OntoZilla: An Ontology-based, Semi-structured, and Evolutionary Peer-to-Peer Network for Information Systems and Services

Feng-Yuan Chuang
fychuang@ntu.edu.tw

Yuh-Jzer Joung*
joung@ntu.edu.tw

Department of Information Management
National Taiwan University
Taipei, Taiwan

ABSTRACT

We present a system called "*OntoZilla*", which combines ontologies and P2P systems, with a vision of improving the process of information searching and facilitating greater integration as well as interoperability. In OntoZilla, peers supporting the same concept are grouped into the same cluster, and the relationships between clusters are modeled according to the concepts they specialize in. Therefore, a query belonging to a specific concept can be routed to the suitable group of peers in a systematic way, thus supporting efficient concept search.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing protocols; H.3.5 [**Online Information Services**]: Web-based services; I.2.4 [**Knowledge Representation Formalisms and Methods**]: Semantic networks

1. INTRODUCTION

Information technology offers us fast channels of communication and huge volumes of information. Current keyword search methods can retrieve irrelevant data that contain certain terms with different meanings. However, they may miss relevant data that contain different terms with the same meaning as the desired content [6]. Besides, the links among data commonly offer little information. As a result, we have to spend a lot of time identifying the similarities, differences, and relationships among pieces of knowledge, which is tedious and time consuming. This may be attributed to the fact that computers can only aid in information searching to a limited extent, because information is mostly represented in formats that machines cannot understand well. *Ontologies* [11], which express knowledge and the relationships in the knowledge with clear semantics, can make knowledge machine-interpretable and thus improve the process of information searching.

*Corresponding author.

INFOSCALE 2007, June 6-8, Suzhou, China

DOI 10.4108/infoscale.2007.876

Moreover, the information world has evolved from a single isolated scheme to loosely-coupled networks. Heterogeneous information sources need to be integrated to share knowledge. Services and services, plus agent programs, need more interoperability to achieve maximum synergy. To make these tasks scalable and maintainable, automatic processing of information becomes necessary. Automatic processing requires machine-interpretable representation of information semantics. Ontologies are again a key technology that smooths the way to greater integration and interoperability.

Search flexibility and interoperability must be founded on a network infrastructure that facilitates distributed information sharing. Among various network schemes, P2P systems have emerged to be the most promising candidate, e.g., Gnutella [8], SETI@home [13], and Freenet [3]. In a P2P system, peers are organized into an overlay network to share resources. With an aggregation of resources at peers, P2P systems have presented overwhelming superiority over traditional centralized systems. Pitifully, the lack of semantic capability hinders the advancement of P2P systems and makes them only adopted in several fields, such as file-sharing applications.

We believe that combining ontologies and P2P systems will make them benefit from each other and thus result in synergy. In a P2P system combined with ontologies, users can annotate their resources with ontologies, and easily share the resources with each other via the overlay network. Since the resources are described in machine-interpretable formats, computers can help us precisely retrieve the desired content. These intelligent forms also enable users as well as computers to better utilize the resources. On the other hand, a wide variety of services can also benefit from the combination of ontologies and P2P systems. This is because by semantically annotating services and organizing them in a P2P network, the services can dynamically discover each other and select the suitable ones, achieving better automatic service execution, composition, and interoperation.

Many researchers have focused on a combination of ontologies and P2P systems. The work in [4] uses a clustering strategy to group peers with identical or similar content, improving the search performance in P2P networks. However, its query routing still relies heavily on unsystematic broadcasting. Note that clustering has also been used to improve search in P2P [2]. Without ontologies, however, peer relationship cannot be fully utilized to explore the network. The work in [1] is based on structured networks, which often exploit *Distributed Hash Tables (DHTs)* [12, 14] to offer efficient exact match search while ensuring scalability.

However, using bit vectors to represent information may require some compromises with respect to the use of semantic knowledge. Besides, maintaining a hash table at distributed nodes would not be an easy task, especially when nodes may join and leave the network frequently. Moreover, as peers may not be willing to accept arbitrary connections or content from others [4], the rigid topology that requires peers to manage dedicated resources would restrict peer autonomy [5, 7]. Finally, some research [10] leverages centralized mechanisms to build a network that supports flexible and expressive queries. Unfortunately, the use of centralized mechanisms makes systems vulnerable to attacks and failures, and must assume the existence of powerful and enthusiastic super-peers. All these drawbacks highlight the need for a robust design that also meets the requirement of flexibility.

In contrast to the above approaches, we design a system called **OntoZilla**, to utilize the synergy between ontologies and P2P systems, with a vision of improving the process of information searching and facilitating greater integration as well as interoperability. In a P2P system, a peer may have interests in some kinds of information, or concentrate on providing particular expertise or services. This characteristic can be viewed as the *Special Interests* of a peer. Peers in our system are organized according to their relationships, which are based on their special interests. Peer relationships can be modeled with ontologies. For example, a peer with the knowledge of "programming languages" and another peer with the knowledge of "object-oriented languages" can be related by a hierarchical relationship; a peer providing the service of "hotel reservation" and another peer providing the service of "travel agent" can be related by a cooperative relationship. Using *semantic links* to reflect the relationships among peers, queries can be routed to the peers with suitable information or services in a systematic way. Thus, our system can avoid blind flooding, which is often adopted by unstructured P2P networks.

