Object-Oriented Modeling of a Satellite Tracking Software*

M. Normandeau and M. Barbeau (ve2bpm)
Department of mathemat ics and computer science
University of Sherbrooke
Sherbrooke, Québec CANADA J1K 2R1
{normand, barbeau}@dmi.usherb.ca

Abstract

This paper presents a case study of an object-oriented development of a satellite tracking software. It is designed following the Real-Time Object-Oriented Modeling (ROOM) methodology. The design resulting from the application of ROOM is implemented i n C++ on the QNX platform. Concurrent actors are naturally mapped to parallel processes. ROOM yields a modular architecture which is clear, reusable, and maintainable. Use of QNX leads t o a highly performant and reliable system . This work is important because it shows application of advanced software engineering principles in a field where most of the development is still based on structured (and non-structured) techniques.
Key words : Object-Oriented Modeling, Real-Ti me Systems, Satellite Tracking.

1 Introduction

The problem addressed in this paper is the development of a software for ground tracking in real-time of non geostationary satellites in circular or elliptical orbits [2]. Tracking is necessary for communicating through the satellites. Tracking of a satellite means determining its position in space, when it is accessible (*in range*), and where the antennas should be pointed. Prediction of access time is required because, for a given ground station, satellites are in range only during certain periods (*satellite passes*) during a day. Antennas are directional and determination of pointing direction is necessary and expressed in terms of two parameters: azimuth and elevation. Software inputs are the orbital element sets of satellites being tracked, Greenwich Mean Sidereal Time for January 00.00Z of each year, and latitude and longitude of a ground station.

This paper presents a case study consisting of the application of an object-oriented methodology to the development of a satellite tracking software. This experiment is not limited to a simulation but is a complete development from analysis to implementation in an actual tracking station. Development of the software follows the Real-time Object-Oriented Methodology (ROOM) [4]. ROOM is based on an operational actor model. Actors are active entities of a model. Each

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

actor consist of data, behavior, and structure. Its execution is concurrent with other actors and it communicates with them by means of messages. The actor model supports the basic elements of the object-oriented paradigm such as abstraction, encapsulation, inheritance, aggregation, typing, and concurrency [1]. Employment of such a methodology leads to high quality designs which are very well documented, understandable, and easy to communicate as well as a evolve.

This paper is structured as follows. Section 2 presents the object-oriented model. The implementation of this model is discussed in Section 3. We conclude with Section 4.

2 Object-Oriented Model

This section details the object-oriented ROOM model devised for the satellite tracking problem. As the ROOM methodology dictates, our satellite tracking software model is made of actors. These actors represent the main active and concurrent components of the system. For each of them, a structure and a behavior are specified. They communicate among themselves and with the outside world through ports which are references to specific communication protocols. These protocols are sets of messages that actors intend to exchange with one another. The structure presents the components which make an actor and how they are related to one another. Components can be either actors, ports or bindings which are links between pairs of ports. Diagrams may be used to illustrate the structure and they represent actors as white boxes, ports as squares on the boundaries of actors, and bindings as lines between pairs of ports (see Fig. 1). On the figure, only the names of the actors and ports are displayed.

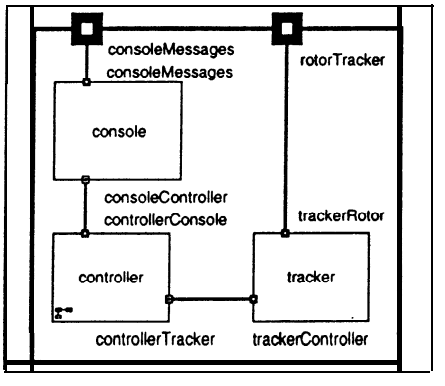

Figure 1: Tracking system structure

A behavior specification is made of states and events. An actor moves from state to state when events are triggered. Diagrams may illustrate the behavior of actors and they represent states as rounded boxes and events as arrows (see Fig. 2). The black circle at the top left corner of the diagram leads to the first state when the system starts.

