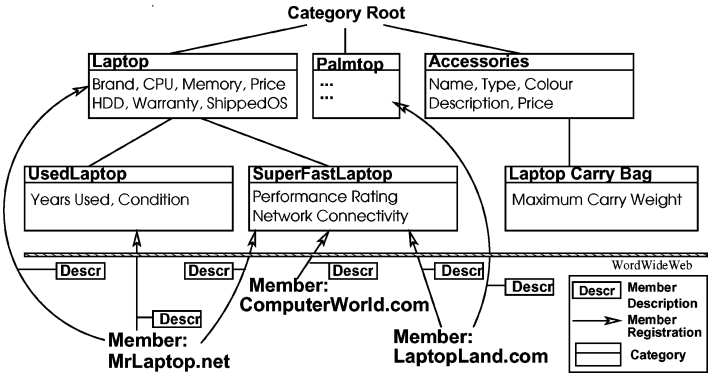


Fig. 1 Community of Laptops, its categories and members

*Definition 1 (*syntax of class descriptions). Let $\mathcal{CN}$ be a set of class names and $\mathcal{A}$ be a set of attribute names. Class descriptions are inductively defined as follows:

- C is a class description for each class name $C \in \mathcal{CN}$.
- Let C, D be class descriptions and $R \in \mathcal{A}$ an attribute name. Then $C \sqcap D$, $\forall R.C$ and $\exists R$ are class descriptions as well.

A model-theoretic semantics for this language is given by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. It consists of a nonempty set $\Delta^{\mathcal{I}}$ the domain of the interpretation, and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function is associated with each class name $C \in \mathcal{CN}$ a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and with each attribute name $R \in \mathcal{R}$ a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Additionally, the extension of $\cdot^{\mathcal{I}}$ to arbitrary class descriptions has to satisfy the following equations: $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$. $(\exists R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \exists y : (x, y) \in R^{\mathcal{I}}\}$.

Based on this semantics, the notions of subsumption and equivalence between class descriptions are defined as follows. Let C and D be class descriptions:

- C is *subsumed by* D (noted $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretation $\mathcal{I}$.
- C is *equivalent to* D (noted $C \equiv D$) if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all interpretation $\mathcal{I}$.

2.2. Creating a community

To create a new community, WS-CatalogNet provides: (i) *community definition editor (a graphical user interface)*, (ii) *description logic converter* and (iii) *web service wrapper*. With the community definition editor, a community administrator creates the community ontology. S/he first defines the name of the community and the root category. The sub categories are then added and attributes are defined for each category. Note that a sub category inherits all attributes of its parent category. As part of the community ontology, each category (respectively for each attribute) is annotated with a list of synonyms. For example, for the attribute `CPU`, synonyms may be {`processor`, `chipset`, `chip`}.[2] After the initial ontology is defined in the editor, the description logic converter automatically generates the class descriptions of the given ontology. For example, the category `UsedLaptop` in figure 1 may be described as follows:

$$\texttt{UsedLaptop} \equiv \texttt{Laptop} \sqcap \forall \texttt{YearUsed.Number} \sqcap \exists \texttt{YearUsed} \sqcap \forall \texttt{Condition.String} \sqcap \exists \texttt{Condition}$$

It states that the category `UsedLaptop` inherits all the attributes of the category `Laptop`, and has two additional attributes, namely `YearUsed` and `Condition`.

Being a web service, a community in WS-CatalogNet implements a standard set of operations which can be invoked by the user or the peers (e.g., *addCategory()*, *addPeer()*, *queryCommunity()* etc.). The final step of community creation involves generating the WSDL which shows details of the standard operations (e.g., signature of the operations) and the community ontology (i.e., categories and attributes). The newly created community is deployed and registered to the private UDDI hosted by WS-CatalogNet.

[2] The editor is integrated with WordNet WordNet which is used to automatically suggest synonyms for categories and attributes.

2.3. Registering catalogs

A potential members are of two kinds: (i) catalog provider who already has his catalog accessible via a web service, (ii) catalog provider who needs to create a web service which accesses his catalog. For the latter, WS-CatalogNet provides a catalog provider with functionality which is similar to the one used in creating communities. Using the provided functionality, the catalog provider can describe the catalog's ontology (in terms of categories and attributes), generate WSDL for the catalog (e.g., signatures of the standard set of operations for members), deploy and register it with UDDI as a new web service.

The catalog providers (who now have their catalogs running as web services) search UDDI to discover communities of interests and join them via the *registration* process. A catalog provider can join or leave a community of interest at any time. When registering, the member first indicates which categories, in the community, the member's catalog belongs to. For each category selected, the member specifies what kind of attributes are supported for the category (called *member definition*). The member definition from a member are also converted to the class descriptions and added to the community's ontology. For example, the provider called `mrlaptop.net`, who offers a range of used laptops, can register with the community `Laptops` using the following member definition:

$$\texttt{mrlaptop.net\_UsedLaptop} \equiv \texttt{Laptop} \sqcap \forall\, \texttt{YearUsed.Number} \sqcap \exists \texttt{YearUsed}$$

The above states that the member supports all attributes in the category `Laptop` in addition to the attribute `YearUsed` of the category `UsedLaptop`. A a member can provide several definitions.

2.4. Peering catalog communities

The community administrators search UDDI and discover other communities to form a peer relationship. An administrator of a community C_i may decide to form a peer link with another community C_j when C_i has categories that are considered analogous, or interchangeable to $\texttt{C}_\texttt{j}$'s categories. (e.g., category `CD-RW Drives` in community `SonyRetailers` and category `CD-RW/DVD` in community `BestCDRom`).