On the other hand, since peers may join and leave the network constantly, their relationships have to be updated as the network evolves. In our system, each peer describes itself with a *peer description*, which is represented with an ontology language (e.g., OWL [9]). By exchanging peer descriptions periodically, peers can become acquainted with each other. As a result, they can evolve their relationships by referring to peer descriptions and then establishing semantic links with more suitable peers. Therefore, our system is more flexible than P2P networks based on rigid structures.

The rest of this paper is organized as follows. Section 2 presents our system design. Section 3 presents the detailed construction and maintenance algorithms. Conclusions and future work are offered in Section 4.

2. ONTOZILLA

2.1 System Overview

Special Interest Groups

In a P2P system, a peer may have interests in some kinds of information, or concentrate on providing particular expertise or services. This characteristic can be viewed as the *Special Interests* of a peer. Peers' special interests can be categorized into groups, referred to as *Special Interest Groups* (SIGs). Peers with identical or similar special interests are grouped into the same SIG. The categorization of special interests can refer to the first-level classes of some classification systems, for example, the ten main classes of the Dewey Decimal Classification system and the main categories of the Yahoo! Directory. In this paper, we use "Music", "Business", and "Computers" as the sample SIGs for illustration.

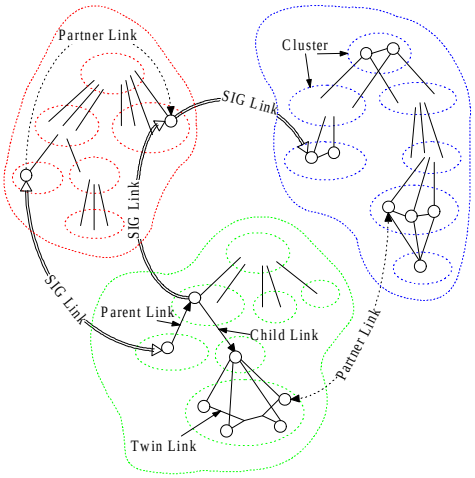

Figure 1: A Sketch of OntoZilla

Clusters and Semantic Links

Within each SIG, peers employ some classification systems to hierarchically classify their special interests into *concepts*, or *classes*. Each class can be created as a cluster that further groups peers supporting the same concept. Peers in the network are organized according to their relationships, which are based on their special interests and supporting concepts. Peer relationships can be modeled with ontologies. We design several kinds of links with semantics, or *semantic links*, to reflect peer relationships. Peers that have an inheritance relationship can connect with each other via *parent links* and *child links*; peers in the same cluster can connect with each other via *twin links*; peers in different SIGs can connect with each other via *SIG links*; peers that have a cooperative relationship can connect with each other via *partner links*. As a result, peers supporting the same concept form a cluster, and all clusters in the same SIG form a hierarchical tree. Moreover, the communication channels between different SIGs are established. Fig. 1 shows a sketch of our system. Each color in this figure represents a different SIG. The semantic links among peers can direct queries to the peers with suitable information or services, thus enabling efficient query routing.

Classification Systems

The number of the classification systems used in a SIG is not limited. For simplicity, however, we assume that peers in a SIG use only one classification system. In a classification hierarchy, each class is associated with at least one node of the hierarchy. The link between a parent node and a child node denotes an "*is subclass of*" (is-a) relationship. The classes that have direct is-a relationships with a lower class are called its *parent classes*, whereas the classes that have direct is-a relationships with a higher class are called its *child classes*. The classes that have indirect is-a relationships with a lower class are called its *ancestor classes*, whereas the classes that have indirect is-a relationships with a higher class are called its *descendant classes*. Parent and ancestor classes are all called *super-classes*, whereas child and descendant classes are all called *subclasses*. For simplic-

ity, we only consider the single inheritance scenario. Therefore, a class can have only one parent class, but have multiple child classes.

The class associated with the root node of the classification hierarchy is the root class of the SIG, and its class name follows the name of the SIG. The name of its subclass follows the name of the associated node in the hierarchy, and so on. For the purpose of familiarity and illustration, we assume that peers in the "Computers" SIG use the *ACM Computing Classification System* (CCS) to classify information and services. Hence, in this SIG, the name of the root class is "Computers (0)", and the names of its child classes are "Hardware (1)", "Computer systems organization (1)", "Software (1)", "Computing methodologies (1)", etc. Each number in parenthesis denotes a cluster's hierarchical level in the CCS.

Ontology-based Descriptions

Each SIG is described with a *SIG description*. A SIG description contains the **SIG name** and some **annotations** of the SIG. SIG descriptions are known by all peers in the network. Similarly, each class is described with a *class description*. A class description contains the **class name**, the **SIG name**, the **classification hierarchy**, and some **annotations** of the class. SIG and class descriptions are represented with ontology languages; therefore, the *similarity measures* between SIGs or between classes can be intelligently evaluated with semantic matching techniques. A similarity measure is used to determine the semantic similarity between two matters. If the similarity measure exceeds a specific *matching threshold*, we can say that the two matters describe the same thing.

