VPeers: A Peer-to-Peer Service Discovery Framework for Virtual Manufacturing Organizations

Anbo Xiang
Liming Liu
Qiong Luo

Available at: https://works.bepress.com/limingliu/18/
VPeers: A Peer-to-Peer Service Discovery Framework for Virtual Manufacturing Organizations

Anbo Xiang¹, Liming Liu², Qiong Luo³

¹ Development Research Center of the State Council, Chao Yang Men, Dong Cheng, Beijing
abxiang@drc.gov.cn

² Department of Logistics, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
ltliulm@polyu.edu.hk

³ Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
luo@cse.ust.hk

Abstract

We present VPeers, a peer-to-peer service discovery framework for a Virtual Manufacturing Organization (VMO). In our framework, each VPeer (a VMO member) publishes the services that it intends to share and maintains a list of friend VPeers that it has recently interacted with. When a VPeer receives a search query, it will first search its local services. If the search query is not satisfied locally, this VPeer will forward the query to its friend VPeers with an additional time constraint. This process continues recursively until the search query is fully satisfied or the time allowed by the constraint has elapsed.

The features of the VPeers framework include: (1) a fully distributed search process; (2) a search query involving both string matches and range predicates; and (3) a ranking scheme that ranks search results by predefined quality measures. Our prototype implementation shows the feasibility of this approach.

Keywords: Virtual Manufacturing Organization; Peer-to-Peer; Decentralized Service Discovery

1. Introduction

A VMO (Virtual Manufacturing Organization) is a dynamically integrated, synthetic manufacturing environment in which enterprises communicate, collaborate, and interoperate with one another over the Internet or enterprise intranets (Figure 1). It has been proposed as one of the most advanced and efficient forms for modern networked manufacturing organizations [25], since every member in a VMO can concentrate on its own core competencies and use them to provide state-of-the-art services. Moreover, each member can accomplish a wide scope of production activities by leveraging the worldwide resources and services of other members with minimum investment risks. For example, enterprises in a VMO share their machinery resources, such as milling machines and lathes with Internet connection and entity interface (which provides the channel for a member to transmit and transform control commands and data). In order for VMO members to utilize resources and services efficiently, a middle agent may be used for presenting, managing, and locating the services [6] and the service information could be maintained in a centralized database in the VMO. However, due to the dynamic nature of a VMO and the autonomy of each VMO member, the lack of flexibility and robustness of the centralized architecture makes the centralized service discovery process undesirable. This motivates the proposed peer-to-peer service discovery framework, which we call VMO Peers or simply VPeers.

In this fully distributed (or pure) peer-to-peer service discovery framework, each VMO peer serves as both a client and a server for service discovery. Furthermore, there is no coupling between the network topology and the service location in this pure peer-to-peer network. This is suitable for the VMO environment, because there is no need for central coordination and the members can join and leave autonomously. In contrast, a pure centralized approach or a hybrid peer-to-peer (where some functions are still
Two important research questions need to be addressed in order to implement a pure peer-to-peer service discovery framework. First, how can a VMO peer locate and search the services of other VMO peers efficiently? We chose to trade centralized control (and possibly higher performance) for flexibility and robustness, but we still want to guarantee an acceptable search performance for each VMO peer. Second, how does a VMO peer effectively represent and publish the information about its available services? The VMO peers may be heterogeneous in various aspects. Consequently, it is mandatory that the peers agree on a common format of information presentation.

In our VPeers framework, each VPeer publishes its services that it intends to share in a common format. It maintains some knowledge about other VPeers from whom it has recently received services. When a VPeer receives a search query, it will first search the service locally. If the search query is not fully satisfied locally, the VPeer will forward the query to its friend VPeers with an additional time constraint. This process continues recursively until the search query is fully satisfied or the time allowed by the constraint has elapsed.

The VPeers framework has three features: (1) a fully distributed search process, in which each VPeer contacts its friends based on the friendship that it records over time; (2) a search query involving both string matches and range predicates, because VMO services have text fields such as names and categories as well as numeric attributes (e.g. service response time and price); (3) a ranking scheme that ranks search results by quality measures which are defined based on the semantics of the search attributes.

The remainder of the paper is organized as follows. We provide a brief overview of related work in section 2. In section 3, we present the system architecture of the VPeers framework and describe the components. In section 4, we describe in detail the peer-to-peer service discovery process in the VPeers framework. We present the prototype implementation of the VPeers framework in section 5. In section 6, we discuss our experimental evaluation of the prototype system. Finally, in section 7, we conclude the current work and provide suggestions for future research.