The terms used in categories and attributes may be different from one community to another. This differences needs to be resolved for the communities to collaborate with queries (i.e. community `SonyRetaliers` needs to know how a query described in its own ontology should be translated into the peer `BestCDRom`). Let us consider that the administrator of a community (noted `Source`) forms a relationship with another community (noted `Target`) (assume that `Target` is discovered through searching UDDI). At this point, the administrator is presented with the details of categories and attributes of the two communities. The administrator then defines a mapping description which states how the categories (respectively the attributes) in `Source` are mapped to `Target`. For flexibility, we consider three types of mapping description: the administrator makes (i) explicit and complete mapping (*full mapping*), (ii) explicit categories only mapping (*partial mapping*, no mapping for attributes), (iii) no explicit mapping (*no mapping*). This mapping information is stored in `Source`. When the translation of a query between peers is needed, and explicit mapping description is not available, the `Source` and `Target` uses synonyms (see Section 2.2).[3]

[3] WordNet is used to assist the administrator in the mapping process by deriving lexical relationships between categories/attributes of *Source* and *Target*.

3. Collaborative query processing

We now describe how the peers collaboratively participate in a query processing. The query interface in WS-CatalogNet lets the user easily formulate a query (by point&click). A query is expressed in terms of categories and attributes. That is, the user will click a category and then select attributes to be queried on, and specify values for the attributes if desired (e.g., "*category:laptops, attributes: memory, brand, HDD, warranty, price, values: warranty* $\geq$ *1, price* $\leq$ *2000*"). The query interface automatically converts the query to a class description.

Since a community does not store product data locally, processing the query requires locating catalogs that are capable of answering the query. These catalogs can be selected from the local members of the community or the members of the peers. We propose a collaborative query processing technique that consists of two steps: (i) identify combination of members whose query capabilities, when put together, satisfy all constraints expressed in the query, (ii) resolve the query by sending it to the selected members.

For the first step, we adopted a query rewriting algorithm—Best Quality Rewriting (BQR),[4] which takes as input the community definition, member definitions and the query Q (all in their class descriptions format) then produces the following output:

(a) Q_{local}: the part of the query Q that *can be answered* by the community's local members, that is, the attributes specified in the query that are supported by the local members. It also gives the combination of the local members that can answer all (or part of) the query. For each selected local member, we compute the part of the query to be sent to it.

(c) Q_{rest}: the part of the query that *cannot be answered* by the local members. The community will collaborate with peers to identify any external members who can answer this part of the query. Hence, Q_{rest} is forwarded to peers. The expected answers of the forwarding is the combination of the external members that are capable of answering Q_{rest}.

Each community has a *collaboration policy* which controls what should be done with Q_{rest}. The collaboration policy can express[5] (i) which part of the query should be forwarded, (ii) when the query should be forwarded (e.g., when no local members can answer, when the community is too busy etc.) (iii) to which peer (e.g., all, top K, random, etc.,) the query should be forwarded and (iv) how far the query should be forwarded (hop limit).

The community collects the returned results from the peers and chooses the best members (local and external) out of the given combination, based on the quality the members (e.g., reliability) and user preferences. After all members are selected, each of the member (both local and external) processes parts of the query they can process, and the results are returned to the community. The community is responsible for performing the join operation on the collected results before displaying them to the user.[6]

Figure 2 shows the query interface which displays the categories and attributes of community `FlightCenter`. The user formulates a query mainly by point&click. Let us say the query Q is "**category***:InternationalFlights,* **attributes***: fromCity = Sydney, toCity = Paris, price* $\leq$ *1000, travelInsurance = full* ". The `QueryProcessor` module generates corresponding class description of the query. Then it runs the BQR algorithm to compute Q_{local} and Q_{rest}. Let us assume that the local member `STAFlightCenter` is selected as a relevant catalog to

[4] The readers are referred to Benatallah05IS for details this algorithm.

[5] The policy is written in a simple attribute-value pair fashion.

[6] It should be noted that, although important, the issue of assembling actual results returned by selected catalogs, is outside the scope for this paper.

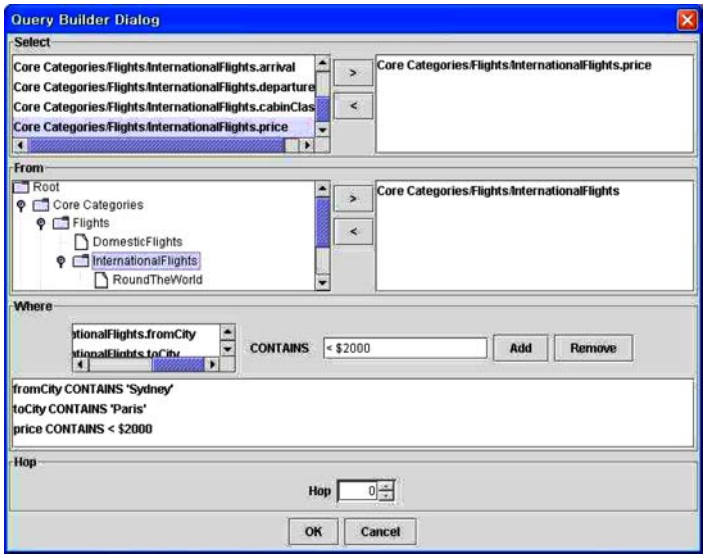


Fig. 2 Expressing a query on the `FlightCenter` community

answer the user query because it provides the attributes: `fromCity`, `toCity` and `price` (i.e., Q_{local}). Then, Q_{rest} now contains the attribute `travelInsurance`.