Peers describe their special interests also with ontology languages. Therefore, peers' special interests can be compared with SIG or class descriptions to evaluate their similarity measures. Each peer joins the SIGs that are semantically similar with its special interests, and further joins the clusters that are semantically similar with its supporting concepts. A peer may join more than one SIG and more than one cluster. For presentation purposes, we assume that each peer joins only one SIG and one cluster. Therefore, each peer joins the network first by selecting a SIG that most matches its special interests. Then, it selects a cluster that most matches its supporting concepts and joins that cluster.

Ontology-based, Semi-structured, and Evolutionary Network

Peers in the network use semantic links to reflect their relationships. Peers supporting the same concept form a cluster, and all clusters in the same SIG form a hierarchical tree, called the *cluster tree*. Moreover, the communication channels between different SIGs are established. These designs result in an ontology-based and semi-structured network topology. According to the reflected peer relationships, queries can be routed to the peers with suitable information or services in a systematic way. Thus, our system can avoid blind flooding, which is often adopted by unstructured P2P networks.

On the other hand, since peers may join and leave the network at will, their relationships have to be updated as the network evolves. In OntoZilla, each peer describes itself with a *peer description*, which is represented with an ontology language. A peer description contains the peer's **SIG name**, **SIG ID**, **class name**, **cluster ID**, **IP address**, along with some **additional information** that can be used to judge the adequacy of the peer. The SIG ID is generated by the first peer, or *creator peer*, of the SIG. It is the creation timestamp concatenated with the creator peer's IP address. The cluster ID is generated in a similar way. SIG IDs and cluster IDs are used to detect partition.

By exchanging peer descriptions periodically, peers can become acquainted with each other. Consequently, they can evolve their relationships by referring to peer descriptions and then updating semantic links, resulting in an evolutionary network. In addition, when a peer needs to establish semantic links to peers in a cluster or a SIG as its neighbor peers, if there are several choices, it can flexibly select one according to some *neighboring criteria*, such as physical distance proximity, availability, security, privacy, and trust. Peers can establish semantic links with each other in this flexible manner, thus enhancing the efficiency and effectiveness of the network. Therefore, our ontology-based, semi-structured, and evolutionary P2P network, OntoZilla, is more flexible than P2P networks based on rigid structures.

2.2 Semantic Links

A peer establishes semantic links to other peers by simply adding their peer descriptions into its routing table. The peers recorded in the routing table are called its *neighbor peers* since it has direct links to them. Semantic links can be categorized into three types:

- **SIG links**, which are used to connect the peers of different SIGs. The link connecting peer x of SIG_x to peer y of SIG_y is represented as $x \Rightarrow y$.
- **Family links**, which are used to connect the peers of the same SIG. The link connecting x of SIG_x to y also of SIG_x is represented as $x \rightarrow y$.
- **Partner links**, which are used to establish a cooperative relationship between peers. For example, if peer x has to frequently retrieve information or acquire services from peer y, then x can maintain this relationship by establishing a partner link to y, thereby to speed up subsequent accesses to y. The link is represented as $x \dashrightarrow y$.

If two peers establish semantic links with each other, their relationship can be represented with a bidirectional arrow, e.g., $x \Leftrightarrow y$, $x \leftrightarrow y$, and $x \dashleftarrow\dashrightarrow y$.

In each SIG, peers establish family links with each other to reflect the relationships within their SIG. Family links can be further divided into the following four types:

- **Twin links**: Peers belonging to the same class are grouped into a cluster. They connect with each other via *twin links*. We use $\xrightarrow{T}$ to denote a twin link. The topology in a cluster is not restricted to any specific structure. However, we require the structure of a cluster to be connected to support exhaustive search within the cluster. Note that the size of a cluster is typically small. Thus, if one wishes, he/she can simply organize peers in a cluster as a complete graph.
- **Parent links** & **Child links**: A peer connects to peers of its parent class with *parent links*, whereas it connects to peers of its child classes with *child links*. We use $\xrightarrow{P}$ to denote a parent link, and use $\xrightarrow{C}$ to denote a child link.
- **Ancestor links** & **Descendant links**: Clusters in a SIG may be created or destroyed as the network evolves; thus, not all clusters of a SIG may exist in the

network. As a result, a peer may need to establish direct connections with the peers of its closest ancestor class (via *ancestor links*) or closest descendant classes (via *descendant links*). We use $\xrightarrow{A}$ to denote an ancestor link, and use $\xrightarrow{D}$ to denote a descendant link.

A cluster's parent cluster and directly-connected ancestor clusters are called its *direct higher clusters*, whereas a cluster's child clusters and directly-connected descendant clusters are called its *direct lower clusters*.

- **Cousin links**: Since not all clusters of a SIG may exist, the cluster tree may be partitioned into several subtrees without common nonempty superclass. To connect these subtrees, peers in the root clusters of disjoint subtrees establish *cousin links* with each other. We use $\xrightarrow{O}$ to denote a cousin link.

2.3 Evolution of Peer Relationships

Since peers may join and leave the network at will, peers have to update their relationships as the network evolves. This is done by *peer gossip*, which lets peers periodically exchange their peer descriptions and thereby know the evolution of the network. Family links are updated/replaced in accordance with the *family link priorities*, as follows: For each peer, the priority of parent links is higher than that of ancestor links, and the priority of child links is higher than that of descendant links. For example, suppose a peer in the cluster "Distributed databases (4)" has established ancestor links to its ancestor cluster "Computer-communication networks (2)", but later on it finds that its parent cluster "Distributed systems (3)" has been created. As a result, it can establish parent links to the parent cluster, and may then drop its existing ancestor links.