2. Related Work

Agent location mechanisms and agent-based virtual enterprise integration have received much attention in the literature [2, 6, 12, 17, 18, 19, 24]. Shehory [24] proposed a scalable agent location mechanism that does not require middle agents and protocols for using them. Pancerella et al. [19] presented a CORBA-based distributed object software system for manufacturing organizations. Berry et al. [2] developed software agents in an enterprise information architecture where the agents manage enterprise resources and facilitate user interaction with these resources. Our VPeers work is built upon the CORBA-based service
location system of Cheng and Liu [6]. The major difference between our work and the previous work on agent-based virtual enterprise is that we utilize sophisticated query processing techniques on a peer-to-peer architecture without human intervention, and thus avoid subjective evaluation.

The peer-to-peer paradigm has been studied extensively from business and technical perspectives [1, 7, 8, 9, 10, 11, 13, 22, 26]. Peer-to-peer systems can be centralized (such as Napster, which has a central directory server to which users can submit queries) or decentralized. Of the decentralized systems, some are structured to have a tight coupling between the network topology and the location of data [22], while others, such as FastTrack [11] and Gnutella [13], are unstructured. Our VPeers is an unstructured peer-to-peer system. It has advanced search capabilities for virtual enterprises, which goes beyond the common file sharing applications of the existing peer-to-peer systems. To the best of our knowledge, VPeers is the first of its kind in VMO systems.

The eXtensible Markup Language (XML) is the new standard for information representation and exchange on the Internet. For this reason, we use XML to describe the shared services of a VPeer, and publish the information on the Internet. Thus, a VPeer’s information can be discovered by other Internet users, not limited to those in the VPeers framework with the P2P query mechanism. We take advantage of the results in publishing relational data as XML documents [23] to design our Document Type Definition (DTD) and to implement our XML generator. We have also benefited from the algorithm of decomposing a complex query into several simpler sub-queries [14] in our query processor.

3. The VPeers Framework

We now present the system architecture of our VPeers framework and describe the components in detail.

3.1. Overview of System Architecture

An overview of the framework is shown in Figure 2. A VPeer consists of four sets of components: the server components, which manage and maintain local services and the knowledge about other friend VPeers, and also serve queries from other VPeers; the client components, which facilitate the local user to send out tailored queries; the query processing components which focus on the processing of search queries, and the data management components which are responsible for storing the local service information as well as the “global” information around the VPeer. The following subsections describe these components in more detail.

![System Architecture of a VPeer](image)

Figure 2: System Architecture of a VPeer

3.2. VPeer Server

The server components of a VPeer include a service administration interface, an XML generator and an incoming-query listener.
The service administration interface enables the administrator of this VPeer to manage the local services as well as its knowledge about other VPeers. The interface accepts the administrator’s input and passes the information to the data manager for storage.

The local service information can be sent to the data manager for storage. It can also be published to the network. To utilize the Web services for this purpose, we chose XML as the format of the published information and defined a common DTD for the service information. The task of publishing the information into XML is accomplished by the XML generator.

The query listener is a server component that waits for other VPeers to submit their queries. Upon receiving a query, the query listener will forward the query to the query dispatcher in the query processor and return the result to the querying VPeer.

3.3. VPeer Client

The client components of a VPeer include a service search interface for local users and an outgoing-query sender, indicated by “User Search Interface” and “Query Sender” in Figure 2, respectively.

The service search interface helps the local users to formulate their queries about their target services. The search attributes include service name, type, price, expected time to finish, number of results to return, etc. When a user query is formulated, it is sent from the search interface to the query dispatcher. The query dispatcher will select friend VPeers to answer this query and let the query sender send the query.

Recall that the query listener module of the server has query receiving functionality similar to that of the service search interface, but they differ in that the query listener receives formulated queries and serves other VPeers while the user search interface facilitates local users to formulate their queries and be served at other VPeers.

3.4. Query Processor

The query processor of a VPeer consists of a query dispatcher, a local query evaluator, and a result ranker. It simplifies the service discovery mechanism applied in virtual manufacturing environment. For each query, there is an originating VPeer that initiates the query and there are multiple participating VPeers that participate in answering and forwarding the query.

The query dispatcher is the coordinator module of the query processor. It is responsible for:
(a) receiving queries from the server query listener or the user search interface,
(b) calling the local query evaluator to evaluate a query locally,
(c) selecting friend VPeers for the query sender to send a query to,
(d) calling the result ranker to rank the final results, and
(e) returning the query results to the query listener or the user search interface.

