PETERPAN: A Middleware for Deploying Web Services on Distributed Computing System using Peer-to-Peer Paradigm

Sissades Tongsima, Shobhna Srivastava, Ekasit Kjiisipongse, Kittinarong Laongwaree
Apirak Panatkool, Sithichai Laoveerakul, Weerachai Anotaipaiboon

Cluster Computing and Grid Technology Section
National Electronics and Computer Technology Center
112 Paholyothin Rd., Klong 1, Klong Luang, Pathumthani 12120, THAILAND

Abstract

The Internet has formed the largest computing resource and data repository on this planet. The advent of it greatly benefited us in creating an important building block of the collaborative environment. There are some problems which impedes the success of creating it effectively. From the computing resources view, it is very difficult to permit users to transparently locate and collectively acquire the underlying resources, due to the heterogeneous nature of resources on the Internet. The majority of practical applications on the Internet are still restricted to Web applications which only allow users to interact with Web servers. There are only few applications that are treated for the aforementioned heterogeneity issue. Web services (WS), being one of them, is the trend of the new-era business and scientific applications which support the interoperability of coordinating various implementations. Nevertheless, WS is not ready yet to be deployed on a larger scale.

To address the problems, we propose the software layer (middleware) called PETERPAN, which acts as a middle man to interacts with distributed applications on one end and the Internet resources on the other end. This middleware internally promotes the deployment of Web Services on a larger scale by using the Web Service flow control engine to manage the integration of multiple Web Services. Furthermore, to communicate with other resources in a decentralized fashion, PETERPAN adopts Gnutella, a popular peer-to-peer (P2P) protocol. This decentralized paradigm has many advantages over the centralized paradigm when dealing with distributed systems. This paper exhibits the design and implementation of PETERPAN architecture that supports the above issues. For the deployment of the prototype of this design, a PETERPAN module must be installed on each peer where shared resources such as Web Services, data etc., must be registered to itself. Finally, we discuss about possible distributed applications of this middleware to demonstrate the potential of making use of this work to solve existing problems in our society.

1 Introduction

Emergence of the Internet has made it possible to interconnect millions of computer-like devices ranging from powerful supercomputers, computer farms, and even handheld device such as PDAs
and mobile phones. According to the current trend, both computing power and interconnection network speed are increasing each year by the support from both industries and academia. Imagine how wonderful it will be to be able to draw computing power from these devices in the same way we are using electrical power—not knowing from which the (computing) power is generated. The Internet and the world wide web (WWW) have now become an important part of many businesses just by allowing users to primitively access information located on different sites via web applications. In fact, the geographical distributed structure of the Internet provides a perfect environment for distributed computing. With these technologies, computing paradigm is no longer dominated by the single computing environment but rather a distributed one. Several attempts have been made to utilize these Internet computing resources by means of distributed computing such as SETI@home and FOLDING@home (SETI@home 2002; FOLDING@home 2002; Korpela et al. 2000). One of the significant attempts in utilizing the Internet distributed environment is to define the standard middleware to create a powerful virtual computing infrastructure called the Grid. The Globus project (Foster and Kesselman 1999) led by Foster and others strives to create a middleware that can help applications utilize distributed resources on the Internet.

Nevertheless, there are many difficulties that need to be resolved before achieving the previous goal. The prominent challenges to bring up such a system include (but not limited to) addressing heterogeneity of networked resources, promoting concurrency of running applications, increasing security, improving scalability of the system, managing resource failure, and providing transparent access for users. Many solutions have been proposed to tackle these challenges from various viewpoints. In this work, we concentrate on the peer-to-peer (P2P) computing paradigm (Milojicic et al. 2002; Aberer et al. 2002) to help us draw the computing power as well as share content on the Grid. Intuitively speaking, the interaction of any two computers on a network is regarded a P2P communication, if each system can be both client and server when required. In other words, P2P explores distributed resources on the Grid and utilizes them in decentralized manner.

The success of P2P have been shown over the past few years from many practical applications which include ICQ, Gabber, Napster, Gnutella, SETI@home and many others (Milojicic et al. 2002). These applications can be then categorized in three groups: Instant Messaging (IM), Content Sharing, and Distributed Computing. However, these applications, though created by using P2P notion, cannot work collaboratively in a distributed environment. Based on this peer-to-peer protocol, we propose a middleware called, PETERPAN (PTP), to communicate between heterogeneous resources and other applications present in the form of Web services, which have now set the standard for future distributed applications. The computing resources and data that is to be efficiently utilized by the middleware are collectively refer to as PTP-resources. In fact, the existence of PTP layer is for applications to access PTP-resources in the peer-to-peer style.