Similarly, for all ancestor links of a peer, the closer the ancestor class's level, the higher the ancestor link's priority. Likewise, the closer the descendant class's level, the higher the descendant link's priority. Moreover, if a peer finds that its parent or an ancestor cluster has been created, it may drop its cousin links since its own cluster is no longer the root cluster. So the priority of parent and ancestor links are higher than the priority of cousin links.

Finally, recall that our semi-structured property allows a peer to flexibly select peers to establish semantic links, as compared to a rigid structure imposed by DHT-based P2P systems. Peer gossip also allows peers to learn of "better" neighbors, so as to improve the efficiency and effectiveness of our system.

3. CONSTRUCTION AND MAINTENANCE

In this section, we present the construction and maintenance algorithms of OntoZilla.

3.1 Join

A peer, x, first determines its own SIG, SIG_x, and class, X, and then joins the network in accordance with the *SIG join process* and the *class join process*. The goal of the SIG join process is to have x establish SIG links with peers in other SIGs, thereby opening the communication channels between x and other SIGs. The class join process allows x to join its own cluster and establish family links with other peers in SIG_x, thus enabling efficient routing within this SIG.

3.1.1 SIG Join Process

To begin with, peer x randomly selects a peer, say y, as the *contact peer* and connects to the network. Next, x broadcasts SIG inquiry messages through y within the range MAX_HOP_SIG_INQUIRY. Each SIG inquiry message contains x's SIG name and IP address. All the peers that receive the messages and know peers of SIG_x will reply to x with SIG answer messages. Each SIG answer message contains some IP addresses of peers in SIG_x.

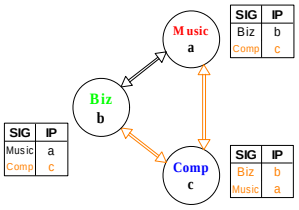

Figure 2: A SIG Join Example

If x receives SIG answer messages, it selects a peer, say z, of the same SIG, and joins this existing SIG with the help of z. Otherwise, x creates a new SIG that most matches its special interests, and creates a cluster that most matches its supporting concepts.

Then, x must exchange peer descriptions with other peers. Peer x broadcasts peer join messages, which carry its peer descriptions, within the range MAX_HOP_PEER_JOIN. All the peers that receive the messages are informed of x's participation in SIG_x, and then send their peer descriptions to x. Among those peers, the peers of different SIGs from SIG_x can add SIG links to x into their routing tables, establishing the communication channels to SIG_x. The peers of SIG_x can establish family links to x or replace some of their existing family links with the new link to x according to the family link priorities. The receiving peers can also replace some of their existing neighbors with x if x is superior to them.

On the other hand, after x receives peer descriptions for its join messages, for each SIG that is different from SIG_x, x has to select some peers of that SIG according to some given neighboring criteria, and then add SIG links to them into its routing table. Also, x has to establish family links with peers of the same SIG in the class join process.

We illustrate the SIG join process with the following example. First, peer a, whose SIG is "Music", joins the network and creates a "Music" SIG. Next, peer b, whose SIG is "Business", joins the network by using peer a as its contact peer. Then, b broadcasts SIG inquiry messages through a within the range MAX_HOP_SIG_INQUIRY. As b cannot find the "Business" SIG, it creates a new SIG. Following that, it broadcasts peer join messages through a within the range MAX_HOP_PEER_JOIN. After a is informed of the creation of the "Business" SIG, it sends its peer description to b, and then adds a SIG link to b into its routing table.

On the other hand, after b receives the response description, it adds a SIG link to a into its routing table. As a result, a and b establish SIG links with each other. Subsequently, peer c, whose SIG is "Computers", joins the network by using b as its contact peer. It then follows a similar procedure to create a "Computers" SIG and establish SIG links to b and a, each which will also establish a SIG link to c. The result is depicted in Fig. 2.

From this figure, we can see that each peer has SIG links to all the other SIGs. However, as the network size grows, peers in different SIGs may not reach each other immediately. In OntoZilla, each peer periodically broadcasts peer gossip messages within a random range. Like a peer join message, a peer gossip message also contains a peer description;

thus, peers can establish semantic links also by referring to gossip messages. Through a chain of peer gossip, peers in different SIGs can become acquainted and eventually reach each other.

3.1.2 Class Join Process

In the SIG join process, x knows some peer z in its SIG when it discovers an existing SIG it belongs to. Peer x can then use z as a "*guide peer*" to join its class cluster in the SIG. The class join process is divided into two stages: the *class finding stage* and the *family linking stage*.

