WS-CatalogNet: Building Peer-to-Peer e-Catalog

Hye-young Paik[1*] and Boualem Benatallah[2] Farouk Toumani[3]

[1] School of Information Systems, Queensland University of Technology, Brisbane, Australia
h.paik@qut.edu.au
[2] School of Computer Science, University of New South Wales, Sydney Australia
boualem@cse.unsw.edu.au
[3] LIMOS, ISIMA, University Blaise Pascal, France
ftoumani@isima.fr

Abstract. One of the key issues in product catalogs is how to efficiently integrate and query large, intricate, heterogeneous catalogs. We propose a framework for building a dynamic catalog portals using catalog communities and semantic peer relationships between them. The aim is to facilitate distributed, dynamic and scalable integration of e-catalogs. Our approach is based on Peer-to-Peer architecture. Peers in our system serve as data integration mediators having individual schema to support semantically rich queries. Connections between peers are established based on domains and the relationships they represent. Schema and relationships in peers are used for routing queries among peers.

1 Introduction

Behind every *product portal* such as Amazon.com, there exists diverse, potentially large number of product catalogs which have to be *integrated and queried*. We highlight two main problems in current product portals: (i) The integration is centralised and based on a single categorisation server. It is not suited for the dynamic Web based environment where catalogs are added or removed frequently, and the content/capabilities of existing catalogs may change over time. Conventional approaches where the development of an integrated catalogs require the understanding of each of the underlying catalog are inappropriate. (ii) Modern Web catalog portals have very limited search capabilities. In fact, they mostly rely on information retrieval technology, where a keyword is searched against entire content. Instead of centralised integration, we aim to facilitate distributed, dynamic and scalable catalog integration. We make the following contributions:

– We propose the concept of *catalog community* as a means to architect the organisation and integration of a potentially large number of dynamic product catalogs. A catalog community is a container of catalogs which offer products of a common domain (e.g., `Laptops`, `PCGames`). It provides a description of desired products without referring to actual sellers (e.g., `a seller of IBM Laptops`). In order to be accessible through a community, product sellers (providers) need to register their catalogs with it. Catalog providers can join or leave any community of interest at any time. Catalog communities may newly form or disappear.

* This paper was written when the author was at CSE, University of New South Wales.

– We propose a metadata indexing model for querying the catalog community also, for representing semantics of relationships among peer catalog communities. Query routing between the communities is done via such relationships (even with no or minimal knowledge of the schema about the other party).
– We implemented a prototype named *WS-CatalogNet*. The prototype is a web service-based [3] tool for (i) building catalog communities, (ii) creating memberships between a catalog provider and a community, and (iii) defining peer relationships between communities.

The paper is organised as follows. Section 2 introduces catalog communities and how they are used in integrating/organising product catalogs as peers. In section 3, we explain how a community processes a query (both locally and collaboratively with other peer communities.). The implementation of the proposed system is discussed in section 4. Section 5 discusses related work. Finally, section 6 concludes the paper with some remarks on the ongoing work.

2 Design Overview

An online product catalog contains a set of *categorised* products that can be browsed and searched electronically. Our observations showed that most portal sites provide customers with two ways of searching: browsing via categories, searching using keywords. The two approaches can also be combined by restricting keyword searches to a category. In general, the portal will return a product as a "match", if any part of the description of the product contains the keyword. Answering queries based on keywords has obvious short-comings [8, 10]. The query model in current web portals (e.g., Amazon.com) does not allow users to query precisely using rich descriptions of the products. For example, users can only ask for "*products related to laptop*", rather than "*a laptop with SONY as a manufacturer, more than 2 year warranty*".

This section explains how we utilise the concept of catalog communities and peer relationships between them to meaningfully organise the available information space. Each catalog community is specialised in a single domain of products and defines a set of attributes that effectively describes the domain (e.g. attributes in domain of `Laptops` could be brand, CPU, size of RAM, size of HDD, weight, network capabilities etc). Using the attributes of the community, a query like "Give me a list of laptops made by Sony with at least 250 MB memory and 40GB HDD and more than 1 year warranty" becomes a trivial task. Some existing portal sites would serve this kind of query as an *advanced search* feature, where the customers use a form that lists supported searchable fields. However, for portals to provide such advanced search features, they would have to maintain (centrally) different sets of attributes for different types of products, and provide corresponding search interfaces for each type. This makes it very difficult, if not impossible, for the portal sites to provide semantically rich query. In our approach, community designer/provider maintains their own set of attributes.