Q_{rest} is forwarded to the peers of `FlightCenter` according to its collaboration policy (eg., it may state that Q_{rest} should be forwarded to all of its peers with hop limit of 3). Assume that Q_{rest} (`travelInsurance`) is forwarded to communities `TravelInfo`, `WebJetDeal`. After forwarding, both communities return `SmileTravel` and `Best Travel` respectively, as the members who can answer `travelInsurance`. `Flight`

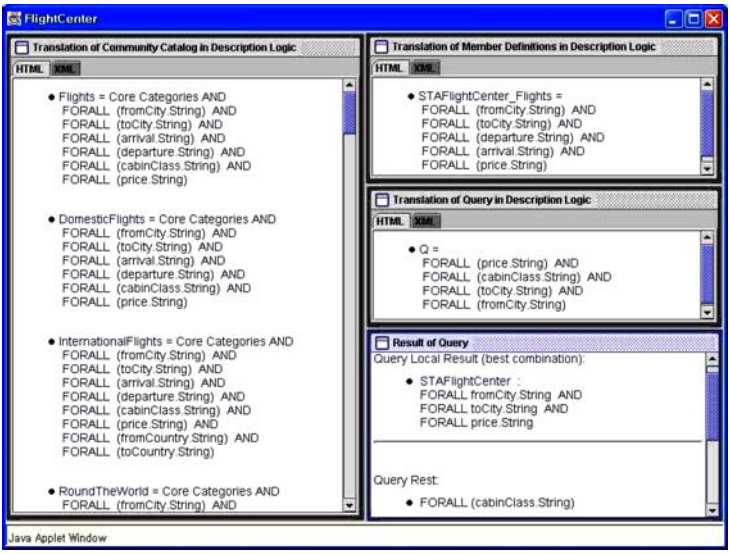


Fig. 3 Querying `FlightCenter` using BQR, showing Q_{local} and Q_{rest}

`Center` decides to combine the local member `STAFlightCenter` and the external member `SmileTravel` (referred by `TravelInfo`) and send the query to them. Figure 3 shows the first result of running BQR.

4. Selection of relevant e-catalogs using content summaries

As the number of e-catalogs of communities grow larger, processing queries may become costly. Two main factors impact the efficiency of our query processing approach: (i) complexity of query rewriting at the first step of our query processing approach,[7] and (ii) the network communication cost in the second step of the query processing, due to the execution of queries in a distributed environment.

In this section, we extend our previous work using a data summarisation and classification technique that can improve the query efficiency. The main idea is to build and integrate the members' *content summaries* into the community. When processing a query, the summaries are used in a pre-processing step to *filter* the members that are likely to provide a relevant answer to the query. The summary extracted from a single member's catalog content is called *linguistic summary*. As an example, it is of the form: *Most flights in the member* `STAFlightCenter` *are from Sydney to Places in China*. Assume that this information is stored in the community `FlightCenter`. When processing a user query that asks for an international flight from Sydney to Paris, `FlightCenter` can ignore `STAFlightCenter` as its content is irrelevant to the answer. This example highlights two main advantages of the pre-processing step: (i) it reduces the number of member definitions to be considered by the query rewriting algorithm, and (ii) avoid sending queries over the network to the members that will return empty answers.

In order to implement the pre-processing step, each community needs to store all of its members' content summaries, hereafter called *local summaries*, and also to maintain a global view of such summaries, called *community summary*.

Let us now define more precisely the notion of *linguistic summaries*. Let Q be a quantity in agreement (e.g., the linguistic quantifier `most`), R be a collection of data and S is the summariser, i.e., a summary of a set of values (e.g., *cheap*). A *linguistic summary* is an expression of the form: 'Q *objects* in R are S'. For example, *'Most flights in the catalog A are cheap'* is a linguistic summary.

In the sequel, we give a general overview of the summarisation technique and then we describe a summarisation approach that enable to build local and community summaries. More details about the summarisation technique can be found in Raschia (2002) and Voglozin et al. (2004).[8]

4.1. An overview of the summarisation technique

By analysing the content of a member catalog (e.g., stored as a set of relational tuples or XML elements), the summarisation produces a hierarchically arranged set of linguistic summary tuples. An overview of the summarisation technique is described at Figure 4. The summarisation technique relies on the following two steps:

[7] In Benatallah05IS, we showed that the query rewriting algorithm is exponential to the number of member definitions to be considered.

[8] see also http://www.simulation.fr/seq.

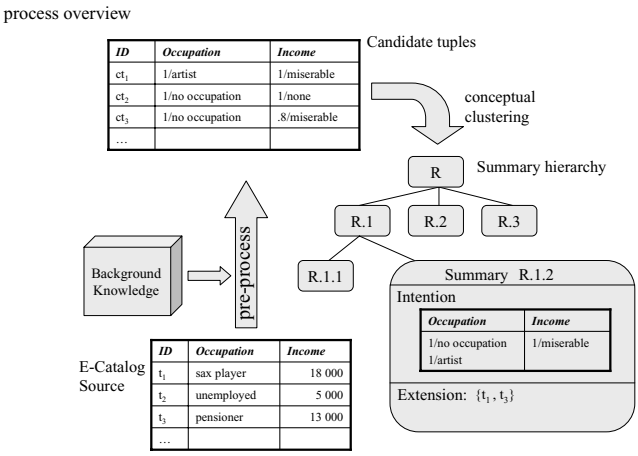


Fig. 4 The overall process of summarisation

The pre-processing step: the raw data in a catalog are first prepared for summarisation. Using *Background Knowledge* (BK) supplied by a community designer, this step computes a linguistic representation of catalog tuples. As explained latter, the BK contains information such as attribute transformation values, domain-specific linguistic descriptors to classify catalog tuples, etc. At the end of this step, all attribute values in catalog tuples are translated into a finite set of *linguistic descriptors*. For example, as depicted in Figure 4, the values of the attributes {ID, Occupation, Income} of a given e-catalog are translated into more general linguistic descriptors. This allows to categorise the corresponding tuples according to the community BK. For example, value 18000 in the attribute `Income` is now described as *"(1/miserable)"*. This descriptor indicates that the corresponding catalog tuple belongs to an income category characterised by the linguistic term "`miserable`" and it is estimated with full certitude (i.e, degree of certitude, also called *degree of truth*, equals to 1) . The value translation is based on a fuzzy set (Zadeh, 1965) membership function as explained later.

