WATOO: An Internet Access Software to a Satellite Tracking Station*

M. Normandeau, S. Bernier, J.-M. Desbiens, and M. Barbeau
Département de mathématiques et d'informatique
Université de Sherbrooke
Sherbrooke, Québec CANADA J1K 2R1
{*normand, berns00, desbiens, barbeau*}*@dmi.usherb.ca*

Abstract

The goal of the project presented in this paper is to provide means to share the resources of a satellite tracking station via the Internet. Being costly and cumbersome, this kind of installation is not affordable to everybody. But with the outreach offered by the Internet, it is possible to share this material. In particular, we hope that this project will make easier participation of students to communication programs with astronauts (such as SAREX).

The project itself can be divided into two main areas: the client application and the satellite telecommunications server. The client application is the user interface. Using this application, an internaut is able to connect to a satellite telecommunications server and talk on a satellite frequency (an amateur radio license is required). The server manages the control information exchanges, antennas, and radios. Both the client and server designs are object-oriented. The client is programmed in Java and the server is implemented in C++. Our software is called WATOO which stands for World wide Access To Orbiting Objects. This paper presents the user interface of WATOO and briefly reviews the design of the client and server.

*The work described in this paper was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR), and Bombardier Inc.

1 Introduction

This paper presents a novel software providing access to satellite telecommunications stations via the Internet. Installation of a satellite telecommunications station is not affordable to everybody because is requires a certain investment of money, expertise with satellite telecommunications equipment, and adequate space for antennas. Our goal is to develop software that enables distant access and sharing of this kind of material through the Internet. In particular, we hope that with such software, participation of students to communication programs with astronauts (such as SAREX) will be more accessible. Our software is called *World wide Access To Orbiting Objects* (WATOO).

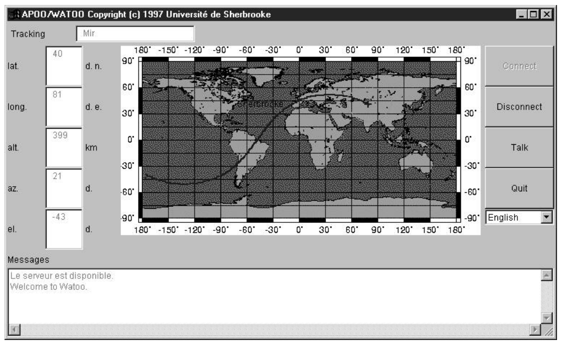
The software is divided in two parts: the client application and the satellite telecommunications server. The client application is the user interface. Through this application, a radio amateur internaut may get connected to a satellite telecommunications server and talk on a satellite voice radio frequency. The server controls the antennas and radios and sends/receives voice over the radio channel.

The satellite telecommunications server comprises the following pieces of equipment. Outside the station, there is a system of crossed yagi antennas for the 2 m and 70 cm bands. Their length is approximately 10 feet. There are azimuth and elevation rotors to maintain the direction of antennas toward a satellite. Inside the station, there is a Yaesu FT-736R radio covering the 2 m and 70 cm bands. The satellite telecommunications server runs on a Pentium personal computer executing the Linux operating system. The computer has a hardware interface with the rotors of the antennas and a hardware interface with the radio, so they can be automatically controlled by software. The computer also has a sound card connected to the radio audio in an out.

This paper presents the user interface of WATOO (Section 2) and briefly reviews the design of the client and server (Section 3).

2 User interface

The user interface of the software is illustrated in Figure 1. In the top left corner of the window, the name of a tracked satellite is posted. On the left side, there is a panel of four numeric values indicating the current satellite latitude, longitude, and altitude, and antenna direction, in terms of azimuth and elevation. For the sake of simplicity and readability by non-experts, all numeric values are displayed as rounded integers, although computations within the server are done using double floats. In the middle, there is a map showing the ground location of the server and the point directly below the satellite (the subsatellite point (SSP)). An ovoid figure is drawn around the subsatellite point that delimits the coverage area. Of course, the location of the satellite telecommunications server, and not necessarily the one of the client, has to be within the coverage area to make communication possible. On the right side, there is a panel of four buttons. A click on the button *Connect* (*Disconnect*) triggers the client to server connection establishment (release) process. The button *Talk* launches a voice over the Internet communication software. The software that we use for this purpose is named Speakfreely [6]. The last button on the bottom allows selection of a language in which names of fields and buttons as well as messages are displayed (presently, English and French are supported).



APOO/WATOO Copyright (c) 1997 Université de Sherbrooke

Tracking: Mir

lat. 40 d. n.
long. 81 d. e.
alt. 399 km
az. 21 d.
el. -43 d.

Messages:
Le serveur est disponible.
Welcome to Watoo.