2.1 Catalog Communities

A catalog community is a *container* of product catalogs of the same domain (e.g., community of `Laptops`). It provides a description of desired products without referring to

actual sellers (e.g., `www.laptopworld.com`). A community maintains a set of categories that describe the domain it supports. For example, the community of `Laptops` in Figure 1 (A) has categories which organise the products as `Used Laptops`, `Super Fast Laptops` and `Ultra Portable`. Each category in a community is associated with a set of attributes that are used in describing the products. There are a set of attributes defined for the community as a whole (the `default` category), plus attributes that are specific to each category. For example, the default category of the community of `Laptops` may have {`Name, Brand, CPU, RAM, HDD, DisplaySize, DisplayType, Price`}. The category `UsedLaptop` in community `Laptops` has extra attributes that are specific to the category such as `YearUsed, Condition`, etc.

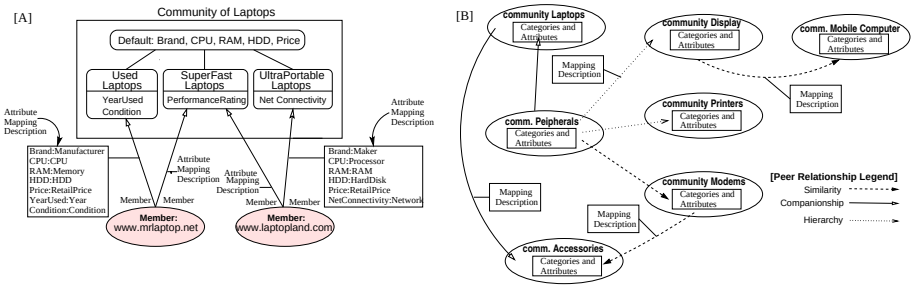

Fig. 1. (A) Community of Laptops, (B) Peer Relationships Between Communities

For a catalog to be accessible through a community, the catalog provider must register the catalog with the community. In registration, the provider associates his catalog with one or more categories of the community. Also, during registration, the provider supplies a description of how its local product attributes are mapped to the the attributes in the associated categories of the community. By registering, the catalog provider becomes a member of the community

Example 1. **Community of Laptops** Consider a community that describes laptops. Figure 1 (A) shows the community of `Laptops` which has three product categories `Used Laptops`, `SuperFast Laptops`, `UltraPortable Laptops`. It has default attributes that apply to all categories, and some category specific attributes. Let us assume that a catalog provider `www.mrlaptop.net` can provide products that belong to categories `Used Laptops` and `SuperFast Laptops`. To become a member, the catalog provider associates his catalog with categories `UsedLaptops` and `SuperFast Laptops` and provide attribute mappings between the local catalog description and the category description of the community. Here, the mapping can be incomplete, meaning that the catalog provider may not have all attributes that specified in categories of the community.

2.2 Peering Catalog Communities

A single community by itself is a fully functional, *integrated*, domain specific product catalog. It can serve semantically rich queries regarding their own domain. These individual communities collaborate with each other by forming peer-to-peer relationships (peer relationship in short). We consider three types of peer relationship:

- **Hierarchy**: the domain of one community is a specialisation/generalisation of the other (e.g., community `Printers` is a specialisation of community `Peripherals`).
- **Similarity**: the domain of one community is similar to another, in the sense that the two have common categories that are considered analogous, or interchangeable (e.g., category `CD-RW Drives` in community `SonyRetailers` and category `CD-RW/DVD` in community `CheapCDRom`).
- **Companionship**: the domain of two communities compliments each other (e.g., the user who buys a CD burner from `SonyRetailers` may want to look at some blank CDs from a community `BlankCDMedia`).