The summarisation step: a summary hierarchy is constructed incrementally using a concept formation algorithm, accepting one source tuple at a time. New data are incorporated into a concept hierarchy using a local optimisation criteria to decide how should the hierarchy be modified Fisher (1987). A quality measure is evaluated to compare the effect of operators that modify the hierarchy topology, namely, creating a new node, creating a new level, merging two nodes, or splitting one. Using fuzzy logic (Zadeh, 1965) in the evaluation of this measure, our concept formation algorithm is less prone to suffer the well known threshold effect of similar incremental algorithm.

A degree of truth τ is associated with each summary linguistic descriptor, providing an indication of how "close" this descriptor is to a given catalog tuple. This measure of truth τ of a summary linguistic descriptor is performed as follows:

1. for each tuple r in a catalog R, we compute the degree, noted $S(t)$, to which t satisfies the summariser S;
2. let $r = 1/n \sum_{i=1}^{n} S(t)$, the proportion of tuples in R that satisfies S, then $\tau = Q(r)$, denotes the degree of the membership of r in the proposed quantity agreement.

Table 1 An airline catalog $\mathcal{R}$

From	To	Price (in AUD)
Sydney	Paris	2000
Sydney	Lyon	2000
Brisbane	Paris	1800
Canberra	Grenoble	1700

Consider a catalog $\mathcal{R}$ (Table 1) that stores information about flights. Formally, let $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ be the set of attributes of $\mathcal{R}$. For instance the attributes are `From`, `To`, and `Price`. The domain of attributes, denoted by D_A, could be an interval $[0, 500]$ for the attribute `Price`, or a set of terms {Sydney, Brisbane, Canberra} for the attributes `From` and `To`. An element $t \in \mathcal{R}$ is represented by a vector $< t.A_1, \ldots, t.A_n >$, where attribute values $t.A_i$, $A_i \in \mathcal{A}$, are basically crisp. For example, `t1.To` = `Paris`. The translation, however, should be able to handle approximate and vague information, hence the summarisation approach uses fuzzy set theory instead of crisp values.[9]

Consider the crisp set Ω as the universe of discourse. $\mathcal{F}(\Omega)$ is the power fuzzy set of Ω, ie., the set of fuzzy sets of Ω. An element $F \in \mathcal{F}(\Omega)$ is defined by the membership function:

$$\mu_F : \Omega \to [0, 1],$$
$$x \mapsto \mu_F(x).$$

In the following, the membership degree of x in F is indistinctly denoted by $F(x)$ or $\mu_F(x)$. In the case where $\Omega = \{x_1, \ldots, x_n\}$, $n \in N$, the membership function of F is denoted by $\mu_F = \{x_1/\alpha_1 + \cdots + x_n/\alpha_n\}$, where $F(x_i) = \alpha_i$, $i \in [1, n]$. For example, the membership function that determines a fuzzy set of `NSW` (New South Wales) is $\mu_{NSW} = \{$Sydney/1, Bankstown/1, $\ldots\}$.

4.2. Local summary and community summary

We describe now how local and community summaries are built using the summarisation technique presented in the previous section. Figure 5 gives the general architecture of our summarisation approach.

First, local summaries are generated from the content of individual e-catalogs. Then, a community summary is computed by integrating local summaries. As mentioned before, local summarisation uses a Background Knowledge Base (BK) that contains a shared definitions (eg., vocabularies for product categories and attributes, membership functions, etc.). Note that the KB is shared by all members in a community. For example, to summarise data tuples such as `From` and `To` locations in a flight catalog, the knowledge base would have the knowledge describing `NSW` as a generic abstraction of `Sydney` or `Australia` as an abstraction of `NSW`, etc.

*Example 1 (*Local summary). Let us assume that an airline catalog stores flights information as a set of tuples described using the following attributes (`From`, `To`, `Price`). Let us also assume the airline catalog contains following set of tuples {`D1` = (`Sydney`, `Paris`, `1010`), `D2` = (`Sydney`, `Paris`, `900`), `D3` = (`Bankstown`, `Paris`, `1100`)}. Then, an example of a local

[9] L.A. Zadeh, Fuzzy sets, Information and Control 8 (1965) pages 338–353.

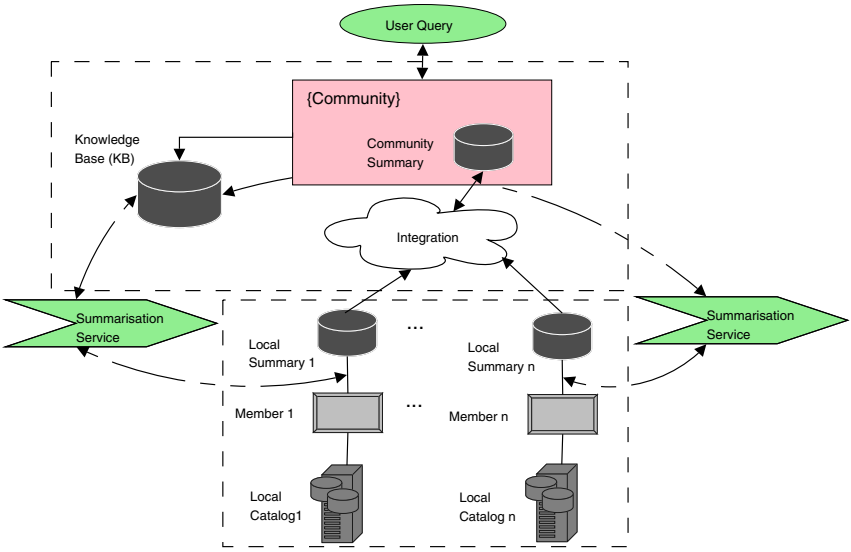
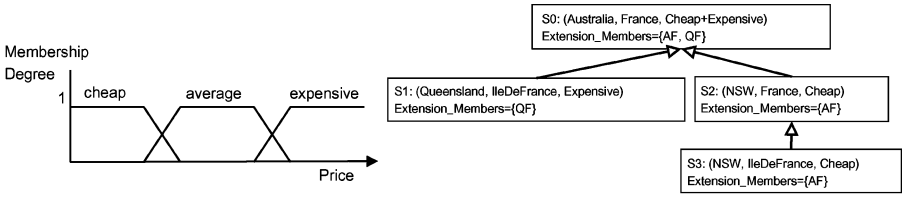


