Hindawi Publishing Corporation
Advances in Astronomy
Volume 2010, Article ID 496765, 8 pages
doi:10.1155/2010/496765

Review Article

Protocols for Robotic Telescope Networks

Alain Klotz[1,2]

[1] *Observatory of Haute-Provence, Route de l'observatoire, 04870 Saint-Michel l'Observatoire, France*
[2] *CESR, University of Toulouse, 9 avenue Colonel Roche, 31028 Toulouse Cedex 4, France*

Correspondence should be addressed to Alain Klotz, alain.klotz@free.fr

Received 11 June 2009; Accepted 28 December 2009

Academic Editor: Alberto J. Castro-Tirado

Autonomous robotic observatories can use modern communications to receive pertinent information from institutes that generate events to observe (e.g., supernovae, near-earth asteroids, gravitational lensings, and gamma-ray bursts). This paper is addressed to astronomers who are not specialists in computer science. We give explanations of some basic and advanced protocols to receive events and how to implement them in a robotic observatory software. We describe messages such as GCN notices, VOEvents or RTML, and protocols such as CGI, HTTP, SOAP, RSS, and XMPP.

1. Introduction

Autonomous robotic observatories (AROs) use complex software to control devices and to check the security conditions (weather, power supply, and mechanical sensors). This software must exchange messages with its different parts and with external resources. When an ARO is not connected to another one, the messages can use any language, but for decade, there have been coordinated efforts to standardize communication protocols [1]. The ultimate goal is to send alert messages to telescope networks that can be understandable by any observatory within the network. The scientific goal of observatory networks is to maximize the observation efficiency in order to characterize the nature of objects recently discovered (e.g., asteroids, supernova, gamma-ray bursts) or to perform optimized follow-ups (e.g., gravitational lensing and variable stars). In this paper, we focus on protocol techniques usually used by telescope networks in the context of the Virtual Observatories (VOs). These protocols are mostly defined and maintained by the International Virtual Observatory Alliance (http://www.ivoa.net/) (IVOA) and the scope of this paper is to present them to the astronomers that are not specialists in computer science.

2. Agents, Servers, Clients

In the context of VO, we define an agent as a software or a library that carries out a predefined task of an ARO (e.g., in [2]). For example, the telescope agent is a software which can drive the telescope mount.

Generally, agents are slaves, that is, waiting for basic orders. In the context of computer science, they are named servers. A server is an agent that waits forever until an order is received. When an order is received, the corresponding task is executed, and the agent returns to the waiting loop. All server agents are managed by a more intelligent agent that collects the status of the observatory and the schedule of observations and decides to send orders to the server agents. This intelligent agent is a client in the context of the computing science (see Figure 1). First, the intelligent agent establishes client connexions to the server agents. Second, the intelligent agent sends messages to server agents in an order corresponding to the sequence of observations scheduled. A good example of ARO agents is the suite RTS2 [3] which can be adapted to many hardware configurations. An example of a more complex telescope network and its agents (RoboNet-II) is described in [4].

3. What Is a Protocol?

Computer science uses the concept of Open Systems Interconnection Reference Model (hereafter OSI model) to define how to communicate between computers. The OSI model defines seven layers: physical, data link, network, transport, session, presentation, and application. The three first layers are related to the media used to send data (Internet Protocol,

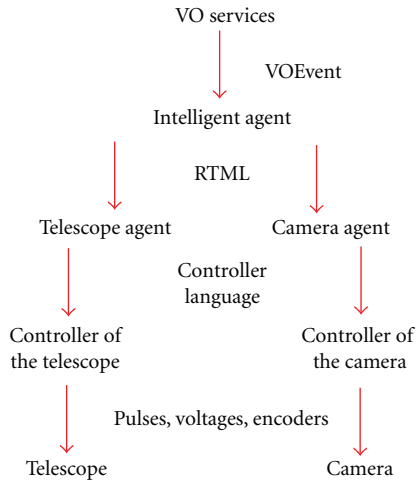

Figure 1: Simple schema of agents used in an autonomous robotic observatory.

Point to Point Protocol, cables, hubs, etc.). The four last layers concern how data are managed in the computer. ARO agents use protocols defined essentially in the application layer. In this paper, we will use the following protocols: CGI, HTTP, SOAP, RSS, and XMPP.

The application layer protocols define how the messages are transported to the other agents, but the contents of the messages are written in a given language. In this paper, we use the following languages: HTML and XML.