Figure 1 (B) shows a few catalog communities and peer relationships among them. Defining relationships determines how communities interact with each other. We assume that between two communities there only be one peer relationship.

Forming Peer Relationships. Let us assume that community C_1 wants to form a relationship with C_2[4]. Since the terms used to define categories and attributes are different from one community to another, the administrator of C_1 needs to express how C_2's categories and attributes are mapped to C_1 for the collaboration to happen. We consider three possible ways of describing these mappings.

- **Full support:** C_1 provides explicit, one-to-one mapping of between the attributes in C_1 (respectively, in categories) and those in C_2. Therefore, the query expressed in C_1's terms can be fully translated into C_2's.
- **Category support:** C_1 does not disclose the attributes, but provide what kind of categories are available C_1. In this case, the mapping only expresses which category in C_1 maps to which is category in C_2.
- **No Disclosure:** C_1 does not define any mapping regards to C_2's categories and attributes. In this case, no mapping is explicitly described. It is left to C_2 to figure out how to answer C_1's queries.

Communities are used to divide a vast information space into meaningful, manageable spaces. By doing so, it is much easier to agree on the global attributes (i.e., within the community) for product descriptions, because a community represents products from a single domain that would have shared, common properties. By defining the peer relationships, individual communities discover other communities in the available space and know how to collaborate with them.

[4] For space reasons, we will not discuss how C_1 discovers C_2 in details, but WS-CatalogNet provides metadata search facility for discovering other communities in WS-CatalogNet.

3 Query Processing in *WS-CatalogNet*

Two Stage Search In our approach, users can engage in two-stage information seeking activities: (i) start looking for information in general terms, (ii) then get into specific information. In the first stage (*metadata search stage*), users explore communities and their relationships in WS-CatalogNet. Metadata queries are used to discover communities. Typical examples are "*List all communities that sell DVDs*" or "*What are the related communities of Laptop*", etc. Once the user locates a community of interest, the user is now in the second stage (*product search stage*) where a more specific query is submitted to the community. In this paper, we focus on the second stage and look at how individual communities process the queries. Also, we discuss how a community collaborate with other communities (known via peer relationships) to process a query.

To submit a query to a community interest, the user examines the defined categories and attributes of the community and formulating a query using them. We assume that the query can be expressed in a simple SPJ form. For example, for the community of `Laptops`, the user can submit a query like the following: *SELECT Brand, CPU, HDD, Price FROM Category_UsedLaptop WHERE YearUsed* $<$ *3 AND Price* $<$ *2000.* Note that `FROM` part of the query is a name of the category. That is, the query is specifically directed to a category (e.g., Category_UsedLaptop).

Since the community does not store the product data locally, every query submitted to a community is re-directed to its members for processing. Also, the community can ask other communities to process the query by forwarding it to peers (depending on what the user want). Before we explain how the above query would be processed by the community, we look the elements involved in the query forwarding.

3.1 Query Forwarding among Catalog Communities

When a community forwards a query to other communities, it considers two elements: Metadata indices on the category/attributes mappings and a query forwarding policy.

Metadata Indices A community use a metadata index on the category/attribute mapping description to decide to which members (respectively, peers) the query will be forwarded to. Two types of metadata indices are used: *intra-community index* and *inter-community indices*.

Intra-community Index. The intra-community index is used for routing queries to local members of the community. When the members register, they provide descriptions regard to how their local product attributes are mapped to attributes in the community. This index is built based on which categories (respectively attributes) are supported by which member. The first level index key is on the name of the categories (including the default). The value of the key refers to a second level index key, which is on the name of attributes of the category. The value of second level key refers to a list of members that can serve the attribute in a query (i.e., a member has specified a mapping for the attributes). A member can be registered with one or more categories. Figure 2 (A) illustrates the intra-community index.