Fig. 5 General architecture of the summarisation



(a) An example of a characteristic function (b) Obtaining a Community Summary from Local Summaries

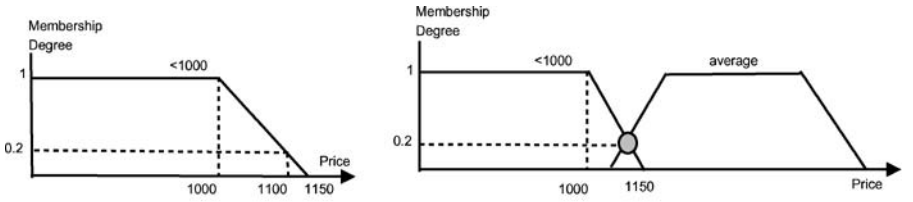
Fig. 6 A characteristic function, local/community summaries

summary of this catalog is represented by the summary tuple `S1` = (`NSW`, `Ile De France`, `Cheap`)[10] where:

- `NSW`, `IleDeFrance` are linguistic values which are defined by *membership functions* of the community BK. As an example, a membership function that defines the fuzzy set `NSW` is $\mu_{\text{NSW}} = \{\text{Sydney}/1, \text{Bankstown}/1, \ldots\}$, $\mu_{\text{IleDeFrance}} = \{\text{Paris}/1, \ldots\}$. Here, the value $(Sydney/1)$ means that $Sydney$ belongs to the *fuzzy set* `NSW` with a membership degree 1.
- *Cheap* is also a linguistic value defined by a membership function μ_{Cheap}. The membership function μ_{Cheap} is described in Figure 6(a).

Once catalog members of a given community are summarised, the obtained *local summaries* are then integrated into a *community summary*, which is used by a community for selecting relevant catalogs when processing queries. The integration process of local summaries into a *community summary* is based on the *summarisation* step as defined in Section 4.1. Indeed, we sort the local summaries by the number of their nodes, and we choose the one that has the

[10] Note that, a summary is in fact a hierarchy of summary tuples.



(a) Characteristic Function (b) Similarity degree between two characteristic functions

Fig. 7 A characteristic function, similarity between two characteristic functions

greater number of nodes as the initial hierarchy of *community summary*. Next, we sequentially classify nodes of local summaries in the *community summary* in using the *summarisation* step.

*Example 2 (*Community summary). Let us consider the following three local summaries:

- `S1` = (NSW, IleDeFrance, Cheap), a local summary of Air France (`AF`).
- `S2` = (`NSW`, `France`, `Cheap`), a generalisation of `S1`
- `S3` = (`Queensland`, `IledeFrance`, `Expensive`), a local summary of Qantas (`QF`).

Using the integration process, the local summaries are hierarchically structured to form a community summary of a flights community, as depicted in Figure 6(b).

*Example 3 (*Using the Summaries for Filtering). Continuing with the previous example, assume the following query: *List all flights from Sydney to Paris with prices less than 1000*. The community can use the global summary to identify `AF` as a relevant catalog, because its summary suggests that it provides flights from *NSW* to *Ile de France* with prices in the cheap category, effectively eliminating `QF`.

4.3. Filtering irrelevant catalogs using the summaries

The aim of the filtering process is to select, from the community summary, the nodes which give the *best matching* with a user query, hence identifying the relevant members. The matching between a user query Q and an element S (i.e. node of the hierarchy) of the Community Summary is calculated with a similarity measure *Similarity* which is defined below.

For the matching, we reformulate the query values using the membership functions, for example (price < 1000), the value < 1000 will be expressed using the membership function $\mu_{<1000}$ as defined in Figure 7(a):

The similarity measure is then calculated as follows:

$$Similarity(Q, S) = Min_{i=1,n}(Sim(Q.A_i, S.A_i))$$

where n is the number of attributes in Q and Sim is a similarity degree between Q and S on the attribute A_i. Let $\mu_{Q,i}$ and $\mu_{S,i}$ be the characteristic functions that represent the $Q.A_i$, $S.A_i$ values. The Min operator is used as an example of aggregation operator.

$$Sim(Q.A_i, S.A_i) = Sup_{x \in D_i} Min(\mu_{Q,i}(x), \mu_{S,i}(x))$$

where D_i is the definition domain of the A_i attribute.

*Example 4 (*Measuring Similarity). Consider the three following summaries that constitute the community summary of the FlightCenter community :

- S1 = (NSW, Ile de France, Cheap) (members in S1 = {AF, CA})
- S2 = (NSW, Ile de France, Average) (members in S2 = {AF})
- S3 = (NSW, Ile de France, Expensive) (members in S3 = {QF, BA})

Now, let Q be a user query *List all flights from Sydney to Paris with prices less than 1000*. Suppose that the Knowledge Base of the community contains the following definitions:

- NSW $\rightarrow$ μ_{NSW} = {Sydney/1, Canbera/1 ...}. The element Sydney/1 means that the membership degree of the value Sydney to the characteristic function μ_{NSW} is equal to 1. Note that $\mu_{\text{NSW}}(\text{Sydney}) = 1$.
- IleDeFrance $\rightarrow$ $\mu_{\text{IleDeFrance}}$ = {Paris/1, ...}
- $\mu_{\text{Sydney}(x)} = 1$, if x = Sydney, and 0 else.
- $\mu_{\text{Paris}(x)} = 1$, if x = Paris, and 0 else.
- μ_{Cheap}, μ_{Average} and $\mu_{\text{Expensive}}$ are defined as in Figure 6.

