Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

Application Programming Interface for WOSP/WOSRP

Gilbert Babin[1], Hauke Coltzau[2], Markus Wulff[2], and Simon Ruel[1]

[1] Département d'informatique, Université Laval,
Sainte-Foy (Qubec) Canada G1K 7P4
{babin,ruelsimo}@ift.ulaval.ca
[2] Fachbereich Informatik, Universität Rostock,
18059 Rostock, Germany
{django,mwulff}@informatik.uni-rostock.de

Abstract. The Web Operating System (WOS™) allows for a user to submit a service request without any prior knowledge about the service and to have it fulfilled according to the user's desired constraints/requirements. Such services may be specialized hardware or software, or both. The WOS considers the communication layer to be the centralized part. The communication protocols may thus be seen as the "glue" of the WOS architecture. This paper presents an Application Programming Interface (API) to access WOS communication services. In order to present how the communication layer works, we introduce all the concepts related to the communications in the WOS and will show how these components are put together to support communication.

1 Introduction

The Web Operating System (WOS™) [3, 4, 5] was developed to provide a user with the possibility to submit a service request without any prior knowledge about the service (where it is available, at what cost, under which constraints) and to have the service request fulfilled within the user's desired parameters (time, cost, quality of service, etc.). In other words, the WOS is designed to enable transparent usage of network-accessible resources, whenever a user requires a service, wherever the service is available. These services may be specialized hardware or software, or a combination of both. A user needs only to understand the WOS interface, and does not need to known how the service request is fulfilled. Therefore the WOS provides a computation model and the associated tools to enable seamless and ubiquitous sharing and interactive use of software and hardware resources available on the Internet.

The WOS is designed as a fully distributed architecture of interconnectec nodes where the communication protocols are considered to be the centralized parts. The communication protocols may thus be seen as the "glue" of the WOS architecture. Communication between nodes is realized through a simple discovery/location protocol, the WOS Request Protocol (WOSRP), and a generic service protocol, the WOS Protocol (WOSP). The WOSP protocol is in fact

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

a protocol language with a corresponding parser and serves to easily configure service-specific protocol instances. For example, one WOSP instance could implement an interface to XML, CBL (Common Business Library) or GIOP/IIOP of CORBA. At the lower levels of the protocol stack, we assume the usage of the TCP/IP protocol family.

This paper describes an Application Programming Interface (API) to access WOS communication services provided by WOSP/WOSRP. The need for new mechanisms to use communications in the WOS has arisen from WOS application developers who wanted to have a better control over communications. Many versions of WOSP may exist. Each version supports the communications for one class of services provided by the WOS. All these versions, however, share a common syntax. From a pure object-oriented perspective, we clearly see that a single class (*WOSP_Parser*) can manage syntax processing, while multiple classes are required to process the semantics. This is exactly what was done in the first implementation of the WOS communication layer [1, 2]. To make this possible, we had defined a class (*WOSP_Analyzer*) which was specialized for each version of WOSP. Once a message was processed for its syntax, the *WOSP_Parser* class would locate the appropriate specialization which would process the semantics.

We identified two majors problems with this approach:

1. The service classes could not be developed independently from the communication layer, because they had to specialize *WOSP_Analyzer*.
2. The communication layer was controlling the flow of processing for the whole system, since *WOSP_Parser* was explicitly calling the specialized *WOSP_Analyzer*.

In addition, the initial design of WOSRP/WOSP assumed that a synchronous dialog was required between two nodes in the connection-oriented mode. It turns out that this requirement imposes too much constraints on the service class developer. For instance, the application developer must guarantee that all communications between clients and servers be synchronized.

The new API presented in this paper alleviates these problems by removing every aspects of the semantics processing from the communication layer. This should provide more flexibility to the service class developper. It also supports asynchronous communications between clients and servers. In order to present how the new communication layer API works, we first introduce all the concepts related to the communications in the WOS (Sect. 2). We will then show how these components are put together to support these communications (Sect. 3). We conclude this article in Sect. 4.

2 Communication Concepts for the WOS

This section presents some basic concepts required to understand communications in the WOS.