The combination of the application layer protocol and the message language defines the protocol as defined in the title of this paper. Let us take an example. The telescope agent is waiting for a message using an event loop which is based on a TCP socket server. At a time, the TCP socket receives the following message from the intelligent agent:

```
<Coordinates>
  <RightAscension>41.25</RightAscension>
  <Declination>+22.5</Declination>
</Coordinates>
```

This message is written in the eXtensible Markup Language (XML) and is understandable by an astronomer. This is a text language with a syntax rule based on start-tags (<>) and corresponding end-tags (</>). The tag `Coordinates` implies that the following coordinates stand for the next target to observe. The parameters are the Right Ascension and the Declination. The corresponding values lie in the start/end tags RightAscension and Declination respectively. The tag Declination could be given before that of RightAscension without any difference of interpretation. If the telescope controller is based on the LX200 Meade protocol, the telescope agent must convert the coordinates to the language of the controller (the full list of the LX200 Meade commands can be found in http://www.meade.com/support/CommandSet.html):

```
:Sr02:45:00#
:Sd+22*30:00#
```

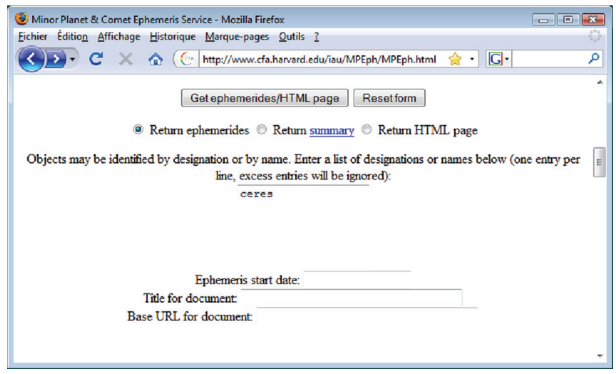

Figure 2: Web form of the Minor Planet Center ephemeris center as seen using a browser.

Then, these two commands must be send to the RS232 socket of the telescope controller. These low level commands depend on the manufacturer of the telescope controller and the role of the telescope agent is to transform commands from a standard form (cf. the XML text above, and see Section 6 for the RTML definition) to the low level command language specific to the telescope controller (e.g., the LX200 Meade language).

4. Protocols to Prepare Observations

In a full ARO, the schedule of observations is generated by a schedule agent that sorts the observations to maximize a criterion related to the scientific results [5]. It is also important to simplify human data inputs of observation requests to avoid human errors when entering target coordinates. For example, it is preferable to ask for an asteroid name rather than ask for its coordinates that change all the time. In this section, we show how to incorporate two simple name solver codes in an ARO agent.

4.1. CGI and HTTP. In this section, we present a classical web service delivered by the Minor Planet Center (MPC). MPC provides on-line ephemeris for minor planets. First, connect to the MPC server using a web browser (Figure 2). The Uniform Resource Locator (URL) is the address of the MPC server computer that delivers the web page. This page is written in Hyper Text Language Markup (HTML) which is based on start and end tags (e.g., <HTML> and </HTML>). In the case of the MPC web service, an excerpt of the web form is defined by the HTML code in Algorithm 1.

The interpretation of this code by the browser gives what can be seen on Figure 2. The main tag is <FORM>. It defines web gadgets (hereafter widgets) as buttons, radio buttons, text areas, and so forth. The contents of these widgets will be send to the URL associated parameter ACTION in the tag FORM using the method POST. This URL triggers a Common Gateway Interface script (CGI) in the MPC server when the user pushes the submit button. To summarize, data format is CGI-POST and the protocol is HTTP.

```
<FORM
  METHOD=POST
  ACTION="http://scully.cfa.harvard.edu/~cgi/MPEph2">
  <input type=submit value="Get ephemerides/HTML page">
  <input type=reset value="Reset form"><br>
  <input type="radio" name="ty" VALUE="e" CHECKED>
  <input type="radio" name="ty" VALUE="s">
  Return <a href="#summary">summary</a>
  Return ephemerides
  <input type="radio" name="ty" VALUE="h">
  Return HTML page<br>
  Objects may be identified by designation or by name.
  Enter a list of designations or names below (one
  entry per line, excess entries will be ignored):<br>
  <textarea name="TextArea" cols=17 rows=5></textarea>
  <br>Ephemeris start date: <input name="d" maxlength=20 size=17 VALUE="">
  <br>Title for document: <input name="tit" maxlength=60 size=40>
  <br>Base URL for document: <input name="bu" maxlength=80 size=40>
</FORM>
```

Algorithm 1

Obviously, an ARO must be able to send a CGI form without any human action (i.e., without a web browser). How to do it? The software of the ARO must use a code that simulates the web form displayed by the browser. The corresponding code to obtain ephemeris of Ceres is given by the seven following lines written in the Tcl script language (see Algorithm 2) (documentation available at http://www.tcl.tk/. Tcl/Tk is installed in most Linux distributions. Type tclsh to launch a Tcl interpreter.):

At the end of the script, the variable `html_text` contains the HTML code of the result. The next action to do is to parse the HTML text to extract the coordinates of the minor planet.