Algorithm 1 peer x.**joinCluster**()

```
if x can find its own cluster X then
    ⟨* Case 1 *⟩
    joinThroughCluster(X);
else if x can find its parent cluster P then
    ⟨* Case 2 *⟩
    create a cluster X;
    joinThroughCluster(P);
else if x can find its ancestor cluster A then
    ⟨* Case 3 *⟩
    create a cluster X;
    joinThroughCluster(A);
else if x can find its child cluster C then
    ⟨* Case 4 *⟩
    create a cluster X;
    joinThroughCluster(C);
else if x can find its descendant cluster D then
    ⟨* Case 5 *⟩
    create a cluster X;
    joinThroughCluster(D);
else
    ⟨* Case 6: x's own cluster is disjoint with all existing subtrees
    in this SIG *⟩
    create a cluster X;
    O ← selects the root cluster of a disjoint subtree;
    joinThroughCluster(O);
end if
```

Class Finding Stage

In this stage, x has to find its own cluster. However, x may be the first peer of its class to join the network. If x cannot find its cluster, it finds the cluster that is closest to its hierarchical position, and joins the network through that cluster. The process is as follows. First, x copies z's routing table to itself. Since peer relationships within a SIG are guided by family links, x can find the closest cluster without the need to flood the whole network. Through family links, x can visit clusters in turn and approach its own cluster gradually. Then, x joins a cluster in accordance with Algorithm 1. This algorithm uses another algorithm **joinThroughCluster**, which allows x to establish family links in the family linking stage.

Family Linking Stage

In this stage, x has to establish family links with peers in the same SIG. Before presenting the algorithms of this stage, we first see an example of clusters depicted in Fig. 3. In this figure, peers belonging to the same cluster are enclosed with a dashed line. Each cluster is also given a more meaningful class name taken from the ACM CCS. Peers x_1 and x_2 are in cluster X (of "Computer-communication networks (2)"). Likewise, p_1, p_2, and p_3 are in cluster P, which is the parent cluster of X. Cluster X has two child clusters C_i and C_j. X also has a descendant cluster D. Peers in D establish ancestor-descendant links with X since the parent cluster of D has not yet been created. Clusters C_i, C_j, and D are all direct lower clusters of X, whereas X is the direct higher cluster of them.

From Fig. 3, we can see that each peer has to maintain

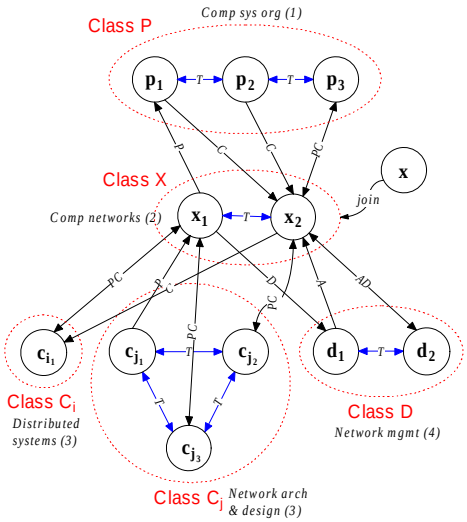

Figure 3: An Example of Clusters

some family links. First, peers in the same cluster connect with each other via twin links. In addition, each peer has to maintain parent links to its parent cluster, such as the link connecting peer x_1 to p_1. Moreover, each peer has to maintain child links to every child cluster. For example, peer x_2 has a child link to c_{i_1} in its child cluster C_i, and another link to peer c_{j_2} in its child cluster C_j. Ancestor and descendant links are maintained in a similar way. Hence, all peers in cluster X have to maintain descendant links to cluster D, whereas all peers in cluster D have to maintain ancestor links to X. By connecting peers in this fashion, a peer can reach at least one peer in its own cluster, in every direct higher cluster, and in every direct lower cluster.

3.1.3 Fault Tolerance

In OntoZilla, each peer in a SIG has to maintain family links to its own cluster, direct higher clusters, direct lower clusters, and cousin clusters. The minimal number of links that a peer needs to connect to each cluster is referred to as `MIN_LINK_TO_CLUSTER`.

Consider the following scenario. As discussed in Section 2.3, when a peer x finds that its parent cluster P is newly created, it may drop its ancestor links since the priority of parent links is higher than that of ancestor links. However, if the creator peer of P later leaves the cluster before other peers join P, peers in x's cluster have to re-establish ancestor links to their ancestor cluster. The problem becomes more evident if the creator is the only peer in the cluster and joins and leaves the network very often. To cope with this problem, we enhance fault tolerance with the following rule.

RULE 1. *Only when the number of links a peer uses to connect to a specific cluster is greater than or equal to* `MIN_LINK_TO_CLUSTER`*, can the peer drop its links to another cluster with a lower family link priority.*

Fault tolerance can be further enhanced by the following rule:

RULE 2. *When the number of peers in a specific cluster is less than* `MIN_LINK_TO_CLUSTER`*, all the peers that have links to the cluster have to establish links to another cluster with a lower family link priority.*

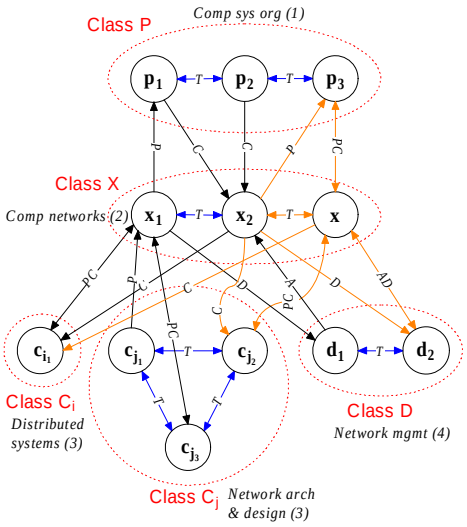

Figure 4: An Example of Case 1

3.1.4 Algorithms of the Family Linking Stage

We now present the algorithms of the family linking stage. According to Algorithm 1 in the class finding stage, peer x may encounter six cases. In the family linking stage, the way to establish family links in Cases 2 and 3 are similar. Also, the way to establish family links in Cases 4 and 5 are similar. Hence, we present only the algorithms of the family linking stage in Cases 1, 2, 4, and 6.