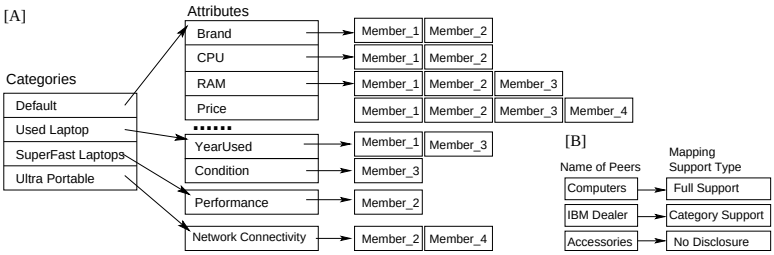

Fig. 2. In Community of Laptops: (A) Intra-Community Index, (B) Peers/SupportType Index

Inter-community Indices. The inter-community index is used for routing queries to peers. A community may need to forward a query to peers in situations like the following:

- When no matches to the query is found, rather than returning an empty result, the community may forward the query to other communities.
- The relationships between communities are formed on the basis of hierarchy, similarity or companionship semantics. Communities use these relationships to expand the size of results, or to offer the users alternatives products to what was originally requested.
- When the community temporarily unable to process the query (e.g., undergoing some changes), rather than returning a message that says "unavailable", it might be beneficial to forward the query to other communities.

There are two types of inter-community indices:

- *Peer/SupportType Index*: This type of index is used to find out what type of mapping is supported in a peer. An example in Figure 2 (B) shows that the community of `Laptop` has full mapping description of the community of `Computers`, only category mapping description of the community of `IBM Dealer` and so on.
  If C_1 is in *full support* type with C_2, the query from C_1 can freely be translated in C_2's terms. If C_1 has only category mapping description of C_2, C_1 knows to which category in C_2 the query can be forwarded. However, it still needs a way of resolving attribute names mismatch. (e.g., `BookName` in C_1 could be `Title` in C_2). When C_1 has no mapping description (i.e., no disclosure) of C_2, it is up to C_2 to address the mismatch problem before processing the forwarded query.
- *Attr-Category/Synonym Index*: To help communities solve mismatch problem, we use synonym-based matching approach. As can be seen in Figure 3 (A), an index of synonyms is kept in a community. In this type of index, a list of synonyms for each attribute (respectively, category) defined in the community is constructed For example, for the attribute `CPU`, synonyms are {`processor, chipset, chip`}. These synonyms are used, between communities, to find an alternative terms for the attributes/category used in a query.

Query Forwarding Policy Query forwarding is controlled by two elements: *peer selection* and a *forwarding policy*.

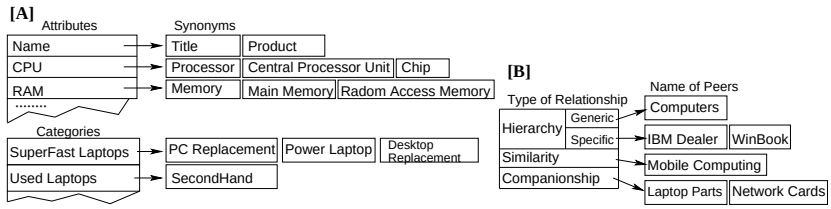

Fig. 3. In Community of Laptops: (A) Attr-Category/Synonym Index, (B) Relationship/Peer Index

– *Peer selection*: It may not be efficient for a community to forward a query to every peer. Then the question is "how do we select peers for forwarding?". We use the different types of relationships among peers as a basis for the selection process, namely; hierarchy, similarity and companionship (see Section 2.1). If the purpose of forwarding process is to expand the size of result, we may choose peers with similarity or hierarchy. If the purpose is to find alternative or complimentary products to what is was originally requested, peers with companionship relationship will be useful. In our approach, we currently make a simplified assumption that the user decides the type of relationships that she wants to use in forwarding. At the time of submitting a query, as a preference, she indicates which type of relationships is relevant to what she is expecting in the query forwarding. A type of index called *Relationship/Peer index* is used to identify the peers in each type of relationship. Figure 3 (B) shows an example of such index we use. The index key is on the type of relationships and the value of the key refers to names of peers.
– *Forwarding policy*: A forwarding policy dictates whether the forwarding is allowed or not, when the forwarding should be initiated and permitted hop count for subsequent forwarding. The basic structure of a `forward policy` is as follows:

```
forward policy: when empty|expand|always
                hop n
```

