*Astronomical Data Analysis Software and Systems XVII* **E.40**
*ASP Conference Series, Vol. XXX, 2008*
*J. Lewis, R. Argyle, P. Bunclarck, D. Evans, and E. Gonzales-Solares, eds.*

# Ice middleware in the New Solar Telescope's Telescope Control System

Sergiy Shumko

*Big Bear Solar Observatory, New Jersey Institute of Technology, 40386 North Shore Ln, Big Bear City, CA 92314, USA*

**Abstract.** The Big Bear Solar Observatory (BBSO) is now in process of assembling and aligning its 1.6 m New Solar Telescope (NST). There are many challenges controlling NST and one of them is establishing reliable and robust communications between different parts of the Telescope Control System (TCS). For our TCS we selected Ice (Internet communication engine) from ZeroC, Inc. In this paper we discuss advantages of the Ice middleware, details of implementation and problems we face implementing it.

## 1. Introduction

NST is an off-axis solar telescope with the world largest (1.6–meter) aperture. The BBSO is currently finishing installation of the NST. We expect first engineering light in the early 2009.

NST will be equipped with an active optics (AcO) system, which will control the shape of the primary mirror M1 with the help of 36 actuators. AcO will also control position of the M2 mirror.

To control all aspects of operations of the telescope we developed a TCS which consists of several major parts: Telescope Pointing and Tracking Subsystem (TPTS), Active Optics Control Subsystem (AOCS), Handheld Controller and main GUI. The Thermal Control Subsystem is being developed also. These systems run under several operating systems: MS Windows XP, Fedora Linux and Windows Mobile 5. Also in our team of software developers we mainly use two programming languages: Java and C++. Thus we have a heterogeneous distributed control system. To bind all these parts into a TCS we selected the Ice middleware platform developed by ZeroC, Inc[1].

## 2. Ice Advantages

The most important Ice advantages are:

- Ice provides fast and scalable communications.
- Ice is simple to use. Basic Ice application can be written in just a few days after studying the manual. The Ice manual covers all aspects of middleware programming and is well written.

[1] `http://www.zeroc.com/ice.html`

- Ice supports Windows platform, including Windows Vista, as well as Linux platform. Ice also supports many programming languages, including C++ and Java.
- ZeroC has developed Ice Embedded (Ice-E), which supports Microsoft Windows Mobile operating system for handheld devices.
- ZeroC provides source code of Ice under the GNU General Public License (GPL).
- ZeroC continuously updates Ice, fixing problems and adding more platforms/languages. They provide good customer support through developers forum.

Overall, Ice is now mature and solid middleware package.

We would like to add few remarks here. The Ice project was first publicly released in 2003-2004. We started working with it in 2005. At that time Ice supported only Java, C++ and Python. Now it supports much more, including *Ruby*. If we were to start development today we would have considered *Ruby* as a language of choice. Also, having source code available makes customization of Ice possible. For example, we had to modify Ice-E source code to add registry service to make our communication protocol uniform across our platforms.

## 3. Ice Problems

We have not yet encountered any major flaws in Ice. Ironically, one of Ices advantages is a flaw for us: frequent package updates. When we started the TCS development in the early 2005, ZeroC released version 1.5 of Ice. As of October 2008 latest version is 3.3. Often these new releases require updates in our source code. Sometimes changes are major and require major updates in the Ice part of our code.

It is not possible to ignore those updates because ZeroC may eventually drop support for older versions or older versions of compilers. Also ZeroC often adds useful features to Ice. Thus we have to keep up with the updates and one or two times a year revise our own code. Sometimes we can skip minor revisions.

## 4. TCS Implementation

We decided in favor of a star-like structure of the TCS. All messages will be directed through a central communication hub, which we call *Headquarters* (HQ).

This structure has several advantages:

1. Subsystems do not need to know details of the TCS, such as the locations of other subsystems or which instruments a particular subsystem serves. They only need to know the name of the instrument they want to send a message. All work of routing communications lies solely on HQ.
2. All events are logged by HQ, which allows monitoring TCS activity from one central location.
3. Adding new instruments or subsystems becomes a relatively easy task since all that is required to do is to simply add another routing rule.

We also decided that each TCS subsystem acts as a server and as a client at the same time and each subsystem uses the same Ice interface. An Ice inter-

face is a set of operations performed on an Ice object. We actually went even further and use same interface to communicate between different parts inside each subsystem.

The interface for every object consists of seven operations or methods: *Register*, *Unregister*, *SendError*, *SendCommand*, *RequestInformation*, *SendNotification*, *SendReply* and an exception, called *Error* to inform a remote side if an error occurs during the operation execution. All parameters of the above operations have type string or a character sequence. In most cases the parameter is represented by an XML string. The usage of *Register* and *Unregister* methods allows us to solve problem of the bootstrap order. We can start subsystems in any order; they only need to register themselves with HQ and with the Ice locator service, called *IcePack* registry. If HQ was not started then they will register with HQ later after HQ starts and sends its own *Register* message.

We also developed a software for a PocketPC device which we use as a *Handheld Controller*, to manually control simple dome and telescope operations via Wi-Fi network connection. The *Handheld Controller* uses same Ice interface to communicate with TCS because ZeroC also provides Ice-E for Windows Mobile. We had to modify Ice-E source code to add support for the *IcePack* registry since Ice-E is a stripped down version of Ice to decrease its memory footprint and normally does not include most of the additional services.

For more details on TCS implementation please see Shumko & Yang (2006).

## 5. Conclusion

We were using Ice in our projects for almost 4 years. We can confirm that ZeroC's Ice is a good choice for distributed control systems. The choice of Ice as the communication middleware platform has not only saved us development time, it also resulted in a more robust and extensible system. We did not see negative impact on the systems performance, too.

**Acknowledgments.** This work was supported by NSF under grants ATM 00-86999, ATM 02-36945 and by NASA under grant NAG 5-12782.

We would like to thank all the groups, companies and others who provide their software for free use.

## References

Shumko, S. & Yang, G. 2006, Proc. SPIE, 6274, 62741Q