Gilbert Babin, Hauke Coltzau, Markus Wulff, and Simon Ruel. Application Programming Interface for WOSRP/WOSP. Technical report no. DIUL-RT-9901. Université Laval. Ste-Foy, Québec, Canada. September 1999.

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

Application Programming Interface for WOSRP/WOSP

September 14, 1999

1 Introduction

The need for new mechanisms to use communications in the WOS™ has arisen from system developers who wanted to have a better control over the communications. Indeed, in its previous version, the WOS™ communication layer was designed such that it was "in charge" of all the processing being done. Furthermore, it became obvious that any application to be linked to the WOS™ would require to be linked with the entire communication layer.

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 application developer. For instance, the application developer must enforce this synchronicity to his "client" software as well as its "server" software.

Therefore, a design is required that will not require synchronicity. Furthermore, this new design gives total control of the communication to the application. In order to present how the communication layer should work in its future version, we will first introduce all the concepts related to the communications in the WOS™. We will then show how these components are put together to support these communication. Finally, we will provide a detailed specification of the application programming interface (API) to use the communication services provided by the WOS™.

2 Communication concepts for the WOS

This section present basic concepts required to understand communications in the WOS™.

2.1 WOSP Message

A WOSP message is a data stream that has the syntax illustrated in Figure 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 below), the position of the command within the message, and

[1]Département d'informatique, Université Laval
[2]Fachbereich Informatik, Universität Rostock

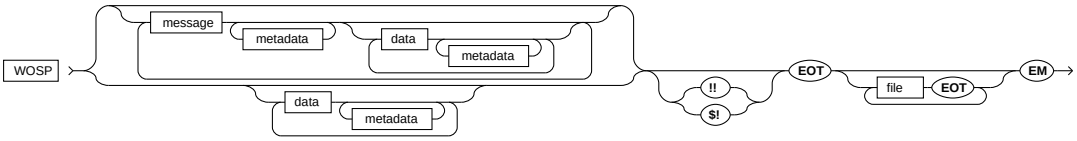

Figure 1: WOSP Generic Syntax

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*

the number of data and metadata elements describing the command. For a reply, we provide the identifier of the original command for which this reply is provided 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.

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 will receive a list of triplets from the WOS™ communication layer.

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 ended. 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 within the WOS™, WOSP and WOSRP messages must be queued. Therefore, the WOS™ communication servers receive WOSP and WOSRP messages and put them 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 below). 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 *many* 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 *many* such queues. However, there 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. These queues are bound to a WOSP connection. There will be *one* such queue for every WOSP connection established with another WOS node.

2.4 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 the same second. 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. It has the same syntax as the timestamp used in the WOSP message identifier :

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 (from ASCII 32 to ASCII 126) alphanumerical string.

Timestamps are provided by a single server running on the WOS node to avoid duplicates and therefore guaranty unicity.

2.5 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 inform 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. All queues also have well known queue servers. This way, if a queue is not empty but no application has registered as queue server, the WOS™ communication layer will launch an application that will register as 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.

3.1 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 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.

3.2 Registering Queue Servers

The registration process depends on the queue type. For WOSRP request queues and WOSRP reply queues, the application only provides the queue type. 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 queue type and the WOSP version[1]. Depending on the WOSP version, 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).

For WOSP connection request queues, the application must also provide the queue type and the WOSP version. In this case, at most one application may register to act as connection request server for a specific WOSP version.

Applications may not register for WOSP connection-oriented queues. When a connection request is made, the WOSP connection request queue server launches the appropriate application and supplies it the MQID of the corresponding WOSP connection-oriented queue. The queue is removed when the connection is closed.

3.3 Sending WOSRP Messages

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

3.4 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). 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 queue server decides whether

[1]Version management is under the supervision of other WOS modules. The application will interact with those modules to obtain the appropriate WOSP version identifier.

it accepts or rejects the connection. The communication layer may also throw a timeout exception if the 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.