From the application point of view, we concentrate on web service technology which promotes creation of open distributed systems. Using standard protocols like XML and SOAP (W3-Consortium 2000a; Pallos 2001), Web services are quickly gaining popularity in making distributed applications. In this Web services perspective, an application can be built by integrating multiple services together making a more complex service. In order to integrate these services, one must be able to locate and acquire specified services. The Universal Description Discovery Integration (UDDI) compliments web services by providing the functionality to search for web services located on different sites. Existing UDDI technology uses central server (operators) to store pointers to “registered Web services” which may be located elsewhere. However, using centralized approach has many drawbacks as it appeared in many research studies.

The proposed middleware can help locate Web services described above in a decentralized fashion. Unlike the centralized UDDI service (UDDI 2000), we extend the P2P search to handle
the Web services discovery. Each PE T ER P AN peer can register Web services to itself. Peers permit other peers to search for the target Web service(s) without going through the UDDI server. In addition to this PE T ER P AN also support the decentralized P2P computing paradigm. To demonstrate the highlights of this work, the list of PE T ER P AN features are as follows:

1. It operates as a middleware which interacts with distributed applications on one end and the distributed computing resources on the other end.

2. Fully supports Web service technologies including other related concepts such as WSDL, WSFL, SOAP, UDDI etc.

3. It provides the WSFL engine which supports the creation and monitoring of complex Web services.

4. It promotes creation of collaborative environments by providing the set of tools to manage multiple resources such as search and allocation of resources in the PE T ER P AN network.

5. It uses Gnutella, the decentralized P2P protocol, to talk to other peers and promote other decentralized operations.

6. It assumes no sole peer responsible for maintaining the existence of PE T ER P AN network. That is the functioning or malfunctioning of a peer does not affect the entire network; thus increasing its availability and robustness.

7. Even though, at the time of writing this document, accessing security is not a part of this implementation, we have already planned to include in our implementation a de-facto standard security approach such as certification authority, encryption and others to keep the system secure.

The detail of this work is organized as follows. The next section discusses about various related technologies that help us build this middleware which include Web services and Gnutella. Section 3 describes the architecture of PE T ER P AN and how each component interacts with one another. Section 4 touches on the possible usage of PE T ER P AN. Finally, Section 5 draws the conclusion of this work as well as identifying the future research work.

## 2 Preliminaries

Many technologies are put together to create PE T ER P AN middleware. This section presents these technologies from the application view that PE T ER P AN support to network (protocol) and resource views.

### 2.1 Web Services and Companions

Web services are collection of program functions that are available on the Internet and use message passing whose content is described by an XML message (W3-Consortium 2001a; W3-Consortium 2001b; W3-Consortium 2000a; Bloomberg 2001). A Web service is a fully functional program which can be integrated with other Web services to create a more complex distributed application. Many people have extended this concept by including other companion technologies, such as WSDL, UDDI and WSFL (Bloomberg 2001; UDDI 2000; Leymann 2001). The Web Service
Description Language (WSDL) describes all interface definitions of the Web service in a structural way by using a common XML grammar. Universal Description and Discovery Interface (UDDI) is another technology that provides simple lookup mechanism for interested parties to locate Web services and the corresponding interfaces.

Creating a Web service is not very difficult since there are some automatic tools, such as GLUE from Mind Electric company (Electric 2003), that can facilitate us. Figure 1 demonstrates a simple Web service written in java called myService. This code converts any string to the corresponding uppercase string. We use GLUE (GLUE 2002) to create the corresponding WSDL as shown in Figure 2. The input and output interfaces of this program will be described in this WSDL file. This Web service can be ready to use by calling through the GLUE APIs while invoking myService. This is merely a call to web server via HTTP protocol. Such a call contains SOAP information which will then be passed to the SOAP engine to process. This myService code is then executed by using information in the SOAP message obtained from the call. The result can be expressed in SOAP format and sent back to the caller.

```java
public class MyService {
    public String toUpperCase(String string) {
        return string.toUpperCase();
    }
}
```

```java
Import electric.registry.Registry;
Import electric.server.http.HTTP;

public class Publish {
    public static void main (String[] args) throws Exception {
        // Start a web server on port 8004; accept messages via glue
        HTTP.startup("http://localhost:8004/glue");
        Registry.publish("MyService", new MyService());
    }
}
```

Figure 1: Very simple java code demonstrating the simplicity of Web Services

GLUE provides a set of Web Service APIs and the execution engine that understands the remote invocation via SOAP protocol. The above example is very much like how CGI (Common Gateway Interface) operates (W3-Consortium 2002). Lets focus on the WSDL which was obtained from GLUE (see Figure 2). This WSDL contains important characteristics such as service name and its arguments of each Web service which can be used to invoke the service. Notice that when invoking this service, we must provide the URL of the service as well as the service parameters. However, there are many Web services located on various systems on the Internet. Here we have a way to locate where the web services reside. UDDI was introduced to the web service community to do that job by putting the meta-data of Web services in the UDDI server.