The local query evaluator examines whether there are local services that satisfy a query. It first decomposes the query with multiple predicates into several simpler sub-queries. It then evaluates the sub-queries in an optimized order. For instance, if the service name search predicate of the query is expected to be more selective than the price range predicate, the local query evaluator will first apply the service name search predicate and then the price range predicate. Unlike a traditional database query processor, our local query evaluator interprets range predicates as similarity searches, in which each result of a predicate is marked with a corresponding fitness value to describe how “good” that result is. Finally, the local query evaluator returns the final result of a query with a computed fitness value.

At an originating VPeer, the result ranker takes the results returned from other associated VPeers and sorts the results by their fitness value. In addition, it uses the fitness information to update the friendship information on each friend VPeer that this VPeer maintains.

3.5. Data Manager

The data manager of a VPeer node is responsible for storing the local service information as well as the “global” information on its friend VPeers. It manages these two types of information both in memory and on disk.
Since the local service information is frequently used in local query evaluation and its amount is relatively small compared with a large number of VPeers, we keep the information in the main memory.

The “global” service information is mainly stored in the local database as relational tables, and some frequently used data, such as information of recently interacted friend VPeers, is cached in the memory. The “global” service information is incomplete with respect to the entire VMO, and only corresponds to the local VPeer’s past experience in the VMO. Nevertheless, this information is crucial for selecting friend VPeers when the query dispatcher makes a routing decision.

4. Peer-to-Peer Service Discovery

In this section, we present the peer-to-peer service discovery process across individual components and describe the technical details of our algorithms.

4.1. Overview of the Process

Let us take VMO in Figure 1 as an example to illustrate the major steps in the peer-to-peer service discovery process. VMO consists of four enterprises (Enterprises 2, 3, 4, and 5), and it has many possible service discovery processes. One of the processes is demonstrated in Figure 3. We will discuss the processes in detail by studying three representative scenarios in the following subsection.

![Figure 3: Overview of the service discovery process](image)

4.2. Service Discovery Process

Consider the scenarios in Figure 4. Suppose that a user query is submitted at the search interface of VPeer at Enterprise 2. We introduce two new measures, friendship and TTL (Time-To-Live), for our illustration. We will define these two measures quantitatively in the next subsections. Here, we use them intuitively.
Scenario A. All VPeers that have services to satisfy the query are friends of the originating VPeer at Enterprise 2, and all friends of the originating VPeer have services to satisfy the query.

In this scenario, the originating VPeer dispatches the query to its friend VPeers at Enterprises 3, 4, and 5. The query message that the originating VPeer sends is flagged with a unique query ID, a TTL value, and the basic information of the originating VPeer.

An effective friend VPeer first checks the TTL value of the query message. If the TTL value is zero, the query will not be processed since it has reached its time to live. The friend VPeer then checks the query ID. If it has processed the query before, it will not respond. Otherwise, the friend VPeer evaluates the query locally. Since the friend VPeers in this scenario all have services to satisfy the query, they return their service information to the originating VPeer and stop. Moreover, because all VPeers that have services to satisfy the query are friends of the originating VPeer, the originating VPeer receives information about all available services in the VMO.

Scenario B. Not all VPeers that have services to satisfy the query are friends of the originating VPeer, but all friends of the originating VPeer have services to satisfy the query.

In Scenario B, the originating VPeer dispatches the query to its effective friend VPeers at Enterprises 3 and 5. Similar to scenario A, a friend VPeer checks the TTL value and the query ID of the query, and evaluates the query locally. Because all friends of the originating VPeer have services to satisfy the query, they return the results to the originating VPeer and stop even though the VPeer at Enterprise 4 also can satisfy the query. One implication of this scenario is that there may be non-direct-friend VPeers who can provide the requested service in the VMO, but the originating VPeer will not have a chance to examine them.

This is a tradeoff that we have made between query forwarding efficiency and result completeness. If we made the friend VPeers to forward a query to their friend VPeers in addition to providing their own service information, the originating VPeer could have eventually get information about all available services. However, this would go beyond normal collaborations between VMO members (not only providing services but also referring to other services) and generate a much larger amount of network traffic. Therefore, we choose to let friend VPeers not forward a query if they can answer the query locally.

Scenario C. Not all VPeers that have services to satisfy the query are friends of the originating VPeer, and not all friends of the originating VPeer have services to satisfy the query.