Case 1 (x joins through its own cluster). The case concerns the situation in which x can locate its own cluster X in the class join process. For illustration, Fig. 3 shows a sample network before x joins.

As discussed in Section 3.1.1, after x joins/creates a SIG, it must exchange peer descriptions with other peers. To do so, x first selects at least `MIN_LINK_TO_CLUSTER` peers from cluster X, and then broadcasts peer join messages through these peers within the range `MAX_HOP_PEER_JOIN`. As a result, some peers in cluster X, in X's direct higher clusters, and in X's direct lower clusters, can receive the join messages. In addition, peers in other SIGs may also receive the messages. All of the peers that receive the messages are informed of x's participation in SIG_x and X. They reply to x's join messages by sending their peer descriptions to x. Then, according to their relationships with x, they can add semantic links to x or update their existing semantic links. Specifically, among those peers, the peers in X can add/update their twin links; the peers in P can add/update their child links; the peers in X's direct lower clusters can add/update their parent links or ancestor links; the peers in other SIGs can add/update their SIG links. All of the peers can select their neighbor peers according to some given neighboring criteria.

On the other hand, through the received peer descriptions, x can establish family links to peers in SIG_x as follows. First, x selects at least `MIN_LINK_TO_CLUSTER` peers from its own cluster to establish twin links to them; at least `MIN_LINK_TO_CLUSTER` peers from its parent cluster to establish parent links to them; and for each child cluster, x selects at least `MIN_LINK_TO_CLUSTER` peers from the cluster and establishes child links to them. Also, if cluster X has descendant clusters, such as D in Fig. 3, x has to establish descendant links in a similar way. As usual, in the above x can select its neighbor peers according to some given neighboring criteria. Fig. 4 shows the network after the join process. The differences from Fig. 3 are colored with orange.

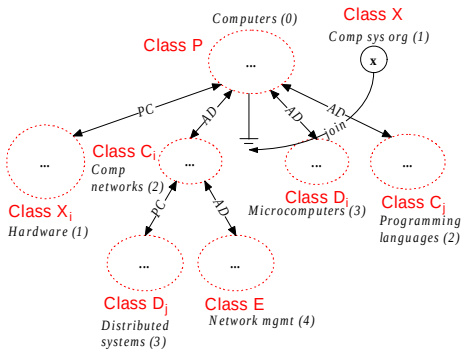

Figure 5: An Example of Case 2: Before x joins

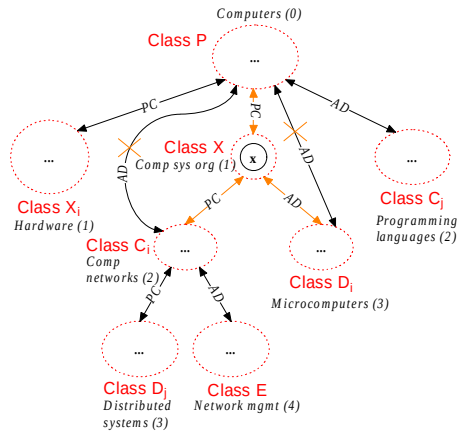

Figure 6: An Example of Case 2: After x joins

Case 2 (x joins through its parent cluster). This case concerns the situation in which x cannot find its own cluster, but can find its parent cluster P. Fig. 5 shows a sample network before x joins. In this example, x's class X is "Computer systems organization (1)"; however, the corresponding cluster has not yet been created. The root cluster P has some direct lower clusters. Class C_i is class X's child class, but class C_j is not. (C_j's parent class is "Software (1)"). In addition, D_i is X's descendant class. According to Algorithm 1, x creates a cluster X, and then joins through its parent cluster P. Then in the family linking stage, x sends a peer join message to an arbitrary peer in P to inform it of the creation of X. The peer then broadcasts x's peer join messages throughout cluster P. The messages are also broadcast throughout the clusters of X's child and descendant classes (i.e., C_i and D_i in the example).

As a result, all peers in P, C_i, and D_i, and perhaps some peers in other SIGs, will receive x's peer join messages. All of the receiving peers then exchange peer descriptions with x and evolve their semantic links. Fig. 6 illustrates the network after x joins. According to Rule 1 of fault tolerance, as cluster X grows, the ancestor-descendant links between P and C_i, and the ancestor-descendant links between P and D_i, can be dropped.