Figure 1: User interface

3 Design

The design has been devised using the object-oriented development method of Booch [1].

3.1 Client application

One of our main goals was to develop a client application that can run across several platforms. At first, we turned our attention to the Java language [3] that aims to have the portability property. Development in Java means re-use as much as possible of Java components, from a rich collection of classes, and Java architectural patterns. Adoption of Java has therefore considerably influenced the structure of the interface. However, we were not able to do everything in Java, while investing a reasonable amount of efforts. Hence, we achieved the portability goal to a certain extent.

The architecture of the client application is pictured by the class diagram of Figure 2. Every class is represented as a rectangle with its name in a top subrectangle and list of methods and in a bottom subrectangle. Edges between classes represent has by-value relations (solid diamond on the component

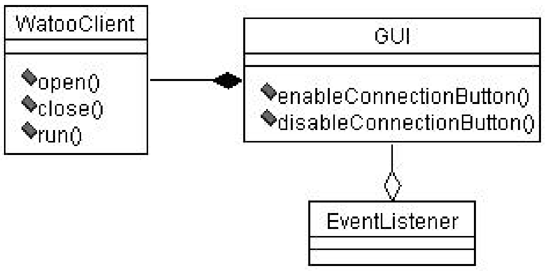


Figure 2: Architecture of the client application

side) and has by-reference relations (hollow diamond on the referee side).

In our diagram, there are three main classes of objects: WatooClient, GUI, and EventListener. This architecture follows the event delegation architectural pattern of Java (JDK 1.1). An important point is the fact that the application logic is separated from the graphical user interface, giving to them a certain independence. The application logic is in class WatooClient and the graphical user interface is in class GUI. An other important point is that there are sources of events and listeners of events. A source of event is typically a graphical user interface object (instance of GUI in our case). A listener is an object that provides a method called by the event source in response to a specific event that occurs at the interface (e.g., the click of a button). In our design, there is specialized (subclass) event listener for every type of event. The method called when an event occurs is polymorphic. An event listener may dispatch work to other components of the application. Hereafter, we describe two key mechanisms of the interface: connection establishment and voice communication.

The mechanism for the connection establishment with the server is illustrated in Figure 3. First, a user clicks on the button *Connect*. An event emerges from the graphical user interface that triggers an event listener. The event listener calls method *open()* on an object of class WatooClient. This method opens a stream transport connection with the server (using TCP), sends to the server the name of a satellite to track, requests disabling (enabling) of the connection (disconnection) button on the object of class GUI, and threads a copy of itself which, by definition, executes the method *run()*. The method *run()* embeds an endless loop that periodically receives from the server a message giving the coordinates of the tracked satellite.

Presently, not much is available in Java for supporting real time duplex voice communication [4]. The voice communication mechanism of the interface simply launches the communication freeware called Speakfreely. The executable code of Speakfreely is platform dependent. However, it is available for several platforms (Windows, various Unix systems). Therefore, the WATOO client application is portable to a platform as long as an executable of Speakfreely is available for that platform.

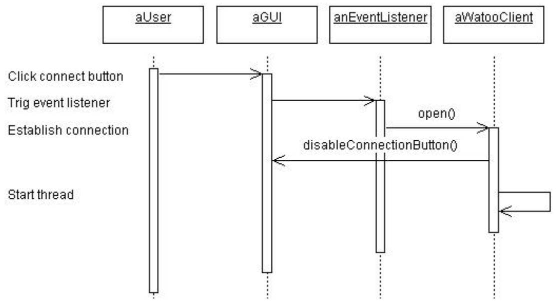


Figure 3: Connection establishment mechanism

3.2 Satellite telecommunications server

The server is responsible for:

- Determining the position in space of a satellite (the SSP and altitude) at a given time using Keplerian orbital elements data.

- Determining the direction in which the antennas should point, expressed in terms of azimuth and elevation.
- Positioning the antennas according to the computed direction.
- Tracking a given satellite, i.e., making that the antennas point towards a satellite.
- Setting the communication mode and frequency of the radio.
- Accepting a connection from a client.
- Periodically transmitting to a connected client the SSP and altitude, of a tracked satellite, and direction of antennas.
- Serving a voice connection with a client.

Since the satellite is constantly moving (for non geostationary orbits), the antenna rig needs to be repositioned periodically. A continuous update of the frequency is necessary to compensate for the Doppler shift effect.