As in scenario B, the originating VPeer dispatches the query to its effective friend VPeers at Enterprises 3 and 5. Similar to scenario B, the friend VPeer at Enterprise 5 returns information about its service and stops. However, the friend VPeer at Enterprise 3 cannot answer the query locally. Therefore, it forwards the query to its friend VPeers at Enterprise 4 and 5 with the original query ID, a decreased TTL value, and the information about the originating VPeer at Enterprise 2. It does not forward the query to the originating VPeer, even though the original VPeer is a friend.

The VPeers at Enterprises 4 and 5 check the TTL value and the query ID before evaluating the query locally. Given that the TTL value is larger than zero, the VPeer at Enterprise 4 evaluates the query locally and sends the information to the originating VPeer at Enterprise 2. In contrast, the VPeer at Enterprise 5 drops the query forwarded by the VPeer at Enterprise 3, because it recognizes that it has processed the query before by checking the query ID. In the end, the originating VPeer luckily acquires information
about all available services in the VMO even though not all services reside on its friend VPeers. This shows the power of peer-to-peer search.

4.3. Query Dispatching

After giving an overview of the peer-to-peer search process in a VMO, we discuss the query dispatching algorithm in detail. We quantitatively define the VPeer friendship concept, and present additional mechanisms for reducing network traffic.

4.3.1. Friendship

Previous research [26] suggests that distributing service functions across a carefully selected subset of nodes yield better performance, availability, and scalability than massively decentralized approaches. In our VPeers framework, we denote the subset of nodes that a VPeer contacts as its friend VPeers.

When an originating VPeer searches for services, it first queries its friend VPeers. If the service is not found at a friend VPeer, the friend VPeer will in turn forward the query to its friend VPeers. The selection of friend VPeers of one VPeer is based on the friendship of these friend VPeers with the specific VPeer. The friendship reflects the capability and potential “help” that one VPeer could provide to another based on historical records.

When a new member enters the VPeers network, we assume that every discovered existing VPeer has an equal chance, thus the same “friendship”, to serve it. On the other hand, the existing VPeers that have discovered the newcomer mark the newcomer as a fresh friend, and will assign the newcomer an appropriate friendship value (which could be a predefined value/function, such as an average friendship). This priority allows newcomer to have the chance of being selected to answer a query for the first time. When a newcomer A satisfies a query for the first time, the querying VPeer B’s record of A’s friendship will be updated. By setting the “fresh friend” mark on A to be false, B stops regarding A as a newcomer. The VPeer updates the friendship records with its interacting with other VPeers gradually. The following paragraphs describe how to obtain and update the friendship.

Given two friend VPeers \( N_i \) and \( N_j \), and the total number \( n \) of friend VPeers that \( N_i \) has, we use the following formula to compute the friendship of \( N_i \) towards \( N_j \) at time \( t \):

\[
\text{friendship}(N_i, N_j) = \begin{cases} 
\frac{\text{provided}_\text{services}(N_i, N_j) \times A_{N_i}(t)}{\sum_{k=1}^{n} \text{provided}_\text{services}(N_i, N_k) \times A_{N_k}(t)}, & n \geq 1, N_j \in \text{(Friend VPeers)} \\
\frac{1}{\text{number of Detected existing VPeers}}, & n = 0, N_j \in \text{(Discovered Existing VPeers)} \\
\text{specific predefined friendship value}, & n \geq 0, N_j \in \text{(Newcomers)} 
\end{cases}
\]

where

\[
A_{N_i}(t) = \begin{cases} 
1 & \text{if } N_i \text{ is available at time } t. \\
0 & \text{otherwise}.
\end{cases}
\]

Friendship of \( N_i \) towards \( N_j \) is quantitatively defined as the ratio of the previous services that \( N_j \) provided to \( N_i \) vs the total services that \( N_j \) received before. Furthermore, we adopt the concepts of fitness (the similarity of a provided service with respect to the corresponding query) and service weight (the weight of a service among all known services) to consider the quality of service.

\[
\text{provided}_\text{services}(N_i, N_j) = \sum_{m=1}^{\text{number of services}(N_i, N_k)} \text{fitness}(m) \times \text{service}_\text{weight}(m).
\]

In Equation (2), \( \text{number of services} \ (N_i, N_k) \) is the total number of services that have been queried by \( N_i \) and provided by \( N_k \). \( \text{service}_\text{weight}(m) \) represents the weight of the service \( m \) among all known services.
Equation (2) quantifies the quality of the provided service with respect to the corresponding query. We further define \( \text{fitness} (m) \) to be a weighted sum of the quality of the service \( m \) with respect to each attribute (for example, service name, price, response time) queried by \( N_i \). The variable \( \text{attribute\_weight}(p) \) shows the importance of attribute \( p \) among all attributes that are queried for a service.

\[
\text{fitness}(m) = \sum_{p=1}^{\text{number of attributes}} \text{fitness}(m)_p \times \text{attribute\_weight}(p).
\]

Finally, the fitness of the service with respect to each queried attribute is defined based on the semantics of the attribute. For example, with respect to the service name attribute, the fitness value is either 1 or 0 (either an exact match or not). For some numeric attributes such as service price, response time, and capacity, we define the fitness of a service with respect to a queried attribute as Equations (4) and (5).