The client/server model of UDDI creates the classic bottleneck phenomenon where resources are stored in a small number of sites and there are many clients connecting to them. We need a better approach to provide continuous and reliable access to those resources. The current UDDI
<?xml version="1.0" encoding="UTF-8" ?>
<!-- generated by GLUE on Thu May 02 14:47:26 -->
<definitions name="MyService" targetNameSpace="http://www.themindelectric.com/wsd1/MyService/">
    xmlns:tns="http://www.themindelectric.com/wsd1/MyService/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tme="http://www.themindelectric.com"

    <message name="toUpperCase0In">
        <part name="arg0" type="xsd:string" />
    </message>

    <message name="toUpperCase0Out">
        <part name="Result" type="xsd:string" />
    </message>

    <portType name="MyService">
        <operation name="toUpperCase" parameterOrder="arg0">
            <input name="toUpperCase0In" message="tns:toUpperCase0In" />
            <output name="toUpperCase0Out" message="tns:toUpperCase0Out" />
        </operation>
    </portType>

    <binding name="MyService" type="tns:MyService">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="toUpperCase">
            <soap:operation soapAction="toUpperCase" style="rpc" />
            <input name="toUpperCase0Out">
                <soap:body use="encoded" namespace="http://tempuri.org/MyService" encodingStyle="http://schemas.xmlsoap.org/soap/encoding" />
            </input>
            <output name="toUpperCase0Out">
                <soap:body use="encoded" namespace="http://tempuri.org/MyService" encodingStyle="http://schemas.xmlsoap.org/soap/encoding" />
            </output>
        </operation>
    </binding>

    <service name="MyService">
        <documentation>MyService web service</documentation>
        <port name="MyService" binding="tns:MyService">
            <soap:address location="http://10.22.35.35:8004/glue/MyService" />
        </port>
    </service>
</definitions>

Figure 2: WSDL of MyService in Figure 1 automatically generated by GLUE
addresses the disadvantages of its centralized approach by replicating the entire information and put them on different operators. Three major operators, namely IBM, Microsoft, and ARIBA provide public UDDI services. For experimental and private uses, there are also other private UDDI operators such as *x-methods* and *serviceforge* that help locate Web services (Bloomberg 2001). However, using replication and caching can only superficially remedy the problem. When the number of UDDI users increases, more replication sites must be introduced; thus prompting with a data consistency problem.

Note also that the standard Web services can efficiently communicate to one another via SOAP (Pallos 2001; W3-Consortium 2000b), we can actually create a more complex application by integrating multiple services together. The sequence or flow of these Web services can be represented by using Web Service Flow Language (WSFL). For example, we can integrate two commercial Web services, namely Google search (Google 2002) and MSthumbnail by Microsoft. In this example, we create a more user friendly Internet search tool that has all Google features and we will also be able to see the thumbnail images of the resulting URLs (instead of getting the plain text listing of the search results). Figure 3 depicts the integration of the two Web services. The google Web service is invoked remotely and the resulting URLs are then sent to MSthumbnail for displaying. Note here that there are thousands of usable business services available today on the Internet. These services are already registered to both public and private UDDIs discussed previously.

![Figure 3: Example of MSthumbnail + Google Web services](image)

With all the promising features of Web services, there are some drawbacks of the companion technologies that undermine its deployment. Consider UDDI with its client/server model. UDDI takes the role of a manager to maintain all of its registered Web services while the register-client has the responsibility of updating the information to the UDDI. This is a time consuming process and does not guarantee whether search-client can get the most updated information. Furthermore, the only information given by UDDI is location of the registered Web services. Hence, it is very difficult to choose an appropriate service.

The standard UDDI which supports the Web service deployment was created with a lot of room for improvement. The significant shortcomings of the original design can be listed as follows:

1. First, if UDDI server fails, all the registered services will be inaccessible.
2. With only UDDI, we cannot continuously control the flow of running multiple services.
3. In addition, for a normal user, registering to UDDI seems complex and users need to do it manually every time you create a Web service.
4. Finally, UDDI server does not monitor the availability of the computing resources which contains the underlying Web services. Consequently, it does not guarantee if the registered Web services would be readily executable.