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

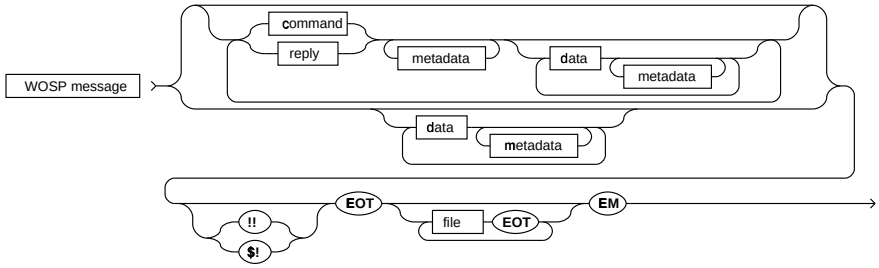

Fig. 1. WOSP generic syntax

2.1 WOSP Message

A WOSP message is a data stream that has the syntax illustrated in Fig. 1, where `<message>` may be a command or a reply. For a command, we provide the command type (execution, query, setup), the command name, and the command identifier. The command identifier is composed of the WOSP message identifier (see Sect. 2.4), the position of the command within the message, and the number of data and metadata elements describing the command. For a reply, we provide the identifier of the original command to which this reply is addressed and a command identifier.

Applications using WOSP to communicate do not create that data stream. Indeed, each element of the WOSP message syntax may be represented by a data structure called a triplet. A triplet contains three fields, as indicated by its name : a triplet type, a triplet name, and a triplet value. Table 1 shows how WOSP message information is stored in a triplet.

Table 1. WOS triplet structure

Type	**Name**	**Value**
EXECUTE	name of execution command	command identifier
QUERY	name of query command	command identifier
SETUP	name of setup command	command identifier
REPLY	identifier of command to which this is a reply	command identifier
DATA	name of data field	value of data field
METADATA	name of metadata field	value of metadata field
FILE	local name of data file	*not used*

A message is therefore composed of a list of triplets. When sending a message, the application composes a list of triplets that it provides to the WOS communication layer. When receiving a message, the application receives a list of triplets from the WOS communication layer.

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

Because all WOSP messages use the same syntax, a single module may be used to construct a data stream from the list of triplets and vice versa. The interpretation of the content of the message depends on the version of WOSP. Therefore, the different versions of WOSP correspond to the different semantics that we can associate to the basic syntactic elements presented here.

2.2 WOSRP Messages

The WOS Request Protocol (WOSRP) serves two purposes : locating versions of service families (i.e., specific WOSP versions) and transmitting WOSP messages to an appropriate server (version and service family). To locate services, two types of messages are used, namely WOSRP request and WOSRP reply message types.

Two modes are available to transmit WOSP messages : connectionless mode and connection-oriented mode. In connectionless mode, an application sends a single WOSP message to another application which acts as a server to a specific WOSP version, without establishing a long term connection with that node; once the message is transmitted, the connection is closed. In this case, the WOSRP message is used to identify the appropriate WOSP version server and to encapsulate the WOSP message.

In connection-oriented mode, the application must explicitly establish a connection with a specific WOSP version server. Once the connection is established, it can send and receive WOSP messages asynchronously, until the connection is closed or broken. In this case, the WOSRP message is used to establish the connection with the appropriate WOSP version server.

2.3 Message Queues

In order to enable fully asynchronous communications, WOSP and WOSRP messages must be queued at the receiving side of the communication link. Therefore, the WOS communication servers put the received WOSP and WOSRP messages into appropriate queues. Once an application is ready to process the next message, it requests it from a queue. A queue is uniquely identified by a Message Queue Identifier (MQID; see Sect. 2.4). We will now present the different queue types used by the WOS communication layer.

WOSRP Request Queue. Queues of this type contain WOSRP requests received by a WOS node. A given WOS node will have *only one* queue of that type.

WOSRP Reply Queue. Queues of this type contain WOSRP replies received by a WOS node. A given WOS node will have *only one* queue of that type.

WOSP Connectionless Queues. Queues of this type will contain WOSP messages received in connectionless mode by a WOS node for a specific WOSP version. A WOS node may therefore have *zero or more* such queues.