For price and response time,

\[
\text{fitness}(m)_p = \begin{cases} 
1 & \text{if upper\_bound} \geq \text{actual\_value}. \\
\frac{\text{upper\_bound}}{\text{actual\_value}} & \text{otherwise}.
\end{cases}
\]

For capacity,

\[
\text{fitness}(m)_p = \begin{cases} 
1 & \text{if lower\_bound} \leq \text{actual\_value}. \\
\frac{\text{actual\_value}}{\text{lower\_bound}} & \text{otherwise}.
\end{cases}
\]

The variable \( \text{actual\_value} \) is the corresponding attribute value of the provided service, and the variables \( \text{upper\_bound} \) and \( \text{lower\_bound} \) are user expected values in the query. For some attributes such as service price and response time, users expect that the provided service is smaller than the upper bound. For some other attributes such as service capacity, users expect that the provided service is larger than the lower bound.

### 4.3.2. Additional Dispatching Mechanisms

In addition to selecting friend VPeers based on VPeer friendship to forward a query, we have three additional mechanisms for reducing query flooding in a VMO.

The first mechanism is to set a TTL value for each query message. The TTL value is set in terms of the number of network hops. When a query message is forwarded from one VPeer to another, its TTL value is decremented after each hop. However, choosing an appropriate TTL value is not straightforward. If the TTL is too high, the node unnecessarily burdens the network. If the TTL is too low, the node might not find the required object even though it exists in the system [15]. We attempt to utilize existing techniques for estimating and dynamically adjusting the TTL value [28].

The second mechanism is to tag a query ID to each query message. The query ID consists of the originating VPeer’s ID (The CORBA Portable Object Adapter and the Smart Agent guarantee the uniqueness of the VPeer ID) and the originating VPeer’s counter value (monotonously increasing). As a result, no matter how many hops that a query is forwarded within its TTL, a VPeer will terminate if it has seen the query before.

The third mechanism is to tag each query message with information about its originating VPeer. Based on this information, other VPeers will not forward a query to its originating VPeer even though the originating VPeer is a qualified friend VPeer. Moreover, a VPeer that has services to satisfy a query can send the results directly to the originating VPeer without going through the intermediate friend VPeers.

### 4.4. Local Query Evaluation

After a VPeer receives a query dispatched from a friend VPeer, it will check if the query can be satisfied locally. The format of a query statement is shown in Figure 5: it defines the communication between the
VPeers at semantic/knowledge level. The query statement is a conjunction of multiple predicates, each of which is either a string match or a numeric comparison. The service name and the service price predicates are required; all other predicates are optional.

To evaluate such a multi-predicate query efficiently, we order the evaluation of the predicates by their selectivity and build indexes on frequently queried attributes of local service information. The more selective a predicate is, the sooner it is evaluated. Therefore, we can filter out services as soon as possible. For instance, the service name predicate is always evaluated before the service type predicate.

The query statement is a conjunction of multiple predicates, each of which is either a string match or a numeric comparison. The service name and the service price predicates are required; all other predicates are optional.

<table>
<thead>
<tr>
<th>QueryStatement =</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceName</td>
</tr>
<tr>
<td>AND serviceType</td>
</tr>
<tr>
<td>AND serviceCapability</td>
</tr>
<tr>
<td>AND servicePrice</td>
</tr>
<tr>
<td>AND serviceHours</td>
</tr>
<tr>
<td>AND serviceResponseTime</td>
</tr>
<tr>
<td>AND serviceDueTime</td>
</tr>
<tr>
<td>AND filteringPolicy</td>
</tr>
<tr>
<td>AND optimizationPolicy</td>
</tr>
<tr>
<td>AND serviceAnnotation</td>
</tr>
</tbody>
</table>

Figure 5: The Query Statement

Additionally, the local query evaluator calculates the fitness value of each service for each predicate and in turn for the entire query. This is used for the result ranking process.