3.5 Sending WOSP Messages

WOSP messages are sent directly to a remote host without establishing a connection or through an already established connection. We will now see how the communication layer transmits these messages.

3.5.1 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, it 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.

3.5.2 WOSP Connection-oriented Messages

Sending a WOSP message in connection-oriented mode requires that a connection is 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. Table 2 shows the state of the connection, based on the messages send and received by both WOS nodes involved in a connection.

Table 2: States While Closing a WOSP Connection

Previous State	Message From Node A	Message From Node B	New State
Connected	contains no !! nor $!		Connected
Connected	contains !!		Request by A
Connected	contains $!		Request by A
Connected		contains no !! nor $!	Connected
Connected		contains !!	Request by B
Connected		contains $!	Request by B
Request by A	contains no !! nor $!		Connected
Request by A	contains !!		Request by A
Request by A	contains $!		Request by A
Request by A		contains no !! nor $!	Request by A
Request by A		contains !!	Closed
Request by A		contains $!	Closed
Request by B	contains no !! nor $!		Connected
Request by B	contains !!		Closed
Request by B	contains $!		Closed
Request by B		contains no !! nor $!	Request by B
Request by B		contains !!	Request by B
Request by B		contains $!	Request by B

4 Communication Layer Interface Specification

In this section we present the public interface of the WOS class, which is the startup class of a WOS node. The methods are classified by the type of services provided. In all cases, parameters that can be omitted, such as port number and timeouts, use default values declared in configuration file `WOS.conf`. This file defines the following parameters :

`TMP_DIR` The directory where temporary files are stored by the WOS.

`JAVA_BIN` The path to the `Java` virtual machine.

`WOSRP_DEFAULT_LOCAL_PORT` The local port where WOSRP waits for messages.

`WOSRP_DEFAULT_REMOTE_PORT` The default remote port for WOSRP messages.

`WOS_TIMESTAMP_PORT` The local port where WOS applications request timestamps.

`WOSP_LOCAL_CONNECT_TIMEOUT` The time elapsed on the requesting machine before a connection request fails.

`WOSP_CONNECT_PROCESS_TIMEOUT` The time elapsed on the processing machine before a connection request fails.

4.1 Reception of Messages

A queue server is in charge of looking at the content of the message queue it serves and of processing the messages found therein. Messages are fetched one at a time from the message queue using a non-blocking (*WOS_GetMessage*) or a blocking (*WOS_WaitForMessage*) method. A timeout (in seconds) may optionally be provided in the blocking approach.

`WOS_Message` *WOS_GetMessage*`(WOS_MessageQueueID id)`

returns: The next message from the specified message queue or null, if no message is available on the queue.

`WOS_Message` *WOS_WaitForMessage*`(WOS_MessageQueueID id)`

returns: The next message from the specified message queue.

`WOS_Message` *WOS_WaitForMessage*`(WOS_MessageQueueID id, int timeout)`

returns: The next message from the specified message queue or null, if no message is available on the queue after *timeout* seconds.

4.2 Registration

Four methods are available for registration, one for each queue type. This is to simplify the processing of parameters. All four methods return a valid MQID when registration is successful, or a null value, otherwise.

`WOS_MessageQueueID` *WOS_RegisterRequestServer*`()`

returns: The MQID for the WOSRP request queue or null, if registration failed.

`WOS_MessageQueueID` *WOS_RegisterReplyServer*`()`

returns: The MQID for the WOSRP reply queue or null, if registration failed.

`WOS_MessageQueueID` *WOS_RegisterConnectionlessServer*`(String VersionID)`

returns: The MQID for a WOSP connectionless queue or null, if registration failed.

`WOS_MessageQueueID` *WOS_RegisterConnectionServer*`(String VersionID)`

returns: The MQID for a WOSP connection request queue or null, if registration failed.