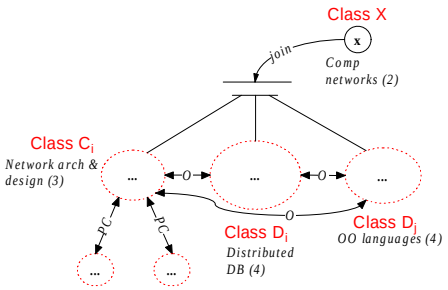

Figure 7: An Example of Case 4: Before x joins

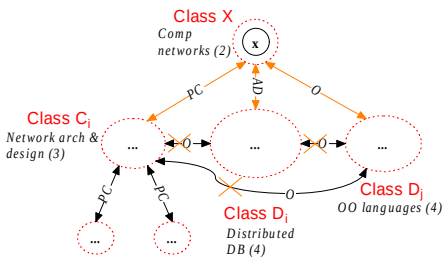

Figure 8: An Example of Case 4: After x joins

Case 4 (x joins through its child cluster). The case concerns the situation in which x cannot find its own, parent, or ancestor clusters, but can find its child clusters. Fig. 7 shows a sample network before x joins. In this example, x's class X is "Computer-communication networks (2)"; however, the corresponding cluster has not yet been created. Class C_i is X's child class. In addition, D_i is X's descendant class, but D_j is not. As clusters C_i, D_i, and D_j have neither parent nor ancestor cluster, they must be the root clusters of disjoint subtrees. Thus, peers in these clusters establish cousin links with each other.

After x finds a child cluster, say, C_i, it creates a cluster, X, and then sends peer join messages to an arbitrary peer in C_i to inform it of the creation of X. The peer then broadcasts x's peer join messages throughout C_i. The messages are also broadcast throughout all the cousin clusters of C_i (i.e., D_i and D_j in the example).

As a result, all peers in C_i, D_i, and D_j, and perhaps some peers in other SIGs, will receive x's peer join messages. All of the receiving peers then exchange peer descriptions with x and evolve their semantic links. Fig. 8 illustrates the network after x joins. Through x's peer join messages, peers in D_j find that X becomes the new root cluster of the disjoint subtree. Therefore, they establish cousin links to x. As X grows, peers in C_i, D_i, and D_j, may drop their original cousin links.

Case 6. If x cannot find its own, parent, ancestor, child, or descendant clusters, its cluster must be disjoint with all the existing subtrees in this SIG. The root clusters of disjoint subtrees in the same SIG are connected with cousin links. Each peer in these root clusters has to maintain at least `MIN_LINK_TO_CLUSTER` cousin links to every other root cluster.

3.2 Query Routing

Queries in OntoZilla are represented with ontology languages. Therefore, they can be classified into specific SIGs and classes with semantic matching techniques. Then, they are routed in accordance with the *intra-SIG routing* and the *inter-SIG routing*.

Intra-SIG Routing. Queries that request resources belonging to a peer's own SIG are routed according to the intra-SIG routing. If a peer receives a query that requests resources belonging to its super-classes, it forwards the query to higher clusters. On the contrary, if it receives a query that requests resources belonging to its subclasses, it forwards the query to lower clusters. Moreover, if a peer receives a query that requests all resources belonging to its own class, it broadcasts the query throughout its cluster via twin links.

If the querying peer is not satisfied with the number of answers, it may issue a query that requests resources belonging to a class including its subclasses, since the resources of subclasses can be viewed as a part of the resources of their super-classes. Still, if the querying peer is not satisfied the number of answers, it may adjust the class of the query to a higher level.

Inter-SIG Routing. Queries that request resources outside a peer's own SIG are routed according to the inter-SIG routing. If a peer needs resources outside its own SIG, it looks up its routing table and forwards the query to a peer in the suitable SIG via SIG links. If its routing table does not contain any peer in the suitable SIG, it broadcasts SIG inquiry messages, which are also employed in the SIG join process, within the range `MAX_HOP_SIG_INQUIRY` to find peers in that SIG. After the peer finds a peer in that SIG and sends the query to it, the query is then routed according to the intra-SIG routing. Therefore, each peer in the network can serve as the window that receives queries dedicated to its own SIG.

3.3 Stabilization

The correctness and efficiency of query routing in our system rely on the semantic links among peers. However, peers recorded in a routing table may be unavailable since peers may fail or leave the network unpredictably. Besides, due to propagation delay and failures, peers' views of the network may be inconsistent. We present two methods to improve the stability of the network.

Peer Probe. In OntoZilla, each peer regularly sends peer probe messages to its neighbor peers to probe their availability, and then drops those semantic links to unavailable peers. Lost semantic links can be recovered *actively* or *passively*. In the active approach, a peer recovers the unavailable links by copying equivalent links (i.e., semantic links with the same SIG or class as the unavailable links) from its neighbors' routing tables. In contrast, when the number of links from a peer x to a given cluster is still greater than zero, the passive approach allows x to defer the recovery action until it is acquainted with some other peer in the cluster, for example, when some new peer informs x of its join into the cluster and exchanges with x its peer description.

Peer Gossip. Peer gossip is the periodical exchange of peer descriptions within some range to allow peers to update their views of the network and to correct their family links if needed. Another purpose of peer gossip is to allow peers to discover more suitable neighbor candidates. By exchanging peer descriptions periodically, peers can establish semantic links with peers that better satisfy the neighboring criteria than existing neighbors. Furthermore, peer gossip can detect partition of clusters and SIGs, as shall be seen in the

next section.