4.5. Result Ranking

After a participating VPeer evaluates a query locally, or an originating VPeer receives query results from friend VPeers, it will perform ranking on the query results. This is necessary for users to receive quality results especially when the scale of a VMO is large. Moreover, if we only return the results with higher rank, we can reduce the net traffic by filtering results. To avoid excessively long waiting time, the query listener will only collect the returned results within a time constraint.

A participating VPeer ranks its local query results by their fitness value in decreasing order and sends the ranked query results to the originating VPeer. The originating VPeer further ranks the results from multiple friend VPeers by their fitness values. If VPeer $N_k$’s offered service has been accepted and actually used by originating VPeer $N_i$, then $N_i$ will update the $\text{provided services}(N_i, N_k)$ value accordingly.

5. Prototype Implementation

We have implemented a prototype of the VPeers framework using Borland’s JBuilder™ 5.0 with support of Visibroker® 4.5 for Java.

5.1. User Interface

The user interface of the VPeers framework includes a server administration interface and a user search interface. Figure 6 shows a screenshot of the user search interface. This console consists of three panels: the General panel, the Query panel, and the Query Results panel. The General panel shows the general setting and information about the VPeer. The Query panel (shown in the screenshot) provides the search
interface that helps the local users to formulate their queries about their target services. The Query Results panel displays the returned results in a tree-view format to local users.

![Screenshot of User Search Interface](image)

Figure 6: Screenshot of User Search Interface

5.2. Service Discovery Implementation

VPeer must be able to discover the presence of other VPeers and resolve their locations (addresses, protocols, and ports) by names or other identifiers. This is complicated by transient connectivity among VPeers and the lack of address records in the Domain Name Server (DNS).

5.2.1. VPeer Discovery

We name a VPeer by the CORBA Portable Object Adapter (POA) name [21], which in turn consists of an organization code, a serial number, and a system title. The POA name uniquely identifies the existence of a VPeer. We built the name resolution and peer discovery mechanism on top of VisiBroker’s Smart Agent. The Smart Agent acts like a DNS server — a local area network needs to have at least one Smart Agent. Two Smart Agents on different local area networks can detect each other through User Datagram Protocol (UDP) broadcast [3]. The Smart Agent provides an object location service so that VPeers can find one other by their name. Point-to-point UDP connections are used for VPeers to send registration and look-up requests to the Smart Agent.

When a VPeer is activated, it registers with one available Smart Agent in the local network. If a Smart Agent terminates unexpectedly, the VPeers registered with that Smart Agent will discover this event and will automatically register with another available Smart Agent. When one VPeer goes offline, the Smart Agent removes it from its list of available objects.

Each VPeer records both its own registered name and entries of friend VPeers. Each entry of a friend VPeer in the local database contains the VPeer’s name and communication information (address, protocol, and port number). During the query forwarding process between VPeers, the Smart Agent is automatically consulted to locate VPeers by their communication information.
5.2.2. Newcomer Management

VPeers form an overlay network over the Internet. To join it, a newcomer VPeer has to discover a small subset of the active participants. This discovery is done by a constraint flooding search (sending discovery messages by broadcast to every possible node, to query whether it is using the same VPeer version) based on a random graph with TTL mechanism. In our implementation, to avoid overwhelming traffic caused by the flooding approach, the discovery is done by querying the Smart Agents, which act as central repositories and hold the communication information of some participants. Therefore, a VPeer coming with a set of preset Smart Agent addresses creates a more favorable situation. Moreover, the manual operation of adding “friend” is more helpful to a newcomer in VPeers. In this way, the newcomer’s existing-VPeer discovery range is extended and is not a limited-scope query anymore. On the other hand, based on the VisiBroker's Smart Agent peer discovery mechanism, we have designed the system so that the administrators of the discovered existing VPeers will be informed and add the newcomer to the friend list upon the newcomer’s joining. This design, together with the priority assigned to the newcomer, guarantees that the newcomer has a chance to be selected to answer a query for the first time.

The case of isolated groups of VPeers becomes more complicated. To simplify the processing, we utilize manual operation for administrator(s) in the separated groups to add friends. Consequently, the separated groups of VPeers also have the chance to communicate with each other eventually.

5.3. Data Manager Implementation

We chose MySQL DBMS (version 3.23.52-nt) to manage a VPeer’s local data persistently. We also developed an in-memory object manager to manage the cached local data in memory.