Therefore, the similarities between Q and the summaries are :

- Similarity (Q, S1) = Min (Sim (Q.From, S1.From), Sim (Q.To, S1.To), Sim (Q.Price, S1.Price))
  - Sim (Q.From, S1.From) = $Sup_{x \in \text{Dcity}}$ Min($\mu_{\text{Sydney}}(x)$, $\mu_{\text{NSW}}(x)$) $= \mu_{\text{NSW}}(Sydney) = 1$
  - Sim (Q.To, S1.To) = $Sup_{x \in \text{Dcity}}$ Min ($\mu_{\text{Paris}}(x)$, $\mu_{\text{IleDeFrance}}(x)$) $= \mu_{\text{IleDeFrance}}(\text{Paris}) = 1$
  - Sim (Q.Price, S1.Price) = $Sup_{x \in \text{Dcity}}$ Min ($\mu_{<1000}(x)$, $\mu_{\text{Cheap}}(x)$) $= 1$

    There exists a set of values for which $\mu_{<1000}$ and μ_{Cheap} are equal to 1. For example, $\mu_{<1000}(500) = \mu_{\text{Cheap}}(500) = 1$, so Min($\mu_{<1000}(x)$, $\mu_{\text{Cheap}}(x)$) = 1 and Max (1, ?) = 1.

  $\rightarrow$ Similarity(Q, S1) = 1, thus, we identify the members `AF` and `CA` as relevant members with degree 1.
- Similarity (Q,S2) = Min(Sim (Q.From, S2.From), Sim (Q.To, S2.To), Sim (Q.Price, S2.Price))
  - Sim (Q.From, S2.From) = Sim (Q.To, S2.To) = 1
  - Sim (Q.Price, S2.Price) = $Sup_{x \in \text{DPrice}}$ Min ($\mu_{<1000}(x)$, $\mu_{\text{Average}}(x)$) $= 0.2$ (see Figure 7(b).)

  $\rightarrow$ Similarity (Q, S2) = 0.2, and thus, we identify the member `AF` as a relevant member with degree 0.2.
- Similarity (Q, S3) = Min (Sim (Q.From, S3.From), Sim (Q.To, S3.To), Sim (Q.Price, S3.Price))
  - Sim (Q.From, S3.From) = Sim (Q.To, S3.To) = 1
  - Sim (Q.Price, S3.Price) = $Sup_{x \in \text{DPrice}}$ Min ($\mu_{<1000}(x)$, $\mu_{\text{Expensive}}(x)$) = 0

  $\rightarrow$ Similarity (Q, S3) = 0, and thus, the members `QF` and `BA` are not relevant.

Table 2 Operations in tModel for product catalog members

Operations for Member M and description
`String Query(String query): query M`
`HashMap GetGeneralInfo(): get metadata about M`
`HashMap GetAttributes(): get product attributes of M`

Note that for the relevant members we keep the best degree of relevance, for example for `AF` member we keep the degree 1 (i.e. Max (1, 0, 2)).

5. *WS-CatalogNet* implementation

WS-CatalogNet is a web service based environment for building catalog communities. It consists of a set of integrated tools that allow catalog providers to create communities, member relationships (between a catalog provider and community) and peer relationships (between communities). It also allows users to access the communities and send queries to them.

In *WS-CatalogNet*, both product catalogs and communities are represented as web services Curbera et al. (2002). The UDDI (Universal Description, Discovery and Integration) registry is used as a repository for storing web services' information. In UDDI registry, every web service is assigned to a tModel. The tModel provides a classification of a service's functionality and a formal description of its interfaces. We design specific tModels for product catalogs (see Table 2) and for communities (see Table 3). These two specific tModels are WSDL document types of tModel.

Table 3 operations in tModel for catalog communities

Operations for community C and description
`String Query(String queryS): query C with queryS`
`HashMap GetGeneralInfo(): get metadata of C`
`HashMap GetAttributes(): get product attributes of C`
`String ForwardedQuery(String originC,int hopLeft, String query): process forwarded query from originC to C`
`Int Register(St ring mbName,HashMap generalInfo,HashMap attribute Mapping): register member mbName to C`
`Int Deregister(String mbName): deregister member mbName from C`
`Vector GetMemberNames(): get member names registered to C`
`ECatalogInfo GetMemberInfo(String mbName): get member's details (general information, product attributes and the attribute mappings between the member and community C)`
`Int AddPeer(String peerName, HashMap,generalInfo, HashMap attributes, HashMap attributeMapping) : add a peer community`
`Int RemovePeer(String peerName): remove a peer`
`Vector GetPeerNames(): get peer names of C`
`ECatalogInfo GetPeerInfo(String peerName): get peer community's details.`