When an application cannot act as queue server anymore, it should unregister itself using the following method.

`void` *WOS_Unregister*`(WOS_MessageQueueID id)`

4.3 WOSRP Messages

The following commands are used to send WOSRP requests and replies.

```
void WOSRP_Request(InetAddress RecipientIP, short RecipientPort,
     boolean SpokenOrKnown, boolean SpecificOrAny, short HopCount,
     String VersionID)
```

```
void WOSRP_Reply(InetAddress RecipientIP, short RecipientPort,
     InetAddress ServerIP, short ServerPort, boolean SpokenOrKnown,
     String VersionID)
```

4.4 Establishment of a Connection

Several methods are available for the establishment of a WOSP connection. The following methods are used by an application to create a connection with a remote WOS node. The timeout (default : `WOSP_LOCAL_CONNECT_TIMEOUT`) is specified in seconds.

```
WOS_MessageQueueID WOSP_SetupConnection(InetAddress IP, int port,
     String versionID)
```

returns: The MQID, if the connection is accepted, null otherwise.

```
WOS_MessageQueueID WOSP_SetupConnection(InetAddress IP, int port,
     String versionID, int timeout)
```

returns: The MQID, if the connection is accepted, null otherwise.

```
WOS_MessageQueueID WOSP_SetupConnection(InetAddress IP, String versionID)
```

returns: The MQID, if the connection is accepted, null otherwise.

```
WOS_MessageQueueID WOSP_SetupConnection(InetAddress IP, String versionID,
     int timeout)
```

returns: The MQID, if the connection is accepted, null otherwise.

The queue server may accept or reject connection requests. The following methods are used for that purpose. A connection request must be read for processing by the queue server within 60 seconds[2] of its reception. The communication layer uses the WOSP connection-oriented queue identifier to uniquely identify the connection. The queue server therefore uses this MQID to properly identify which connection to accept or reject.

```
void WOSP_AcceptConnection(WOS_MessageQueueID id)
```

```
void WOSP_RejectConnection(WOS_MessageQueueID id)
```

[2]This timeout value may be changed by modifying `WOSP_CONNECT_PROCESS_TIMEOUT` value.

4.5 Transmission of WOSP Messages

The following commands are used to transmit messages to other WOS nodes. The transmission mode (connectionless and connection-oriented) depends on the parameters to the commands.

4.5.1 Connectionless Mode

`WOSP_MsgId` *WOSP_SendMessage*`(InetAddress IP, int port, String versionID, WOSP_ListOfTriplets message)`

returns: 0 if the message was successfully sent, -1 otherwise

`WOSP_MsgId` *WOSP_SendMessage*`(InetAddress IP, String versionID, WOSP_ListOfTriplets message)`

returns: 0 if the message was successfully sent, -1 otherwise

`WOSP_MsgId` *WOSP_SendMessage*`(WOS_MessageQueueID id, InetAddress IP, int port, WOSP_ListOfTriplets message)`

returns: 0 if the message was successfully sent, -1 otherwise

`WOSP_MsgId` *WOSP_SendMessage*`(WOS_MessageQueueID id, InetAddress IP, WOSP_ListOfTriplets message)`

returns: 0 if the message was successfully sent, -1 otherwise

4.5.2 Connection-oriented Mode

In connection-oriented mode, the application may also optionally set a close-connection flag when sending a (possibly empty) list of triplets. Depending on the state of the connection, the close-connection flag will result in the transmission of a close connection request or a close connection acknowledgment.

`WOSP_MsgId` *WOSP_SendMessage*`(WOS_MessageQueueID id, WOSP_ListOfTriplets message)`

returns: 0 if the message was successfully sent, -1 otherwise

`WOSP_MsgId` *WOSP_SendMessage*`(WOS_MessageQueueID id, WOSP_ListOfTriplets message, boolean doClose)`

returns: 0 if the message was successfully sent, -1 otherwise