Our model is made of three main actors : a *console,* a *controller*, and a *tracker (see* Fig. 1). The console is the interface between the user and system. It is responsible for offering possible actions to the user, getting his inputs, and sending them to the controller. The structure of *console* is fairly simple as it contains only two ports. The first one, *consoleUser*, is used to communicate with the user. It refers to the *ConsoleMessages* protocol which specifies the possible messages the

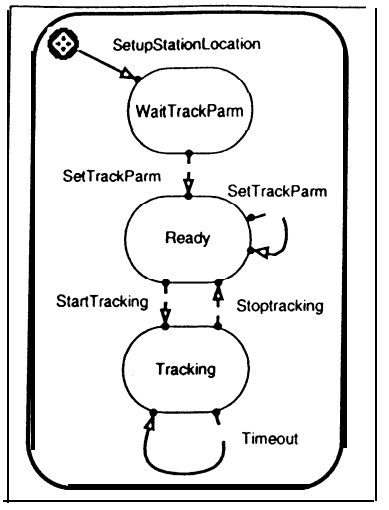

Figure 2: Controller behavior

user can send to *console* and vice versa. The actions available to the user are: *setup a tracking run, start a tracking run, stop a tracking run,* and *quit.* While setting up a tracking run, the user is also asked for a choice of a satellite. The set of actions offered to the user depends on the state of *console.* For example, if no tracking run is executing, it is impossible to stop one. The second port, *consoleController*, allows communication with the *controller* through the *ControllerMessages* protocol. It mainly relays user's decisions to the *controller.*

The second actor, *controller,* executes the user's decisions received from *console. Once* the system is tracking a satellite, it is also responsible for providing to *tracker* the direction in which the antenna should point. The *controller* contains component actors which are : *groundTracker* and *azElProvider* (see Fig. 3). Their own description will be provided further on. The *controller* receives messages from *console* that drives its behavior (see Fig. 2). Its main operations are to set the station's parameters, to initialize a tracking run, and to compute the azimuth and elevation of the antenna at specified intervals. The *tracker* has one port, *ControllerConsole*, referencing the *ControllerMessages* protocol allowing communication with *console.* It also has another port, *controllerTracker*, referencing the *TrackerMessages* protocol through which messages are sent to *tracker.*

The last actor at this level is *tracker* which is a driver to the rotor's interface card. **It** receives the azimuth and elevation from *controller* through its *trackerController* port. It then sends it to the rotor's interface in a way it can comprehend it using its *trackerRotor* port. These ports are the only components of the *tracker* actor. The Rotor protocol referenced by the *trackerRotor* port represents the driver specification of the rotor interface.

Explosion of *controller* uncovers two *sub-actors (see Fig. 3): groundTracker* and *azElProvider.* These embedded actors communicate through ports *controllerGround Tracker* and *controlle rAzEl-Provider.* The *groundTracker*, given orbital elements and current GMT, is responsible for providing the Sub-Satellite Point (SSP) and altitude of a satellite. It encapsulates the satellite-orbit model described in Ref. [2] This communication with *controller* is done via the *controllerGround Tracke r* port. The *azElProvider*, given the SSP, altitude, and tracking station's coordinates (latitude and

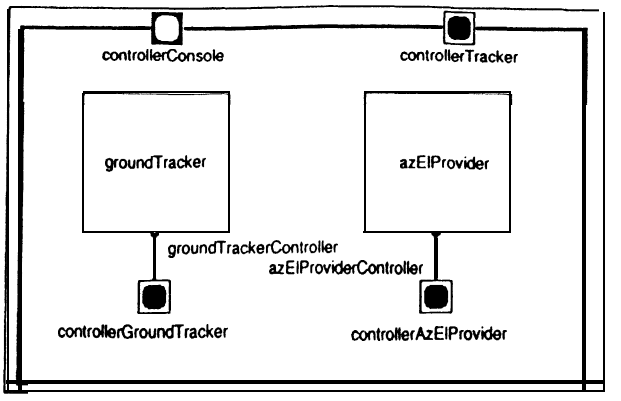

Figure 3: Tracking controller actor

longitude), is responsible for providing the azimuth and elevation of the antenna. The exchange of messages with the *controller* is done through the *azElProviderController* port. The *controller*, as its name implies, coordinates its components and communicates with *tracker* and *console*. It gets inputs from *console* and sends outputs to *tracker*. In order to promote modularity and reuse of actors, the components of *controller* communicate only with their container and not between themselves. This way, module coupling is greatly reduced and reusability is enhanced.