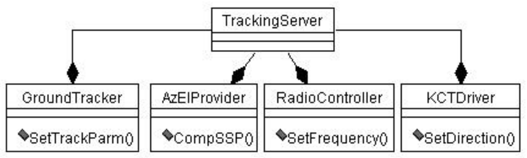


Figure 4: Architecture of the server

The architecture of the server is pictured in Figure 4. In the run time environment, every class is instantiated once. At the heart of this architecture is the TrackingServer that controls all related classes. The GroundTracker class is a computation center for the SSP and radius (distance between geocenter an satellite). The AzElProvider class is another computation center. It calculates the direction of the antennas, the altitude of the satellite, and the center angle[1] based on the SSP and radius as well as the location and altitude of the ground station. The RadioController class is the interface used to set the radio modes and frequencies, which is done using the parameters computed by GroundTracker and AzElProvider. The last abstraction is KCTDriver. It controls the antenna rotors. Given positioning parameters, it is responsible for moving the antennas in the proper direction.

A typical tracking session goes as follow. The object of class TrackingServer accepts a client Internet socket connection. Next, it receives a selected satellite name from the client. Afterwards, it reads the corresponding Keplerian orbital elements from a file and sends them to the object of class GroundTracker (using message SetTrackParm(), see Figure 5). Then, it enters and executes a loop until the socket connection with the client is broken.

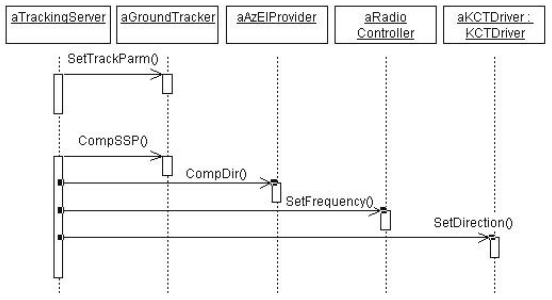


Figure 5: Main loop of the server

The loop of the server is illustrated in Figure 5. The participating objects are aTrackingServer, aGroundTracker, aAzElProvider, aRadioController, and aKCTDriver. There are five main steps. First, aTrackingServer sends message CompSSP() to aGroundTracker. It triggers computation of the current subsatellite point and radius. Second, aTrackingServer sends this information in message CompDir() to aAzElProvider that computes the current azimuth, eleva-

[1]The center angle is defined as the angle between two vectors originating from the geocenter, one directed toward the satellite and another one directed toward the ground station.

tion, altitude, and center angle. This second block of information is sent in message SetFrequency() to aRadioController (that sets the radio channel taking into account the Doppler shift effect) and, in part, in message SetDirection() to aKCTDriver.

Note that in this design, the control flow of the application (put in TrackingServer) is separated from objects that performs specific computations or tasks. This design has two advantages. The control flow is easier to grasp. And, controlled objects can be easily substituted by objects that offer the same services but in different ways (e.g., more accurately, faster, for a different hardware).

The server is implemented under the Linux platform using C++. We selected the freeware Linux for its performance and wide distribution. We selected C++ because the object-oriented design was easy to map this implementation language with good performance. Most of the top level objects are implemented as Unix processes communicating through Unix sockets. Although, this single user version of the server is essentially a sequential process, we went for a multiple processes structure because this server is the first phase of a more ellaborated project. We have foreseen development in a near future of a multiple users, radios, and antennas server that will require parallelisms within its components.

Keplerian orbital elements are automatically gathered daily through the FTP site at the Air Force Institute of Technology (AFIT). This feature has been implemented using the *cron* mechanism of Unix. All calculations are based on the two body model and equations described in Ref. [2].

4 Conclusion

We have presented the interface and the design of a software that provides access through the Internet to satellite telecommunications stations. The current version is single user and supports solely voice communication. Note that we are currently supporting only one type of rotor hardware interfaces (Gracilis Grace) and solely one type of radios (Yaesu). More work is required to support other makes of equipment.

We are presently working on a second client-server software that will support multiple users, various digital modes, and video. We are also sketching a network of distributed, interconnected, and cooperating satellite tracking stations. When a client will log on this system the latter will automatically switch him to a station that will provide the earliest available communication window with the desired satellite. And, upon closure of this window, the system will switch the client to the server that provides the next available communication window. Hence, the client will have access to longer communication time, automatically managed by the system.

Acknowledgments: The authors would like to thank Andre F. Lamothe, Gaston Sirois, and Denis Thibault whom precious collaboration has greatly contributed to the realization of this project.

References

[1] G. Booch, *Object-Oriented Analysis and Design with Applications*, Second Edition, Benjamin/Cummings, 1994.

[2] M. Davidoff, *The Satellite Experimenters Handbook*, The American Radio Relay League, 2nd Edition, 1994.

[3] G. Cornell and C. S. Horstmann, *Core Java - Second Edition*, The Sunsoft Press, Java Series, 1997.

[4] http://www.pentium.com/ial/jmedia.

[5] B. Selic, G. Gullekson, and P. T. Ward, *Real-time Object-oriented Modeling*, John Wiley & Sons, Inc., 1994.

[6] http://www.fourmilab.ch.