WOSP Connection Request Queues. Queues of this type will contain WOSP connection requests received by a WOS node for a specific WOSP version. A WOS node may therefore have *zero or more* such queues. However, there

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

will be *at most one* such queue for every WOSP version known by a WOS node.

WOSP Connection-oriented Queues. Queues of this type will contain WOSP messages received in connection-oriented mode by a WOS node. They are bound to a WOSP connection. There will be *one* such queue for every WOSP connection established with another WOS node.

2.4 Message Identifiers

One of the concerns in WOS communications is that every message is uniquely identified. We also want the identifiers used to support very fast CPUs, where a large number of messages may be generated by the same machine within a short time interval. The WOSP Message identifier is built with these constraints in mind. It is the concatenation of the following elements :

1. the domain name (or IP address) of the machine sending the message,
2. the port number where replies should be sent to,
3. the MQID where replies should be stored, and
4. a timestamp.

The MQID is a timestamp representing the moment where the queue was created. Timestamps (and MQIDs) have the following syntax :

1. the year (4 digits),
2. the month (2 digits),
3. the day of the month (2 digits),
4. the hour (2 digits),
5. the minutes (2 digits),
6. the seconds (2 digits),
7. a 5 character alphanumerical string (taken from a list of ASCII characters non conflicting with the communication layer).

Timestamps (and MQID) are provided by a single server running on the WOS node to avoid duplicates and therefore guarantee unicity.

2.5 WOSP Version Identifiers

Every version of WOSP must be uniquely identified. Furthermore, the possible number of existing versions may grow quite large. Consequently, the format of WOSP version identifiers must accommodate a potentially large domain of values. We chose to identify WOSP versions using a string of 448 Bytes, thus allowing for 2^{3584} ($\approx 10^{1079}$) distinct values.

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

2.6 Queue Servers

The WOS communication layer is responsible for receiving and placing into the appropriate queue all the WOSP and WOSRP messages addressed to a specific WOS node. However, the communication layer does not process these messages. This is accomplished by queue servers. A queue server is an application that informs the WOS communication layer that it will process messages put into a specific queue. We say that the application "registers" as the queue server.

A queue may have at most one queue server. However, an application may act as queue server for many queues. For each queue type, the communication layer knows one application that acts as default queue server for that queue type. This way, if a queue is not empty but no application has registered as queue server, the WOS communication layer can launch an application that will register as queue server (the default queue server).

3 Operation of the Communication Layer

The Communication Layer is responsible for the reception of incoming messages and the transmission of outgoing messages. In this section, we present how these two functions are achieved and how the user applications may access these services of the communication layer. The prototype of the communication layer was developed in JAVA. The communications between WOS nodes are achieved using TCP/IP while access to the API by a WOS application is achieved using the Remore Method Invocation package (RMI) supplied with Sun's JAVA Development Toolkit (JDK).

3.1 Registering Queue Servers

The registration process depends on the queue type. For WOSRP request queues (*WOS_RegisterRequestServer()*) and WOSRP reply queues (*WOS_RegisterReplyServer()*), the application selects the appropriate method. If an application is already registered for these queues, the registration will fail. Otherwise, the application will receive the MQID.

For WOSP connectionless queues, the application must provide the WOSP version (*WOS_RegisterConnectionlessServer(VersionID)*). Depending on which version of WOSP is used, many WOSP connectionless queues may be active for the same WOS node. On receiving a WOSP message in connectionless mode, the communication layer will dispatch the message parts to the appropriate queue (using the WOSP message identifier; not yet implemented).

For WOSP connection request queues, the application must also provide the WOSP version (*WOS_RegisterConnectionServer(VersionID)*). In this case, at most one application may register to act as connection request server for a specific WOSP version.

Applications do not need to register as WOSP connection-oriented queue server. This is done implicitly when the connection is acknowledged. When a connection request is made, the WOSP connection request queue server launches the

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

appropriate application and supplies it the MQID of the corresponding WOSP connection-oriented queue. The queue is removed when the connection is closed.

3.2 Receiving WOSRP and WOSP Messages

Initially, no message queue is available. A queue is created by the WOS communication layer either when a new message of a specific type is received, in which case the corresponding default queue server is launched, or when an application registers as queue server.