The controller behaves as pictured in Fig. (see Fig. 2). The transition between the initial state (black circle) to state *WaitTrackParm* reads the station's latitude and longitude and transmits this information to *azElProvider*. The transition (either from *WaitTrackParm* or *Ready* to *Ready)* labeled *SetTrackParm*, is triggered by the *SetElements* message from *console* that conveys orbital elements for a selected satellite. The transition labeled *StartTracking* from *Ready* to *Tracking* is triggered by the *StartTracking* message from *console* and begins a tracking run. The transition labeled *Timeout* is triggered on a periodic basis and initiates the process of updating the direction of the antenna. A tracking run is halted when the transition labeled *StopTracking* is triggered.

3 Implementation

In this section, we discuss the implementation of our software. Issues are the hardware and Operating System (OS) on which our software must execute, programming language in which our application is coded, and mapping of concepts of our design model to OS and programming language concepts. Let us first consider the hardware. We have the following pieces of' equipment: a Pentium class micro computer, a V/UHF all mode transceiver, a multi- mode digital signal processor, an azimuth-elevation rotor along with its computer interface, a VHF antenna, and a UHF antenna.

We have chosen the QNX operating system which has been developed specifically for real-time applications. It supports multitasking and fast context switching. QNX has a micro kernel architecture which means that its kernel is light enough to reside in the processor cache. Consequently, t is very performant. Several processes can run concurrently and **QNX** provides message based nterprocess communication primitives (send, receive, and reply). The programming language selected for the implementation is C++ because, conceptually, it is the closest to the ROOM model

amoung those available on QNX.

Actors of the ROOM model can be mapped either to QNX concurrent processes or C++ sequential objects. Note that in the ROOM model, there are actors that can potentially run in parallel (such as *console, controller,* and *tracker)* whereas others run purely in sequence with respect to each other. (such as *controller, groundTracker*, and *azElProvider*). On the one hand, actors that have the potential to run in parallel are mapped to processes communicating with the QNX message passing primitives. On the other hand, every group of actors that run in sequence is mapped to a single process. Each actor becomes an object and communicates with the other actors in the group with C++ method calls. This avoids the overhead that results from the calls to OS primitives thus increasing efficiency. Hence, the group of actors *controller, groundTracker*, and *azElProvider* is mapped to a single process whereas the actors *console* and *tracker* are both individually mapped to a process. The resulting implementation therefore results in three processes. These implementation strategies result in a simple and efficient organization.

4 Conclusion

This paper has presented the object-oriented developrnent of a real-time satellite tracking system. It illustrates application of object-oriented design principles in the field of satellite telecommunications. Use of a methodology such as ROOM leads to a very well structured software that makes it easy to understand and maintain. This project is not only a simulation but a full development including a working implementation based on the QNX operating system running applications with real-time performance. The demonstration performed in this project is important because it shows the relevance and suitability of state of the art software, development techniques and tools in a field where classical structured (and non structured) soft-ware development techniques are still largely employed.

Future work in. our project includes development of a graphical user interface, remote control through TCP/IP of the station, automatic control of the transceiver, and improvernent of the satellite-orbit mathematical model.

Acknowledgments The authors would like to thank Francis Bordeleau from Carleton University for fruitful discussions about the ROOM model of our satellite tracking software.

References

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

[2] M. Davidoff. *The Satellite Experimenter's Handbook.* The American Radio Relay League. Second Edition, 1994.

[3] QNX Software Systems Ltd. *QNX System Architecture. 1993.*

[4] B. Selic, G. Gullekson, and I? T. Ward. *Real-time Object-Oriented Modeling.* John Wiley and Sons, Inc. 1994.