4.2. SOAP and HTTP. The Simple Object Access Protocol (SOAP) is close to the CGI-POST. It can be used with HTTP. Let us take an example. We want to use the web service of the Centre de Données Stellaires (CDS) to find coordinates associated to an object name. This name solver is not usable by a browser because the SOAP protocol has no graphical interface in a web browser. Nevertheless, SOAP is easy to use in an ARO agent and SOAP can be better than CGI-POST for many technical reasons that are outside the scope of this paper. As an example, the following lines are written in the Tcl script language to solve the coordinates of Messier 51 (see Algorithm 3).

At the end of the script, the variable xml_text contains the XML code of the result. The next action is to parse the XML text to extract the coordinates of the galaxy. The lines in Algorithm 4 display the XML code returned in the variable `xml_text`.

The URL defined in the parameter xsi:noNamespaceSchemaLocation gives the structure of the tags used in the document. The coordinates in decimal form lie in the tags `jradeg` and `jdedeg`. XML format is much more easy to parse than the HTML code like that returned by the CGI-POST of Section 4.1 because the coordinates to extract from the HTML code is not necessary defined inside start/end tags and the structure of the returned HTML code is not auto-documented as it is the case for XML.

5. Protocols to Receive Events

Just after the discovery of Gamma-Ray Burst (GRB) optical afterglows in 1997 [6] BACODINE was created, an internet message distribution service that became later the Gamma-ray burst Coordinate Network (GCN, [7]). The data input of this service is information delivered by satellites that are equipped by a real-time localizer of GRBs (e.g., Swift, Integral, and Fermi). The useful output of the GCN for ARO is notices. (see http://gcn.gsfc.nasa.gov/invitation.html for a complete description) For each GRB triggered by a satellite, the GCN builds a notice that indicates coordinates, error box and summarizes miscellaneous informations about the characterization of the burst. These notices are generated automatically when a new trigger appears. The distribution is done freely by a simple TCP socket connexion to any ARO that asks to receive notices.

5.1. GCN Notices and TCP Sockets. A TCP socket is one of a simplest way to exchange messages between software. The principle is based on the client-server connexion. First, the software that wants to receive messages opens a port. This is the server and the port is an integer number between 0 and 65535. The other software that will send messages opens a connexion to the Internet Protocol (IP) number at the port number of the server. This is the client. For the GCN service, ARO are the servers and the GCN opens client connexions and sends notices by this way. The GCN must know who are servers to reach. So, it is mandatory to register the IP+port numbers of the ARO agent to the GCN. Then, the ARO agent opens the server socket port and waits for the GCN notices. Note that the port number is attributed by the GCN (the list

```
set minor_planet "ceres"
set url "http://scully.cfa.harvard.edu/~cgi/MPEph2"
package require http
set query [::http::formatQuery TextArea \
  "$minor_planet" ty "e" d "" tit "" bu ""]
set token [::http::geturl $url -query "$query"]
upvar #0 $token state
set html_text $state(body)
```

Algorithm 2

```
package require SOAP
set name "M51"
set resultType "xp"
set server "http://cdsws.u-strasbg.fr/axis/services/Sesame"
SOAP::create sesame -uri $server -proxy $server \
  -action "urn:sesame" \
  -params { "name" "string" "resultType" "string" }
set xml_text [sesame $name $resultType]
```

Algorithm 3

of registered ARO is http://gcn.gsfc.nasa.gov/sites_cfg.html) (e.g., port=5134 for the TAROT telescope).

In GCN notices delivered by TCP sockets, data are coded in a binary stream of 40 integers of 4 bytes each. So, a total of 160 bytes must be received before begining to decode the notice parameters. The corresponding code in Tcl script is shown in Algorithm 5.

In this code, the procedure `socket_accept` is called when the TCP channel is opened. Note that this channel is configured to be activated only when 160 bytes are received. The `fileevent` function creates a call back to the procedure `socket_read` when data are arrived in the opened channel:

```
proc socket_read { channel } {
  set line [read $channel 160]
  if {[eof $channel]} {
      close $channel
  } elseif {![fblocked $channel]} {
    binary scan $line I* longs
    set datas [notice_parser $longs]
    return $datas
  }
  return ""
}
```

The procedure `socket_read` reads the data in the channel and stores them in the variable `line`. Then it transforms this `line` into 40 integers in the list variable `longs`. Now one must call the procedure `notice_parser` to retrieve the celestial coordinates of the GRB (see Algorithm 6).