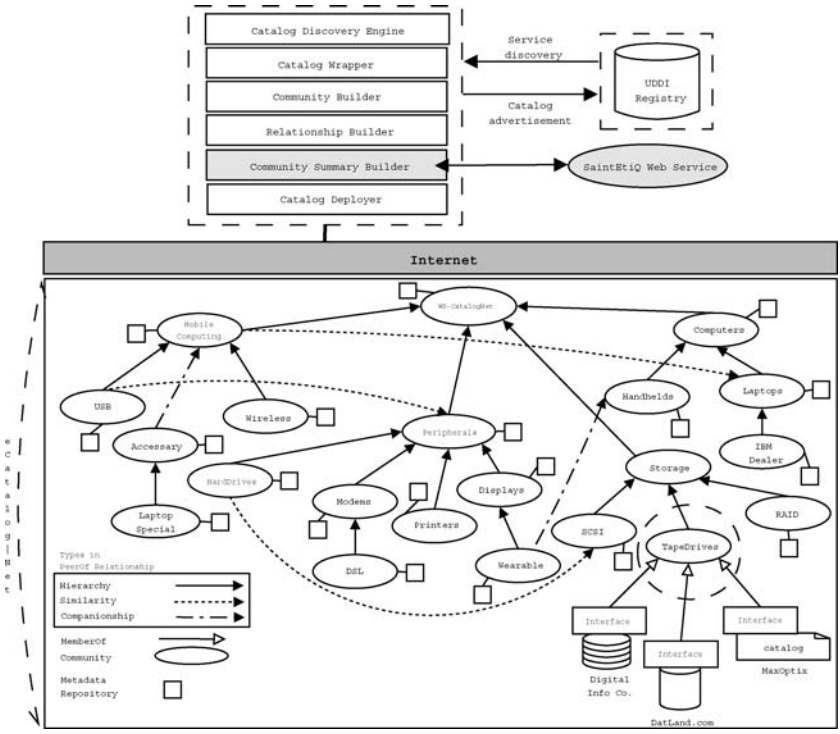


Fig. 8 *WS-CatalogNet* architecture

Overall, the prototype has been implemented using Java. The discovery engine is implemented using the IBM Web Services Development Kit 5.0 (WSDK). WSDK provides several components and tools for Web service development (e.g., UDDI, WSDL, and SOAP). In particular, we used the UDDI Java API (UDDI4J) to access a private UDDI registry (i.e. hosted by the *WS-CatalogNet* platform), as well as the WSDL generation tool for creating the WSDL documents and SOAP service descriptors required by the catalog discovery engine. *WS-CatalogNet* is composed of the following modules: *catalog discovery engine, catalog builder, community builder, relationship builder*, and *catalog deployer* (see Figure 8).

Catalog Discovery Engine. It facilitates the registration and location of product catalogs and catalog communities. When a catalog (respectively community) registers with a discovery engine, a UDDI SOAP request is sent to the UDDI registry. After the registration, the catalog (respectively community) can be located by sending the UDDI SOAP request (e.g., service name) to the UDDI registry.

Catalog Wrapper. It assists catalog providers to create a web service by wrapping the existing catalog. It provides a step-by-step process to create new catalog web service, starting from the creation of WSDL document to the registration of the WSDL to UDDI registry.

Community Builder. It assists catalog providers in the creation and maintenance of catalog communities. Like the catalog builder, it provides a step-by-step process to create new catalog communities, starting from the creation of WSDL document to registration of WSDL to UDDI registry. The community provider has to download a special class named `QueryProcessor` which is provided by our system. The class provides methods for

processing query requests for each catalog community. It relies on metadata about members and peers (i.e., intra and inter-community indices). The metadata is generated during the creation of membership or peer relationship, and stored as XML documents. This class is lightweight and the only infrastructures that it requires are standard Java libraries, a JAXP-compliant XML parser, and a SOAP server. In the current implementation, Oracle's XML parser 2.0 and IBM's Apache Axis 1.0 are used.

Relationship Builder. It allows to create relationships between the communities (respectively between a community and its members). The relationship builder provides functionality to discover catalog communities (respectively product catalogs) stored in UDDI registry. Once a community (respectively a catalog) is found, the relationship builder allows the community administrator to add/remove peer link between two communities (respectively, add/remove member from a community). It also updates the metadata documents about intra, inter-community indices whenever a change occurs (e.g., a member is added, a peer is removed, etc.)

Summary Builder. It allows the catalog providers to build the local summaries and the community administrator to construct the community summary. Local summaries are constructed using a summarisation service called SaintEtiQservices [11].

Catalog Deployer. The catalog deployer is responsible for generating a *service implementation binding directory*. The web service implementation-binding directory contains all the necessary files for web services to be deployed in WSDK server. The directory also includes the implementation files for the new service. WSDL2WebService is a tool in WSDK which generates a fully deployable *service implementation-binding directory* from a single WSDL document. One of the files generated by the tool is a template for the actual implementation of the new service. The template has default implementation of the operations in the service. In the case of product catalogs, It is up to the catalog providers to provide the actual implementation and link it to the web service.

6. Related work

We identify two major areas to discuss related work, namely product catalog integration and peer-to-peer systems. Most existing work in integration of online catalogs use a schema integration approach. For example, Yan et al. (2002) uses synonym set which contains a list of synonyms for source catalog schema attributes. Using 'intersection' of the synonym sets, an integrated global schema is derived. Navathe et al. (2001) suggests a logical integration model, in which all categories of products are organised in a single graph structure and each leaf links to source catalog's product attribute tree which represent local catalog's product classification scheme. However, construction of product attribute tree for each product of a source catalog is not a trivial exercise. Fensel et al. (2001) considers the product catalog integration as content management issue. They use information extraction techniques to derive a structured, integrated schema from a set of *documents* containing unstructured product description. Most approaches result in static integration which can not easily scale up (e.g., to accommodate new catalogs). The specific issue of integrating large number of dynamic catalogs is not considered.

[11] http://www.simulation.fr/seq.

The area of Peer-to-Peer systems has attracted many researchers over the past few years. The first breed of systems such as Gnutella, Napster and other systems alike has focused on sharing files (e.g., music, video clips). These systems only support limited set of metadata and offer limited querying functionality (such as keyword based search, IR-style content matching). Such limitation may be acceptable for simple file sharing purposes, but there also is need for structured querying to exploit the inherent semantics in data Berstein et al. (2002); Papadimos et al. (2003). Our work focuses on the similar issue, therefore, we concentrate the discussion mainly on current efforts that leverage database query paradigm in P2P environment. Most work in this area deal with the semantic interoperability between peers. For example, Ng et al. (2003) uses metadata (schema description and keywords) which is associated with every relation, and candidate peers for query forwarding are selected based on the comparison result of metadata. Kementisietsidis et al. (2003) proposes use of mapping tables which map data *values* between two peers. Such mapping tables are constructed by domain experts. These approach assume no knowledge about underlying schema of the peers when forwarding a query, but bear efforts of users to identify the correct mappings.