There are four relational tables in the local database: LocalVPeer, OriginatedQueries, SatisfiedServices, and FriendVPeers. Table LocalVPeer records the major elements of the local service information. Table OriginatedQueries records the detailed information of queries that are originated from the local VPeer. Table SatisfiedServices records the historical information about the services that friend VPeers have provided to serve this VPeer’s originated queries, including the query ID, the serving VPeers’ name, the fitness of the provided service to the query, and other information. This information helps to determine the friendship of other VPeers toward this VPeer. Finally, Table FriendVPeers records the information of the friend VPeers, includes the friend VPeer’s name, IP address and friendship with the local VPeer. This information helps the VPeer to determine how to dispatch a query. Note that part of the information in the database is loaded into memory as objects to speed up query processing and dispatching.

In addition to managing the local data, the data manager also helps the XML generator to publish its local service information into XML. An XML document of the local service information consists of general information and operation information. The general information describes the services, such as the name, type, capability, price, and response time of the services. The operation information describes the VPeer’s entity interfaces that can be implemented by the remote VPeers and provides guidance about how to operate the shared services (regarded as CORBA objects) remotely, such as the function module name, IDL interface name, CORBA method name, parameter list, and others. Figure 7 shows the DTD of the published XML data.
<!ELEMENT VPeer (Service+)>
<!ELEMENT Service (GeneralDescription, OperationDescription)>
<!ELEMENT GeneralDescription (ServiceName, ServiceType, ServiceCapability, ServicePrice, ServeTime, ServiceResponseTime)>
<!ELEMENT ServiceName (#PCDATA)>
<!ELEMENT ServiceType (#PCDATA)>
<!ELEMENT ServiceCapability (#PCDATA)>
<!ELEMENT ServicePrice (#PCDATA)>
<!ATTLIST ServicePrice unit CDATA #REQUIRED>
<!ELEMENT ServeTime (#PCDATA)>
<!ATTLIST ServeTime unit CDATA #REQUIRED>
<!ELEMENT ServiceResponseTime (#PCDATA)>
<!ELEMENT OperationDescription (moduleName, idlInterfaceName, CORBAMethodName, ParamList, returnValue_Type, despOfTheMethod,)>
<!ELEMENT moduleName (#PCDATA)>
<!ELEMENT idlInterfaceName (#PCDATA)>
<!ELEMENT CORBAMethodName (#PCDATA)>
<!ELEMENT ParamList (param+)>
<!ELEMENT param (name, direction, type, value, desp)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT direction (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT desp (#PCDATA)>
<!ELEMENT returnValue_Type (#PCDATA)>
<!ELEMENT despOfTheMethod (#PCDATA)>

Figure 7: DTD of Published Local Service Data

Note that we use both relational databases and XML at each VPeer. The relational database is used to store the information of the local VPeer resources, and the XML together with its DTD is to publish this information. When a VPeer attempts to find resources in the VMN, it may not know the schema of the resource descriptions and therefore cannot form a valid query statement. For instance, if it does not know there is a "servicePrice" attribute, it will not query on the price. Having the schema information of the resource descriptions, a VPeer can form a valid query statement and issue it efficiently. Furthermore, the published information of a VPeer can help other VPeers to direct relevant queries to it. Finally, when a VPeer evaluates a query locally, it uses the local relational database for efficient processing.

6. Experiments

We describe our experiment setup and present experimental results in this section.

6.1. Experiment Setup

We ran different number of VPeers on two Intel Pentium II™ PCs, which run Windows 2000 professional on a local area network, to study the performance issues. We identified three factors, namely the query length, the VPeer size (in terms of number of local services), and the number of friend VPeers per VPeer, to be examined in our experiments. This will present some insights for us to design the VPeers framework for the real virtual manufacturing environment, where the physical network should be more complex. The three factors and their values are summarized in Table 1.
6.2. Results

We first examine the effect of the VPeer size on the \textit{local search time} (from a VPeer receiving a query to it finishing searching its local services). We then test the effect of the query length on the local search time as well as the \textit{response time} (equal to the sum of the local search time and the network message transfer time). Finally, we examine the effect of the number of friend VPeers on the \textit{query dispatching time} of a VPeer (the time for a VPeer to dispatch a query to all its friend VPeers).

When the number of friend VPeers of a VPeer under test is about 10 and the query length is about 100 Bytes, our experimental results show that the larger the VPeer size, the longer the local search time. Moreover, the rate of increase is higher as the size become larger. This suggests that it is important to choose an efficient way to represent the local service information and to evaluate local queries. We observe the impact of varying the query length on the average response time and the local search time. The VPeer under examination has 100 services (i.e., it is of moderate size, and can offer 100 types of service to other VPeers, e.g., CNC machining, rapid prototyping, etc.), and it has about 10 friend VPeers. We find that the query length has no significant effect on either the local search time or the total response time. This is because that the query length is relatively small compared with the query processing capability of a VPeer and the network bandwidth of the local area network.