2.2 Decentralized Peer-to-Peer Solution

Majority of communication model nowadays is based upon the client/server architecture. A server provides certain services to client(s) via specific communication protocols such as File Transfer Protocol (FTP), Simple Mail Transport Protocol (SMTP) and others. More than 90% of computations takes place on the server side leaving the client free from carrying heavy processing load. The problem arises when the number of clients reaches the limit where the server is not capable of handling anymore request.

Unlike the centralized approach the peer-to-peer (P2P) model highly supports the idea that every computer on the highly interconnected network can assume either client or server role. In other words, at one time the computers which are highly interconnected in the P2P network can provide services to others (being servers) or requesting services from others (being clients). Interestingly, due to the decentralized nature, these machines do not need to permanently be available at all time. They can still perform the normal operations by asking other on-line computers to act on behalf of the offline ones. From this observation, we note that the proposed system utilizes the simple P2P protocol in order to avoid several shortcomings that come with the counterpart centralized architecture. P2P can maximize the use of resources from computers (peers) connected to the P2P network. It gives services to resources with high availability at the lower cost than the centralized approach.

The first prominent problem of the client/server model is the fact that it has a single point of failure. As an example, the traditional UDDI server which we discussed previously exclusively assumes the server role. Even though this server can be replicated and have multiple servers to uninterruptedly provide the service when the server is down, this type of system is still prone to being attacked by hackers, e.g., a hacker can launch a distributed denial of service (DDOS) attack which will clog up the entire system services. This is the nature of any centralized system where one or several real IP addresses must be provided becoming the visible targets to any hacker. Unlike the client/server model, the P2P system does not have single place to keep all the important information. Every machine in the P2P network can be equally treated. That is even though several systems are unavailable, the whole P2P network will be able to maintain its services. Apparently, the service in decentralized P2P form may not be able to give the full service since part of the data is unreachable. It will, however, be more devastating when losing the single server.

Gnutella Protocol

At present time, there are many open standard and proprietary P2P protocols with major drive in development of various protocol implementations from business. The eminent thrust is mainly from file-sharing idea which allow users from geographically distributed locations to share music files such as the work of Gnutella, Napster, Limewire and many others (Gnutella 2002; Limewire 2002; Napster 2002). The participating peers of these group are from people who share the same interest in downloading musics. Even though they all are based on P2P, they contain some differences in their protocol designs. Napster has the centralized server which contains the pointers to the systems that contain the target music. Due to this centralized server notion, the system was tracked down and forced to shutdown it service. Right after Napster was shutdown, Gnutella was
introduced with the decentralized concept in mind; therefore this makes it very difficult to track
down who originates the query since it operates in a highly decentralized manner. There are several
file-sharing programs to do the file sharing include BearShare, GnuCellus, LimeWire, Morpheus,
Phex, and others (KaZaA 2002; Freenet 2002; Limewire 2002). Its popularity stems from the two
facts: providing level of anonymity to users and it is used to share popular multimedia files.

Gnutella (Gnutella 2002; Jovanovic, Annexstein, and Berman 2001) protocol was originally
created by AOL’s Nullsoft division to promote music sharing community while the notorious NAP-
STER was about to be shutdown. The goal of Gnutella protocol is to offer purely decentralized
way of file sharing. Unlike other P2P file sharing solutions, Gnutella is just a communication pro-
tocol where many developers can use in their file sharing software. This protocol does not need a
central server to control how information is routed. It operates by using broadcast routing protocol
to route messages to their destination in this system. A host which connects to the Gnutella system
is called “Servent” (server/client) while the term “Gnet” is used to describe the Gnutella network
(group of servents). Basic messages in Gnutella protocol can be described as follows:

**Ping**  Gnutella uses this message to probe for availability of other servents.

**Pong**  Once the ping message is sent out the pong message will be used by the recipient to response
for the ping.

**Query**  Sending a search request to other servents

**QueryHit**  If the servents have the targeted file, they will response with QueryHit message.

**Push**  If the file is stored at a servent behind firewall, a typical user might not be able to download
this file. The push message is introduced to push the file through this firewall.