For any new message it receives, the communication layer first determines its type. This type is used to place the message in the appropriate message queue. In the case of connectionless WOSP messages, a more detailed analysis of the message is needed. Since many applications may act as queue server for a specific WOSP version, the communication layer must dispatch replies to the appropriate queue by looking at the command identifier of each command contained in the message. This dispatch feature is not yet implemented; in the current implementation, connectionless messages are stored in the first queue found for the corresponding WOSP version.

Figure 2 illustrates in more details how a (WOSP or WOSRP) message is processed. In the first step, the relevant WOSRP Connection Server (either for TCP connections or UDP datagrams) retrieves the message from the network. The message is processed by the WOSP/WOSRP Message Manager (Step 2) which, in turn, issues a warehouse (cache) lookup request (Steps 3 and 4) to the Search Engine to determine whether this version is already served or not, and if not, to retrieve the command to launch the corresponding default queue server. The message is then stored in the appropriate queue.

The queue server eventually fetches a message (Step 6) with one of the following methods:

WOS_GetMessage(MQID). This method is used to retrieve the next available message in the message queue identified by MQID, if any message is available.

WOS_WaitForMessage(MQID). This method waits indefinitely for the next available message in the message queue identified by MQID.

WOS_WaitForMessage(MQID, t). This method waits at most t seconds for the next available message in the message queue identified by MQID.

The WOSP/WOSRP Message Manager locates the appropriate queue (Step 7) and returns the first message found (Step 8), if any, depending on the access method used.

3.3 Sending WOSRP and WOSP Messages

In general, all WOS messages are sent using the same mechanisms. However, variations exist which we will describe in details later in this section. Figure 3 presents an overview of the transmission process. Requests for sending messages are made to the WOS Communication API (Steps 1 and 2). The API accesses

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

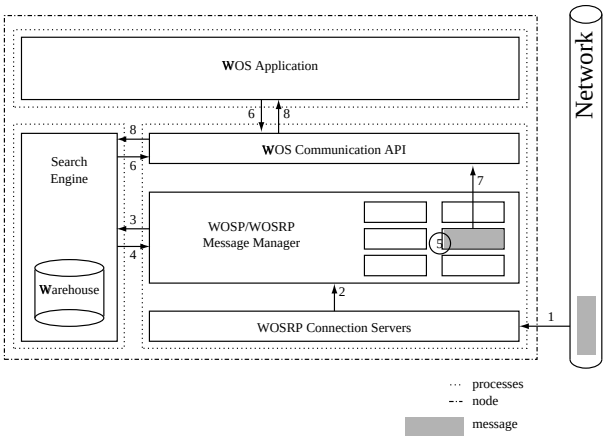

Fig. 2. Receiving messages at a WOS node

the appropriate module within the WOSP/WOSRP Message Manager to build either a WOSRP or a WOSP message (Step 3; see the following subsections), which, in turn, is sent to the network (Step 4).

Sending WOSRP Messages. Any application may send WOSRP requests and replies. Special commands are available for that purpose:

WOSRP_Request(...). This method is used to send WOSRP requests.
WOSRP_Reply(...). This method is used to send WOSRP replies.

Establishing WOSP Connections. For connection-oriented communications, a connection must first be established. An application sends a connection request message and waits for a positive acknowledgment (or a timeout) using the method *WOSP_SetupConnection(...)* with the appropriate arguments. On receiving such a request, a WOS node creates a WOSP connection-oriented queue and the connection request message containing the MQID of the newly created queue is put in the WOSP connection request queue for the appropriate WOSP version. The corresponding connection request queue server decides to accept (*WOSP_AcceptConnection(MQID)*) or not (*WOSP_RejectConnection(MQID)*) the connection. The communication layer may also throw a timeout exception if the connection request queue server takes too much time to process the connection request. In this case, the message is removed from the queue, a negative acknowledgment is sent to the requesting machine, and the WOSP connection-oriented queue is removed. The requesting application will then receive a null value to indicate that the request for connection failed.

When the connection is accepted, a positive acknowledgment is sent to the requesting node. On reception of the acknowledgment, a local WOSP connection-oriented queue is created and the MQID is supplied to the requesting application.