The policy requires two attributes to be specified: `when` and `hop`. The `when` is used to decide when the forwarding should be initiated. If it is *empty*, the query is forwarded when the result from local members is empty (i.e., no match) whereas *expand* sets it so that the query is forwarded even if the results from the local member is not empty. This can be useful if the community wishes to obtain larger number of tuples in the results. If the attribute `when` is set to *always*, it means that the community is currently overloaded with other requests, or temporarily undergoing changes, hence the query will be automatically forwarded without being processed locally. `hop` is used to set the limit as to how many subsequent forwarding can happen. Consider the following example:

```
forward policy: when empty
                hop 3
```

This policy allows forwarding of the query, when the result from local members is empty. The subsequent forwarding should end when the hop count reaches 3.

3.2 Query Processing

Intra-community Query Processing. Let us look at how a query is processed within a single community. At a community, the user can choose to submit a query only to a specific category or to the entire community. Assume a query $\mathtt{Q}(\mathtt{C}, \mathtt{A}, \mathtt{Condt})$, where C is a community, A is a set of of selected attributes and Condt is a set of conditions. Let V denote all attributes that appear in A and Condt. For each attribute $\mathtt{a_i}$ in V, a set $\mathtt{M_i}$ is identified, which contains the members that support $\mathtt{a_i}$ (i=1..number of attributes in V). The decision on where to forward the query to is made after the members that exists in every $\mathtt{M_i}$ are found. Consider the following two queries:

```
SELECT Brand, CPU, Price          SELECT Brand, Price
FROM Category_Default             FROM Category_UsedLaptop
WHERE RAM > '250MB'               WHERE YearUsed < 2
```

In the first query, $\mathtt{V} = \{\mathtt{Brand}, \mathtt{CPU}, \mathtt{Price}, \mathtt{RAM}\}$. According to Figure 2, {Member_1, Member_2} are candidates for answering the query. In the second query, $\mathtt{V} = \{\mathtt{Brand}, \mathtt{Price}, \mathtt{YearUsed}\}$. According to Figure 2, {Member_1} should be considered for query forwarding.

Inter-community Query Processing. Let assume two communities $\mathtt{C_1}$ and $\mathtt{C_2}$. $\mathtt{C_1}$ wants to forward a query to $\mathtt{C_2}$ via the peer relationship. An example query Q.1 is composed as follows using attributes in $\mathtt{C_1}$:

```
Q.1)  SELECT Brand, PerformanceRating
      FROM   Category_SuperFastLaptops
      WHERE  price < 3000
```

- **Full Support:** $\mathtt{C_1}$ has the full description as to how attributes (respectively, categories) in itself are mapped to $\mathtt{C_2}$. In this case, before forwarding, Q.1 is translated so that it uses terms understood by $\mathtt{C_2}$.
- **Category Support:** When $\mathtt{C_1}$ only has mappings for categories, $\mathtt{C_1}$ translate Q.1 so that the category name is understood by $\mathtt{C_2}$. Then, attach the synonym list for each attribute in Q.1. $\mathtt{C_2}$ will use the synonyms to match the attributes. Q.2 is the result of the category translation and attachment of the synonym lists.

```
Q.2)
SELECT Brand (title, product), PerformanceRating (rating, review)
FROM   Category_PowerLaptop
WHERE  price (retail price, listed price) < 3000
```

- **No Disclosure:** When there is no mapping available, synonyms for attributes (respectively for categories) are identified and attached to Q.1 before forwarding (as in Q.3).

```
Q.3)
SELECT Brand (title, product), PerformanceRating (rating, review)
FROM   Category_SuperFastLaptops (PowerLaptop, PCReplacement)
WHERE  price (retail price, listed price) < 3000
```

$\mathtt{C_2}$ refers to the synonyms to find alternative attribute/category names to match the terms in the query with $\mathtt{C_2}$'s own.