In order to describe how Gnutella works, we assume the scenario where there is a servent
(seeker) to search for a file of a specific type. The seeker has no ideas the whereabout of this
song in the network. According to Gnutella, the seeker will initiate the search by sending the Ping
message to other servents which directly connect to it (this is the bootstrapping method to get the
search rolling). Upon the reception these servents send back with the Pong messages. This ping-
pong session will last for a certain number of traversing strides known as the time to live (t.t.1).
By using the ping-pong operation the seeker can build up a connection list which can be then used
in opening the socket connections among peers. At this stage, a Gnutella implementation must
choose which connections it must open. Once the connections are opened, the file discovery can
be launched where the message query will be sent out to the existing connections. If there exists
such a song in one of peers in Gnet, any peer that has the song will send the QueryHit message
back to the seeker. In the same fashion, all these peers including the ones that contain this song will
forward this search query to their peers. Figure 4 depicts how Gnutella operates in an unorganized
network.

There are two counters, namely t.t.1 and hop, that will be updated each time the query message
gets forwarded. The t.t.1 is set to control the number of Servents that each message should be
viewed by. At first this counter will be set to one value and will get decremented each time the
message arrives a Servent. Should this number reaches zero, that message will not be further
forwarded. Since the t.t.1 value may be incorrectly updated, Gnutella also uses hop to measure the
number of machines that explore the request message. The hop number helps detect if the above
problem occurs by checking that t.t.1i + hop = t.t.10. After discovering who has the song, the
seeker can communicate with the corresponding site(s) directly (peer-to-peer) in order to download
the file. This complete the P2P searching process.
3 **PETERPAN Architecture**

In this section we will discuss the design and architecture of PETERPAN. Before beginning the design of this middleware we have kept a few things in mind. First is, that PETERPAN user will be on multiple platforms. And second is, that when we implement it, the development of individual component should not interfere with the other components. To achieve platform independence we have used java (Deitel and Deitel 1999) as the language of coding. In addition in order to reduce inter-component dependencies we have used queues as communication medium between components.

After this preliminary design, the stage of actually identifying the components and defining their functionalities must be performed. To accomplish this we define PETERPAN as:

“It is a middleware that facilitates the development of distributed application using peer-to-peer paradigm. Also it promotes the usage of web services technology.”

This definition raises the following questions to be answered:

1. What sort of applications are we going to support?
2. What are the requirements of these applications?
3. How are we going to fulfill them?
4. And of course how to better support Web Services?
5. What protocol are we going to use to implement peer-to-peer?

Before we begin to answer these questions, consider Figure 5. This figure presents the architecture of the middleware which derives from the above definition.

To the best of our knowledge, the three main types of P2P applications are Instant Messaging (IM), File Sharing (FS) and Distributed Computing (DC). These categories lead to the types of applications that we are going to support which answers our first question. The specific applications that we intend to support are Knowledge Management (KM), collaborative applications, content
sharing, and B2B distributed applications. This results in the first layer of our architecture, the application layer.

There are three major requirements of these applications. That is to discover the peers or the data that is to be used. Furthermore, it needs to exchange the messages with other peers. Finally to support the applications from the distributed computing type, it needs to be able to execute. These requirements are satisfied in the next layer, namely the API layer. The looking, talking and acting parts of the API layer support the three aforementioned requirements. The purpose of this layer is to shield the application layer from the complexities of the layers to follow. Thus, the applications can be developed by anyone who is unaware of distributed systems.

![Diagram showing the component layered stack in PETERPAN design](image)

The answer to the next three questions forms the core of PETERPAN. That means the core of PETERPAN will fulfill the requirements of the API, enhance the performance of Web services and work on a peer-to-peer paradigm. To fulfill the requirements, we have the fourth layer, the internal and external service layer. Recall the three components from the API layer and check what services that they require. When we are looking for peers a simple message from the P2P protocol will give us the required results. However when we are looking for content (file), we have to search our own repository as well as the P2P network. We need to handle the results thus produced which is done by the search service. The next part of the API talking only requires message sending, which is done by the P2P protocol used. The most basic way to implement acting is to download and execute it locally. This results in the download service. A point to remember is that this internal service layer is flexible so that it can incorporate more services as per the requirements.

Since we have multiple services we need to be able to manage them so that they can be used appropriately. Thus we introduce the third layer, namely the service manager layer. This layer is also the answer to the question—how to better support Web services? This layer composes of two modules, the Register and the Flow Control components.

**Register component**

The register component manages the database of all the files and data we share over the P2P network. The most important functionality of this component, however, is to register all the services
that are the part of the service layer, including the Web services. Since the Web services are registered to PE TER PAN, they can be discovered in decentralized manner, unlike the traditional UDDI Web service discovery. The Decentralized discovery enhances the service availability and system robustness as discussed in Section 2.2. The register module was designed with the following keys:

- It must store description of all PTP-resources which include data, file, computing resources and some other network resources.
- The underlying description should be expressed in standard format, XML, in order to be widely acceptable.
- The benefit of using XML grants the ability to extend the resource descriptions to cover all ranges of resources including Web services.