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

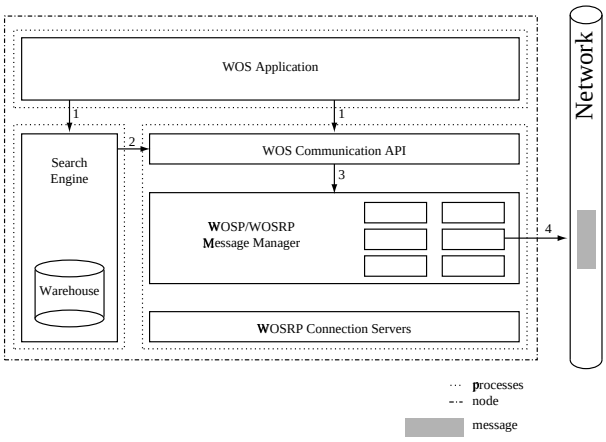

Fig. 3. Sending messages at a WOS node

Sending WOSP Messages. WOSP messages are sent directly to a remote host without establishing a connection or through an already established connection using one or the other form of the method *WOSP_SendMessage(...)*. We will now see how the communication layer transmits these messages.

WOSP Connectionless Messages. An application may send a WOSP connectionless message using one of two approaches :

1. by specifying the remote host IP address and port number (optional) of the remote WOS communication server. In this case, the WOSP message identifier is generated using the default WOSP connectionless queue for the specified WOSP version. If no such queue exists, one is created.
2. by providing a WOSP connectionless queue identifier, along with the remote host IP address and port number (optional) of the remote WOS communication server. In this case, the WOSP message identifier is generated using the MQID provided.

WOSP Connection-oriented Messages. Sending a WOSP message in connection-oriented mode requires that a connection be established. Access to the connection is given by the WOSP connection-oriented queue ID supplied to the application when then connection was created. Therefore, the application must only provide the MQID associated to the connection to send a message.

Furthermore, a node may request that the connection be closed. This information will be part of the WOSP message (the special "`!!`" command in the WOSP syntax). The other participant must acknowledge the termination of the connection. Again, this information will also be part of the WOSP message (the special "`$!`" command in the WOSP syntax). The connection is closed only when both sides have agreed on the termination of the connection. The communication layer keeps track of close-connection requests and acknowledgments.

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

3.4 A Complete Example

The WOS communication layer is accessed by using RMI services. It must therefore create a reference to an RMI registry, which will be used to locate the interface to the communication layer. A WOS application should have the following structure :

```
import java.net.*;
import java.rmi.*;
import java.rmi.registry.*;
import wos.comm.interf.*;
import wos.comm.message.*;
public static void main ( String argv[] ) {
   Registry register;
   WOS_Client interface = null;
   try{
      String hostName = InetAddress.getLocalHost().getHostName();
      register = LocateRegistry.getRegistry(hostName);
      interface = (WOS_Client) register.lookup("WOS_Interface");
      // Your code is placed here
   } catch (java.net.UnknownHostException e){
   } catch (NotBoundException e) {
   } catch (RemoteException e) {
   }
}
```

The object *interface* will provide the access to all the methods of the communication layer.

We show here a complete communication cycle for a WOSRP request/reply (Fig. 4). The cycle comprises 4 sequences of actions, represented by the different dashed lines. Each sequence starts at a numbered circle, the number indicating the relative order of the sequence. We describe each of these sequences as follows :

1. A WOS application requests information about an available service class (i.e., a specific WOSP version). The information is not available locally, and a network search is initiated. The message is placed in the appropriate queue at the remote host. The call looks like this:

```
try {
    InetAddress hostA = InetAddress.getByName("hostA");
    interface.WOSRP_Request(hostA,        // Recipient IP address
                            9671,         // Recipient port number
                            true,         // hostA speaks the version
                            true,         // Version ID included
                            1,            // Hop count
                            "versionID"); // Version looked for
} catch (java.net.UnknownHostException e) {
} catch (java.rmi.RemoteException e) {
}
```

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

2. The Search Engine fetches the next message in the WOSRP Request queue. This is done with the following call to the API:

```
try {
    String MQID = interface.WOS_RegisterRequestServer();
    WOS_Message msg;
    while ((msg = interface.WOS_WaitForMessage(MQID,60)) != null) {
        // Process msg
    }
    interface.WOS_Unregister(MQID);
} catch (java.rmi.RemoteException e) {}
```