4 *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 rep-

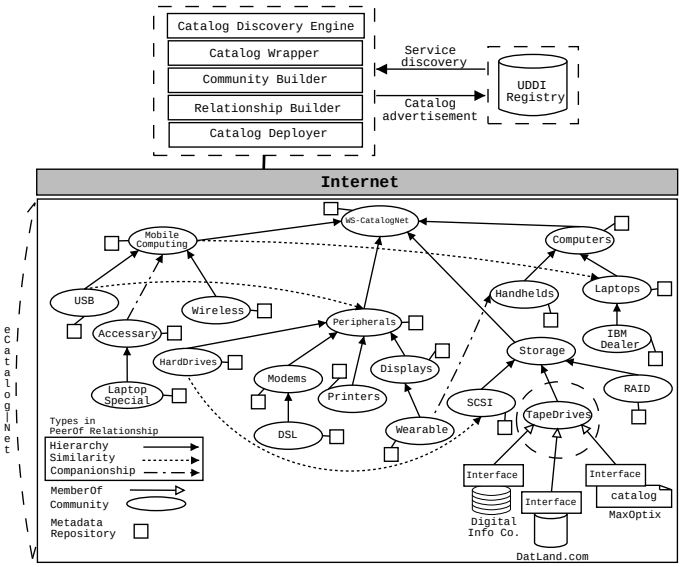

Fig. 4. *WS-CatalogNet* Architecture

resented as web services [3]. 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 1) and for communities (see Table 2). These two specific tModels are WSDL document types of tModel.

Table 1. operations in tModel for product catalog members

Operations for Member, M	**Return Type**	**Description**
`Query(String query)`	`String`	`Query M`
`GetGeneralInfo()`	`HashMap`	`Get metadata about M`
`GetAttributes()`	`HashMap`	`Get product attributes of M`

Table 2. operations in tModel for catalog communities

Operations for Community, C	**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 originComm,int hopLeft, String query)`	`Process forwarded query from originComm to C`
`Int Register(String membName,HashMap generalInfo,HashMap attributeMapping)`	`Register member MembName to C`
`Int Deregister(String membName)`	`Deregister member membName from C`
`Vector GetMemberNames()`	`Get member names registered to C`
`ECatalogInfo GetMemberInfo(String membName)`	`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.`

Figure 5 illustrates each step involved in building a catalog community (respectively a catalog member) as a web service. 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 (Web Service Description Language), and SOAP (Simple Object Access Protocol)). 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 4).