We create the register component by defining an XML meta-data as a medium tool to be used when registering resource to and querying information from it. This meta-data describes all PE TER PAN resources. Unlike the meta-data found in WSDL, our meta-data is resource-oriented not the service-oriented. In other words, the service-oriented data only concerns about how the service will be executed. The resource-oriented approach is similar to object-oriented paradigm where the property and behavior of each object are analogous to properties and services of the corresponding resource. Figure 6 presents the skeleton of our resource-oriented meta-data.

```
<ResourceDescription>
  <property_1> ... </property_1>
  <property_2> ... </property_2>
  ...
  <property_n> ... </property_n>

  <service_1> ... </service_1>
  <service_2> ... </service_2>
  ...
  <service_n> ... </service_n>
</ResourceDescription>
```

Figure 6: Resource-Oriented Meta-data for register component

**Flow control component**

The Web service flow control component (FCC) controls sequence of web services execution. That is it will generate WSFL for the Web services. This module also connects to the GUI, which allows Web service developers to interact directly with the Web service flow control engine. Figure 7 portrays the interface of the FCC engine. The visual editor in this figure helps determine Web services’ properties and their connections. Users can also add conditions to create more complex sequence between services. Output of one Web service can be easily redirected to be an input of
another Web service. Finally, after creating flow of services execution, the final result which is also a service and can be executed by any Web service engine.

The proposed FCC and its interface were inspired by the work of IBM Web service flow language or WSFL (IBM 2002) with an extension on the in/out data flow (message tag). That is the tradition WSFL only specifies the data flow by using simple message tag which must be compatible on both input and output ends, i.e., the output message from one service can only be input of the other if the both have the same structure. In order to promote the re-usability and integration of different Web services, it is not likely to have two fully compatible input/output services available. Put another way, part of data from various Web services can be used as input to another Web services.

The visual editor from Figure 7 depicts the mock up scenario when 6 Web services are working together to sell processor products from either Intel or AMD warehouse. The flow starts with the STORE service which directly exchange information with the STOCK service, perhaps for checking the stock pricing of the two chip-makers. The purchasing decision will be made in accordance with the results obtained from STOCK. Then, the condition vertex called Price condition is used to decide the next service whether it is going to be AMD or INTEL service. The selected service will perform its operation(s) and pass the results to the next service in this flow, the CALCULATEPRICE service, to mark the product price. Finally, the SELL service will probably put the product with correct pricing on display and waiting for customers to order. This scenario reflects the real situation where integration of partially compatible services is possible.

Figure 7: Graphical user interface of the flow control engine

To clarify the contribution we stated above, consider the diagrams in Figure 7. They present how data, called message in WSDL, is transfered from one producer WS to the corresponding consumer WS. The top diagram exhibits the traditional IBM WSFL message transfer where the entire object is moved to the consumer. It is the consumer responsibility to disect the data object and select the right part to be used. We found that this presents an overhead in transfering information. The proposed flow control component addresses this inefficiency by incorporating the part selection process in the output flow language by extending IBM WSFL (as shown in the bottom diagram of Figure 7). Particularly, unlike IBM’s where two fully matched I/O are required in the
integration, this extended WSFL allows integration of services with partial compatible I/O to be expressed by our extended WSFL.

Figure 8 presents two fragments of WSFL which focus on the underlying message portion. The top one displays the traditional WSFL which only roughly describe the whole message object. Unlike traditional WSFL, the bottom fragment shows that the proposed WSFL can also describe parts of the message object (see the tag `<part> </part>`). This helps minimize the amount of message being sent between two Web services.

```xml
<dataLink source="processPO" target="acceptShipmentRequest">
<map sourceMessage="anINVandSR" targetMessage="anSR"/>
</dataLink>

<DataLink Name="generateOutputMap-vector2svg" source="generateOutputMap" target="vector2svg"/>
<part value="Result" type="string" arg="arg0"/>
</DataLink>
```

Figure 8: Comparison of the traditional IBM WSFL (top) and PETER PAN WSFL (bottom)

The visual editor also comes with two sets of component panel, namely flow and service components. Users can create a flow from dragging components in the tool panel and dropping them in the editor canvas. The newly created flow must contain start and stop nodes. The list of services found in this tool panel are obtained from querying to the PETER PAN "search" for Web service resource described previously. The output of this flow control engine is a WSFL of a complex service which contains the flow of multiple Web services. This WSFL can be then converted, using standard tool provided by GLUE to java which can be then transformed to a new WSDL which can be registered to PETER PAN as normal.