3.4 Cluster Merging and SIG Merging

3.4.1 Cluster Merging

Due to concurrent join and propagation delay, disjoint clusters of the same class in a SIG may be created by different peers. Cluster partition can be detected early in the family linking stage, or later on through peer gossip. When a peer discovers a cluster partition, it will notify some peers in the partitioned clusters to start the *cluster merging process*. Merging two partitioned clusters is rather simple. First of all, the two clusters are compared by their IDs. The one with a smaller ID is the winner, while the other is the loser. The loser cluster will be merged into the winner, as the winner is created earlier and so might have a higher chance of containing more peers. Let x_1 and x_2 be two peers of partitioned clusters that are aware of the partition, and assume that x_1 is from the loser cluster. The cluster merging process is as follows: x_1 changes its cluster ID to that of x_2, and then establishes new twin links with x_2. Following that, x_1 broadcasts cluster merging messages throughout the loser cluster via its original twin links. Each message contains the ID of the winner cluster so that each peer in the loser cluster can change its cluster ID to the new one. After all peers in the loser cluster have corrected their cluster IDs, the cluster merging process then terminates.

3.4.2 SIG Merging

Like cluster partition, two SIGs of different IDs may be created simultaneously due to concurrent join, propagation delay, and the TTL limit. Still, partitioned SIGs can be discovered through peer gossip as described before.

To merge two partitioned SIGs, the two SIGs are also compared by their IDs, and the one with larger ID will be the loser, which will then be merged into the winner SIG. Let x_1 and x_2 be two peers from the SIGs that are aware of the partition, and assume that x_1 is from the loser SIG. To merge x_1's SIG into x_2's, x_1 copies x_2's family links, and packs them into a SIG merging message. Next, x_1 uses this message to traverse all clusters in its SIG, notifying some peers in every cluster of the merging process. The SIG merging message can traverse all the clusters in a SIG in the way of *Depth-First-Search* or *Breadth-First-Search*.

Following that, all the peers that receive the SIG merging message, along with x_1, have to serve as the "pioneer peers" of merging their clusters into the winner SIG. A pioneer peer p proceeds as follows. Since the SIG merging message contains some peer descriptions of the winner SIG, p can select a peer in the winner SIG as its guide peer, and then joins a cluster in the winner SIG as a new peer would join that cluster. If p's cluster does not exist in the winner SIG, it has to create a new cluster in the winner SIG by itself. Next, according to the process in the family linking stage, p establishes family links with peers in the winner SIG.

After that, p broadcasts SIG merging messages throughout its original cluster via its original twin links. Each message contains the new cluster ID and the new SIG ID so that each of p's original twin neighbors can change its SIG ID and cluster ID to the new ones. After all pioneer peers and their original twin neighbors have corrected their SIG IDs and cluster IDs, the SIG merging process then terminates.

3.5 Leave

Before a peer x leaves the network, it sends leave messages to all its neighbor peers. Each leave message contains some peer descriptions of x's twin neighbors. Therefore, all the peers that receive the messages and have semantic links to x can replace their links to x with the links to its twin neighbors. For peers that have semantic links to x but did not receive x's leave messages, they can still actively or passively recover the links to x as discussed in Section 3.3.

4. CONCLUSIONS AND FUTURE WORK

To conclude, OntoZilla combines ontologies and P2P systems, with a vision of improving the process of information searching and facilitating greater integration as well as interoperability. It organizes peers according to their relationships, which are based on their special interests and supporting concepts. The system can be applied to many areas. For example, it can be used as a platform for concept search. Since peers supporting the same concept are grouped into the same cluster, and the relationships between clusters are modeled according to the concepts they specialize in, a query belonging to a specific concept can be routed to the suitable group of peers in a systematic way. The full paper will present experimental results to evaluate the system.

5. REFERENCES

[1] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. V. Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In ISWC 2004, LNCS 3298, pp. 107–121.

[2] M. Bawa, G. S. Manku, and P. Raghavan. SETS: search enhanced by topic segmentation. In SIGIR 2003, pp. 306–313.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In DIAU 2000, LNCS 2009, pages 46–66.

[4] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Stanford University, 2002.

[5] N. Daswani, H. Garcia-Molina, and B. Yang. Open Problems in Data-Sharing Peer-to-Peer Systems. In ICDT 2003, LNCS 2572, pp. 1–15.

[6] J. Davies, D. Fensel, and F. van Harmelen. *Towards the Semantic Web: Ontology-Driven Knowledge Management*, chapter 1. John Wiley & Sons, 2003.

[7] Z. Despotovic and K. Aberer. Possibilities for Managing Trust in P2P Networks. Technical Report IC/2004/84, EPFL, Switzerland, 2004.

[8] Gnutella. `http://www.gnutella.com`.

[9] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview. `http://www.w3.org/TR/owl-features/`, April 2004.

[10] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In WWW 2003, pp. 536–543.

[11] L. Obrst. Ontologies for Semantically Interoperable Systems. In CIKM 2003, pp. 366–369.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In SIGCOMM 2001, pages 161–172.

[13] SETI@home: Search for Extraterrestrial Intelligence at home. `http://setiathome.ssl.berkeley.edu`.

[14] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In SIGCOMM 2001, pages 149–160.