In Aberer et al. (2003), new schema mappings are learned from existing mappings by a way of transivity in a *mapping graph* which is formed under an assumption that a group of peers that may have agreed on common semantics. Some approaches use data integration techniques. Halevy et al. (2003) propose schema mediation language in peer-to-peer setting. Each peer contains storage descriptions which specify what kind of data is stored by relating its relation to one or more peer relations. It uses two main techniques (local-as-view, global-as-view) used for schema mediation in data integration systems. Also, Berstein et al. (2002) uses the Local Relational Model (LRM) to translate general queries to the local queries with respect to the schema of the peer. Both work assume all peers employ relational data model.

Papadimos et al. (2003) propose a framework for distributed query processing through mutant query plans (MQP) and multi-hierarchy namespaces. A query plan includes verbatim XML encoded data, references to actual resource locations (URL) and references abstract resource names (URN). A peer can mutate an incoming query plan by either resolving URN to URL, or substituting a sub-plan with the evaluated XML encoded data. The authors of Nejdl et al. (2003) have proposed super-peer based routing[12] in RDF-based peer-to-peer setting. We use similar concept to construct routing indices.

Our work is unique from the other related approaches in the following aspects: (i) peers are used as a small scale, domain specific data integration medium, (ii) we consider types of the relationships between peers. This is intended to give flexibility to communities when they establish peer relationships. Such flexibility means a peer can decide what kind of interaction it wants with the others. The metadata for routing queries also reflect the flexibility in peer relationships, (iii) we cater for situations where there is be no (or partly complete) description of mappings, but peers still can forward queries.

7. Conclusion

We have proposed a scalable integration framework which can deal with potentially large number of product catalogs. The framework follows peer-to-peer paradigm, but unlike simple file sharing systems, each peer has capability to serve semantically rich queries and such queries can be expanded by forwarding to other peers regardless of the level of knowledge

[12] The idea of super-peer is also discussed in YG03.

about the schema of other peers. We also addressed the need for eliminating irrelevant data sources before a start of the query processing to maintain or improve the efficiency as the number of data sources grow larger.

References

Aberer, K., Cudre-Mauroux, P., & Hauswirth, M. (2003). The chatty web: emergent semantics through gossiping. In *Proceedings of WWW'03*, Budapest.

Bouguettaya, A., Benatallah, B., Hendra, L., Ouzzani, M., & Beard, J. (2000). Supporting dynamic interactions among web-based information sources. *IEEE Trans. on Knowledge and Data Eng.*, *12*(5), 779–801.

Baader, F., Calvanese, D., & McGuinness, D. (2003). *The description logic handbook*. Cambridge University Press.

Berstein, P., Giunchigiloa, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., & Zaihrayeu, I., (2002). Data management for peer-to-peer computing: a vision. In *Proceedings of WebDB'02*, (pp. 89–94). Madison. Wisconsin.

Benatallah, B., Hacid, M-S., Paik, H-Y., Rey, C., & Toumani, Farouk (2005). Towards semantic-driven, flexible and scalable framework for peering and querying e-catalog communities. *Information Systems, (*to appear*)*.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing, 6*(2), 86–93.

Fensel, D., Ding, Y., & Omelayenko, B. (2001). Product data integration in b2b e-commerce. *IEEE Intelligent Systems,* Vol. *16*(4), 54–59.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2, 139–172.

Halevy, A., Ives, Z., Suciu, D., & Tatarinov, I. (March 2003). Schema Mediation in Peer Data Management Systems. In *Proceedings of 19th International Conf. on Data Engineering*, Bangalore, India, March 2003.

Kementisietsidis, A., Arenas, M., & Miller, R.J., (June 2003). Mapping data in peer-to-peer systems: semantics and algorithmic issues. In *Proceedings of ACM SIGMOD 2003*, San Diego, CA, June 2003.

Ng, W.S., Ooi, B.C., Tan, K.L., & Zhou, A. (2003). PeerDB: A P2P-based System for Distributed Data Sharing. In *Proceedings of the 19th International Conf. on Data Engineering*, (pp. 633–644). Bangalore, India, March 2003.

Navathe, S., Thomas, H., A., M. Satits, & Datta, A. (2001). A Model to Support E-Catalog Integration. In *Proceedings of the IFIP Conf. on Database Semantics*, Hong Kong, April 2001. Kluwer Academic Publisher.

Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., & Lser, A. (2003). Super-Peer-Based routing and clustering strategies for rdf- based peer-to-peer networks. In *Proceedings of the 12th Int. World Wide Web Conf.*, Budapest, Hungary, May 2003. ACM Press.

Paik, H., Benatallah, B., & Toumani, F. (2004). WS-CatalogNet: building peer-to-peer e-catalog. In *Proceedings of 6th Int. Conf. on Flexible Query Answering Systems*, Lyon, France, June 2004.

Papadimos, V., Maier, D., & Tufte, K. (2003). Distributed query processing and catalogs for peer-to-peer systems. In *Proceedings of the First Biennial Conf. on Innovative Data Systems Research*, Asilomar, CA, January 2003.

Raschia, G. & Mouaddib, N. (2002). SaintEtiQ: a fuzzy set-based approach to database summarization. *Journal of Fuzzy Sets and Systems, 129* (2), 137–162.

Voglozin, W.A., Raschia, G., Ughetto, L., & Mouaddib, N. (2004). Querying the saintetiq summaries—a first attempt. In *Proceedings of 6th Int. Conf. on Flexible Query Answering Systems*, Lyon, France, June 2004.

Wordnet, www.cogsci.princeton.edu/∼wn/.

Yan, G., Ng, W., & Lim, E. (2002). Product schema integration for electronic commerce—a synonym comparison approach. *IEEE Trans. on Knowledge and Data Eng.*,*14*(3).

Yang, B. and Garcia-Molina, H. (2003). Designing a super-peer network. In *Proceedings of 19th International Conf. on Data Engineering*, Bangalore, India.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.