The protocol

Finally the last question but certainly not the least. We have used one of the very well known P2P protocols, i.e, Gnutella.

- It is a de-facto Standard.
- Is easy to understand and implement
- Has many implementations
- Widely used in the Internet
- Can be easily extended
- Works properly in unstable network

There are some other advantages of this protocol that have encouraged us to use it. First it is an open protocol, which allows improvements to be incorporated immediately. It is for any one to share information on the Gnutella network. Gnutella allows users to share any kind of file from Microsoft document to games file or even just recipes. And the most important one is that Gnutella is decentralized i.e. it allows peers to interact directly with each other without any intermediate or central server.
Figure 9 shows the protocol implementation. It is mainly divided in two sections. First is the host manager and the second is the message manager. Let's start with the first section the host manager. The Connection manager tries to connect to the GNet (Gnutella network) and creates peer connection. Once it creates the connection it generates two threads: asynchronous sender and receiver. Then peer connection is added to the Connection Watch Dog. This component checks the status of each connection and keeps alive the connection. Once the connection receives the raw message then it sends it to the message management section. The host manager also has acceptor to accept request from outside. After it accepts the request from outside it follows the same procedure as the connection manager. The difference is that these connections are not locally generated.

Figure 9: The implementation of the PeTERPAN peer-to-peer protocol

Now let's look inside the message management section. Figure 10 shows how messages move in the core PeTERPAN. Once the Message Data arrives in the disector it checks the message for any errors either in the ttl, hop or the message type. Then it generates the appropriate message using the message factory. After the message is generated it is pushed forward to the one of the two queues. Messages that are forwarded to the forward queue are the once that do not need much processing for example ping message. Even the pong message is sent to the check queue because it needs caching. These queue checkers are running as threads to increase performance. Even the message handler run as threads however all threads belong to the same thread pool. The reason is that we want to prevent having excess number of threads running at the same time and that would be difficult to manage.

There is one layer that we have not discussed yet that is the security layer. Any computer that is a part of a network and is sharing data with others can be a target of a malicious user. Thus giving rise to security layer. At this point of time we are concentrating on the development of the core of PeTERPAN, however we are planning to incorporate this layer in the future. Finally stage is to integrate all the components together. As mentioned in the starting of this section we have used queues to connect one component with the other. Figure 11 demonstrates the working model of this implementation. The user of PeTERPAN will the see GUI and the application developer will see the API.
Figure 10: Diagram showing message management flow in PeTERPan

Figure 11: Working model of PeTERPan
The prototype

The current prototype of PeTerPan has the following basic functionalities: querying, searching, communicating among peers, and remotely executing Web Services. We are at a stage where the core of PeTerPan is in a working condition and we are ready to add on application to it. As we mentioned before, this middleware fully supports Web service operations. The distributed search for Web service can be done through the PeTerPan search feature. To carry this out, PeTerPan middleware must be installed on all participating computers. This will create a network of PeTerPan distributed system. To demonstrate how it operates, we provide the console graphical interface seen in the following figures. The three main tabs, namely Gnutella Network, Search, and Download, reflect the core distributed operations.

First of all, for a peer to connect to other peers, one must provides IP address and the TCP/IP port of the client in the format IP:port. Figure 12 exhibits the connection to two Peer-to-Peer systems; one is LimeWire (port 1326) and the other is BearShare (port 6350). Note here that BearShare and LimeWire are two of the well know Gnutella clients (Inc. 2003; LLC 2002). The GUI presents outcome of the information related to the connection. Except for the fourth and the last column the others are self-explanatory. The fourth specifies if the connection is self-initiated called outgoing or another client generates it called incoming. The last column in the picture shows the status of the connection—connecting, running or disconnected.

Figure 12: Connection to other Peer-to-Peer systems

Figure 13 depicts the search results when we specify the keyword xml. It searches the BearShare and Limewire clients for the results. To download the file we can double click on it.

Once we double-click on the file that is to be downloaded it will show the progress bar and store the file in the root directory of PeTerPan as shown in Figure 14.