3. An answer (a WOSRP Reply message) is produced and sent to the requesting node. The message is placed in the appropriate queue at the remote host. The JAVA code looks like this:

```
try {
    InetAddress hostA = InetAddress.getByName("hostA");
    InetAddress hostB = InetAddress.getByName("hostB");
    interface.WOSRP_Reply(hostB,        // Recipient IP address
                          9671,         // Recipient port number
                          hostA,        // Server IP address
                          true,         // hostA speaks the version
                          "versionID"); // Version spoken
} catch (java.net.UnknownHostException e) {
} catch (java.rmi.RemoteException e) {}
```

4. The Search Engine fetches the next message in the WOSRP Reply queue. The information is used to provide and answer to the WOS application and to update the local warehouse. This uses the same code as sequence 2, except that we register as reply queue server instead of request queue server.

4 Conclusion

This paper has described the concepts related to the WOS communication layer, required for the development of a new API. Furthermore, a description of the communication mechanisms was provided. This new API was developed using Java programming language. Access to the communication layer is provided by the Remote Invocation Method (RMI) package. In its current implementation, the API only supports queue servers developed in Java. It would be interesting to investigate other implementation approaches for the API that would allow for queue servers to be developed using other languages. One such approach is CORBA.

Although the new API provides a more flexible utilization of the WOS communication layer and supports asynchronous communications, some issues remain unresolved. For instance, a naming scheme (naming space, naming conventions, name assignment, distributed name management, etc.) still needs to be specified.

Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. "Application Programming Interface for WOSP/WOSRP." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 110-121.

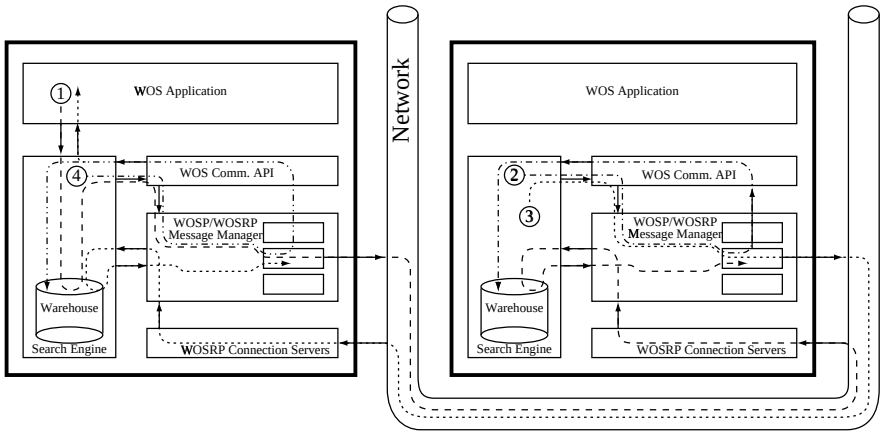

Fig. 4. A WOSRP request/WOSRP reply cycle

We also need to investigate the use of network and/or transport layer protocols other than the TCP/IP protocol family. This will require an extension of the WOSRP protocol to consider other transport mechanisms.

References

[1] Gilbert Babin. Requirements for the implementation of WOS™ protocols. In *Distributed Computing on the Web Workshop (DCW'98)*, pages 129–133, Rostock, Germany, June 1998.

[2] Gilbert Babin, Peter Kropf, and Herwig Unger. A two-level communication protocol for a Web Operating System (WOS™). In *IEEE Euromicro Workshop on Network Computing*, pages 939–944, Västerås, Sweden, August 1998.

[3] Peter Kropf. Overview of the WOS project. In *1999 Advanced Simulation Technologies Conferences (ASTC 1999)*, San Diego, CA, USA, April 1999.

[4] P.G. Kropf, J. Plaice, and H. Unger. Towards a WEB operating system. In *Webnet '97*, Toronto, Canada, November 1997. Association for the advancement of computing in education.

[5] John Plaice and Peter Kropf. WOS communities — interactions and relations between entities in distributed systems. In *Distributed Computing on the Web (DCW'99)*, Rostock, Germany, June 1999.