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,

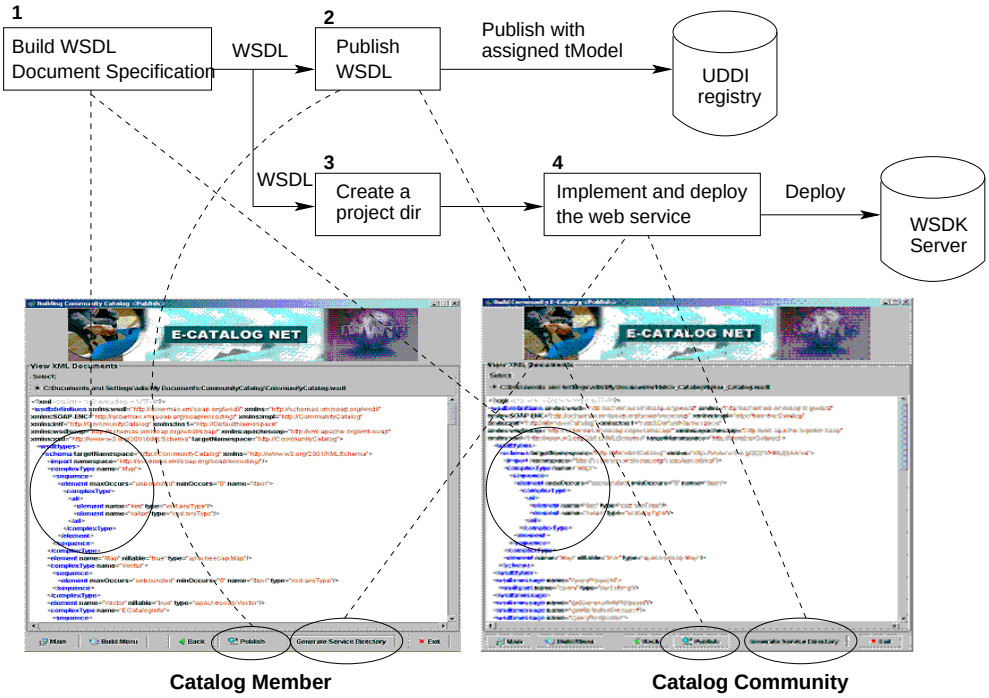

Fig. 5. Steps to build a catalog community (respectively a catalog member) using Web Services

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.

When building a community, the community provider has to download a special class named `QueryProcessor` which is provided 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, we use Oracle's XML parser 2.0 and IBM's Apache Axis 1.0.

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 maintains (i.e., insert or delete) the metadata documents about intra, inter-community indices whenever a change occurs (e.g., a member is added, a peer is removed, etc.)

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.

5 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, [11] 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. [7] 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. [4] 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.

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 ([2, 10]). 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, [9] 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. [6] 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 [1], 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. [5] 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, [2] 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.

[10] 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 [8] have proposed super-peer based routing [5] 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.

6 Conclusion and Future Work

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 about the schema of other peers. Having this framework enables us to embark on more interesting problem which is restructuring of the relationships between peers or the peers themselves. The dynamic nature of the environment we are facing with, makes it critical that the system is able to adapt to necessary changes and do self-monitoring in terms of detecting the needs for such changes automatically. For example, if certain user query is not served quickly, or not served at all, the community might look at its members to see whether it needs more providers to serve users' demand, or need more peers to forward queries. Our ongoing work concerns developing and evaluating the self-monitoring, restructuring, adaptation mechanism in WS-CatalogNet.

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

References

1. K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. The Chatty Web: Emergent Semantics Through Gossiping. In *Proc. of the 12th International World Wide Web Conference (WWW 2003)*, Budapest, Hungary, May 2003.
2. P. Berstein, F. Giunchigiloa, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, pages pp. 89–94, Madison, Wisconsin, June 2002.
3. F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and Sanjiva Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):pp.86–93, 2002.
4. D. Fensel, Y. Ding, and B. Omelayenko. Product Data Integration in B2B E-Commerce. *IEEE Intelligent Systems*, Vol. 16, Issue 4:pp. 54–59, Jul/Aug 2001.
5. A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proc. of 19th International Conference on Data Engineering (ICDE 2003)*, Bangalore, India, March 2003.
6. A. Kementisietsidis, M. Arenas, and R.J. Miller. Mapping Data in Peer-to-Peer Systems:Semantics and Algorithmic Issues. In *Proc. of ACM SIGMOD 2003 Conference*, San Diego, CA, June 2003.
7. S. Navathe, H. Thomas, M. Satits A., and A. Datta. A Model to Support E-Catalog Integration. In *Proc. of the IFIP Conference on Database Semantics*, Hong Kong, April 2001. Kluwer Academic Publisher.
8. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Lser. Super-Peer-Based Routing and Clustering Strategies for RDF- Based Peer-To-Peer Networks. In *Proc. of the 12th International World Wide Web Conference (WWW 2003)*, Budapest, Hungary, May 2003. ACM Press.
9. W.S. Ng, B.C. Ooi, K.L. Tan, and A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *Proc. of the 19th International Conference on Data Engineering (ICDE 2003)*, pages pp. 633–644, Bangalore, India, March 2003.
10. V. Papadimos, D. Maier, and K. Tufte. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. In *Proc. of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, January 2003.
11. G. Yan, W. Ng, and E. Lim. Product Schema Integration for Electronic Commerce–A Synonym Comparison Approach. *IEEE Transactions on Knowledge and Data Engineering*, 14(3), May/June 2002.
12. B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *Proc. of 19th International Conference on Data Engineering (ICDE 2003)*, Bangalore, India, March 2003.