Finally, we can use this console to perform decentralized search on any Web service registered to a peer in this network. Figure 15 shows the search results. At this stage we can choose to either run these services or create a new service by integrating them using the aforesaid visual editor. The visual editor is linked with the middleware. These output WSDLs can be placed in the canvas of the editor as discussed in Flow control component section.
Figure 13: Performing search on other peers

Figure 14: Showing the progress bar when double clicking on the file that we want to download
Figure 15: Web services can also be searched via PeTerPan

4 Case Study on Distributed GIS Applications

As mentioned before, there are certain classes of applications that naturally support the underlying distributed computing concept. In fact, these applications are not too far from our everyday life such as healthcare, banking, and many GIS (Geographic Information System) related applications. In this section, we are going to discuss about the GIS area. There are numerous GIS applications currently used in Thailand due to the high demands of both government and business sectors to effectively utilize resources. Because of this demand, many software vendors are competing to produce software with proprietary GIS data format hoping that it will become standard once they can dominate the GIS product market.

This creates many serious problems when conducting a large scale project (national or international level); that is the incompatibility of each system. These traditional GIS applications are standalone and the information exchanging among these systems is very primitive, e.g., using FTP or high capacity medium to exchange GIS data. Recently GIS applications which concentrates on providing GIS services through the Internet are being developed. By doing so, users can now view GIS results from remote site; however, internet GIS has led to more frequent sharing/exchanging GIS data. To alleviate the incompatibility of data formats, we adopt the Geography Markup Language (OpenGIS-Consortium 2001) (GML standard version 2.0) recommended by OpenGIS Consortium.

It is quite clear that many GIS problems requires distributed computing paradigm. With PeTERPAN this can be done easily by simply converting an existing GIS application to Web Service (Panatkool and Laoveerakul 2002). As mentioned previously, the PeTERPAN APIs shield the distributed computing resources from distributed applications. We developed a technology which will convert data from object-relational database¹ (ORDBMS) to GML data. Figure 16 shows the combination of different Web services together in the flow control visual editor to create the proposed GIS Web service. This GML data will be input to web services such as GML graphics processing service and GML maintenance service. This system is created for GIS governmental programs such as SchoolNet where the government would like to provide the internet access to

¹We are currently using PostGIS from PostGRES relational database system
students in remote schools and rural development project for Omkoi\textsuperscript{2} district.

19

![Figure 16: Current development of GIS web service by using the flow control visual editor](image1)

In summary, by utilizing PeTERPan middleware, we can efficiently deal with large-scale GIS system. The decentralized search in PeTERPan helps locate data and Web services required in this GIS application. In other words, the GIS Web services will take care data sets from different districts in Thailand. Data from each site can be patched and overlaid by our content sharing feature.

5 Conclusion and Future Work

We have presented the design and implementation of PeTERPan middleware which interacts with Web services on one end and distributed resources on the other end. The middleware supports creation of Web Services on a large scale network. The design of PeTERPan focuses on making a software layer that shields Web services from the heterogeneous infrastructure. This middleware uses Gridtella, our own java implementation of Gnutella P2P protocol to manage the decentralized distributed environment.

Future research

The design of PeTERPan has covered many issues; however, to prove the concept of combining Web services and P2P paradigm, the current prototype was limited to few important features, namely resource registration, searching, downloading, and basic Web service operations. The future research will include the following topics in order to improve the performance of this middleware as well as having a better support to new Web service technologies such as the new WSFL (flow language) that supports iteration and condition flow operations of multiple Web services. The possible research topics include (but not limited to) the following lists:

\begin{itemize}
  \item Preventing students from entering the workforce due to nutritional deficiency
  \item Further research into the impact of PeTERPan on rural development projects
  \item Integration with other Web service technologies
\end{itemize}

\textsuperscript{2}Omkoi is a small district of Chiangmai province of Thailand. People who live in this area are suffering from nutritional deficiency.
1. We need to focus more on the security issue including distributed trust, access control, delegation, authenticity to support operations in distributed system environment.

2. The economic model (accounting, cost of services) should be carefully thought of in order to support Web services in business.

3. Subgroup in PET ER P AN—partitioning peers into groups to raise privilege level for peers in the same group.

4. Improvement of the protocols including caching, message routing (reduction of the broadcasting message in Gnutella), and

5. Focus on moving/copying Web services to be executed on peers that have the right combination of resources. Note that to support this Java was chosen as the language of our choice so that the moving can be done smoothly.

References


FOLDING@home (2002). Folding@home distributed computing. WWW: folding.stanford.edu.


SETI@home (2002). The search for extraterrestrial intelligence. WWW: setiathome.ssl.berkeley.edu.


W3-Consortium (2000b, May). Simple object access protocol 1.1. WWW: www.w3.org/TR/SOAP.