We also examine the relationship between the query dispatching time and the number of friend VPeers that a VPeer contacts. The VPeer under test has 100 services, and the query length is about 10 Bytes. The results show that the time a VPeer takes for dispatching a query is proportional to the number of its friend VPeers. Therefore, it is crucial to choose a small subset of suitable friend VPeers for query dispatching in order to improve the system response time.

7. Conclusion

We have proposed VPeers as a peer-to-peer service search framework for virtual manufacturing organizations. To achieve efficient operation over the Internet and to be cost-effective, the presented VPeer framework is applicable for services that are fairly standardized and can be defined accurately, and the processes can be allocated independently. The potentially promising application areas include rapid prototyping; CAD/CAM data translation; product development; distributed manufacturing with solid and integrated software environment, etc. However, services that require specialized technology and skills to interoperate or have sequence dependencies, would then be out of the VPeers’ application scope and require more sophisticated solution.

We have implemented a prototype of the VPeers framework on top of the Smart Agent. The prototype works well in the laboratory environment. The features of the VPeers include:

- The system is completely decentralized. This architecture allows for a large number of VMO members to participate whenever and wherever they want without coordination from a central server.
- The peer-to-peer search paradigm allows the VMO members to find shared resources using complex queries on service attributes, including both numeric attributes and textual attributes.
- The system defines a common format of local service information in addition to a common format of service queries. Furthermore, the local service information are published in XML documents so that other VPeers and potentially other Internet users can query about the shared resources easily.
- The VPeer friendship concept and the query result ranking mechanism enable users to obtain quality search results from a possibly vast number of candidates in a reasonable amount of time.

\begin{table}[h]
\centering
\caption{Factors and Their Values in Experiments}
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Query Length} & \textbf{~10 Bytes} & \textbf{~100 Bytes} & \textbf{~1000 Bytes} \\
\hline
\textbf{VPeer Size} & 14.1KB XML (~10 Services) & 149KB XML (~100 Services) & 1460KB XML (~1000 Services) \\
\hline
\textbf{#Friend VPeers} & 10 & 100 & 1000 \\
\hline
\end{tabular}
\end{table}
• By storing, caching, and indexing the local service information and the historical search results, the system can find target services in an efficient way.

To better understand how VPeers will develop over time, the proposed framework requires further improvement and fine tuning to be stable and robust. We need to track the VPeers’ evolution in terms of connected nodes, number of shared services and total amount of services offered over a period. Simulation can be performed to study the dynamics, inherent behavior and key characteristics (e.g., self-organization and the friendship notion) of the VPeers architecture in this domain. In addition, physical information of the VPeers (such as locations, distances from a requester) may be of importance in the selection of friends and need to be discussed in the friendship formulae, to improve the quality of service. Another direction for future study is to investigate the optimal peer selection problems with the integration of bidding and bargaining functions. In addition to studying more performance issues in our prototype, we also expect to accomplish a few practical VMO applications of this prototype.

Acknowledgment This work has been supported in part by Hong Kong Research Council through grant N_HKUST023/00. We thank the referees for their valuable comments and suggestions that have helped to improve the content and style of the manuscript.

References


Anbo Xiang received his B.Sci. in Physics and M.Sc. in Electrical Engineering from Huazhong Normal University. He received his Ph.D. in Industrial Engineering and Engineering Management from Hong Kong University of Science and Technology in 2006. He is currently a researcher working in Enterprise Research Institute at Development Research Center of the State Council, China. His research interests include operation management and service supply chain management.

Liming Liu received his B.Eng. in Shipbuilding Engineering and M.Sc. in Management from Huazhong University of Science and Technology (HUST). He received his Ph.D. in operations research from University of Toronto. He taught at HUST, Toronto and Hong Kong University of Science and Technology before joining Hong Kong Polytechnic University in 2006, where he is now a professor in the Department of Logistics. His research interests include logistics and supply chain management, stochastic models, port operations and economics, networks and Internet analysis.

Qiong Luo is an assistant professor in the Computer Science Department, Hong Kong University of Science and Technology. Her research interests are database systems, with a focus on data management and analysis techniques related to network applications. Qiong received her Ph.D. in Computer Sciences from University of Wisconsin-Madison in 2002. Her Ph.D. dissertation was on Caching for Web-based Database Applications. Qiong received her B.S. and M.S. in Computer Sciences from Beijing (Peking) University, China.
http://www.cs.ust.hk/~luo