The rules of parsing are defined in a GCN document. (the notice definition can be download at http://gcn.gsfc.nasa.gov/sock_pkt_def_doc.html.) The type of notice (pkt_type) is the most important parameter to decode. The type is related to the source of the trigger (e.g., Swift BAT triggers are of pkt_type=61). The procedure `notice_parser` returns the minimal informations: the type of the notice and celestial coordinates. Obviously, a real implementation of the parser should include other informations from the notice in order to conclude if the trigger deserves to be observed or not. The last Tcl function `vwait forever` must be used to activate the event loop of the socket listener.

5.2. Limitations of Using GCN Notices with TCP Sockets. The advantage of using TCP sockets is that the code is easy to write in any programming language. Almost all the early optical emissions of GRBs were found thanks to this system. However, there are several disadvantages:

(1) Most institutes do not authorize the opening of the GCN port due to security reasons. The network administrator of the institute must configure the firewall to authorize only the IP numbers of the GCN (128.183.16.187, 128.183.96.236, and 128.183.96.237) to connect on the server port. Unfortunately, not all institute authorize to do that. One can skirt the problem by using of a Virtual Private Network (VPN).

(2) Parsing the GCN notice parameters depends on each type of alert (more than 100 are defined in 2009). The parser must treat all cases to be robust. The arrival of a new satellite implies to add codes.

(3) The GCN computer must verify connections to tens of computers. This excellent work is made by Scott Barthelmy but it is almost impossible to extend the system to thousands ARO.

```
<?xml version="1.0" encoding="UTF-8"?>
<Sesame xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
  "http://vizier.u-strasbg.fr/xml/sesame_4.xsd">
<Target>
  <name>M51</name>
  <Resolver name="S=Simbad (CDS, via client/server)">
    <otype>Sy2</otype>
    <jpos>13:29:52.36 +47:11:40.8</jpos>
    <jradeg>202.46820833</jradeg>
    <jdedeg>47.19466667</jdedeg>
    <refPos>1999ApJS..125..409C</refPos>
    <errRAmas>10800</errRAmas><errDEmas>10800</errDEmas>
    <Vel><v>600</v><q>D</q><r>2004A&amp;A...422...39S</r></Vel>
    <MType>Sc</MType>
    <oname>4C 47.36A</oname>
    <nrefs>2092</nrefs>
  </Resolver>
</Target>
</Sesame>
```

Algorithm 4

```
set channel [socket -server socket_accept 5134]

proc socket_accept { channel ip port } {
  fconfigure $channel -buffering full -translation binary
  fconfigure $channel -encoding binary -buffersize 160
  fileevent $channel readable [list socket_read $channel]
}
```

Algorithm 5

The solution is to replace the 160 byte notices by an XML file designed specifically to define an event. In the context of VO, these are VOEvents.

5.3. GCN Notices versus VOEvents. A VOEvent is an XML file (http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/IvoaVOEvent). VOEvent specifications are maintained by the IVOA. The structure of a VOEvent is

```
<voe:VOEvent>
  <Who></Who>
  <What></What>
  <WhereWhen></WhereWhen>
  <How></How>
  <Why></Why>
  <How></How>
</voe:VOEvent>
```

Applied to the GCN notices, a VOEvent should indicate the notice type in the tag What:

```
<What>
  <Param name="Packet_Type" value="61"/>
</What>
```

The celestial coordinates lie in the tag WhereWhen; see Algorithm 7.

The access to the coordinates seems not trivial but the rules of decoding are stored in a file defined in the tag voe:VOEvent by the parameter xsi:schemaLocation (see Section 4.2 for a similar case). A complete parser of VOEvent is difficult to develop but there is no need to update it when a new type of notice is sent.

5.4. VOEvents and XMPP. VOEvents could be broadcasted using TCP socket but it does not solve the problem of managing thousands of connexions by the broadcaster nor the authorization problems. There are many solutions in computer science but the main constrain is that ARO agents must open a client instead of a server connexion to solve the problems of authorizations. However, in this case, the problem is that the broadcaster (i.e., the GCN) cannot connect itself to the ARO agents. One must find a mechanism that allows ARO agents to be informed when a new VOEvent can be downloaded somewhere.

One solution is to use Really Simple Syndication (RSS). The ARO agent download, periodically, a file defined in a fixed URL (for GCN an RSS URL is http://www.estar.org.uk/voevent/GCN/GCN.rdf). This is an XML file that contains two important informations: the tag `PubDate` which indicates when the last notice was created, and the tag

```
proc notice_parser { longs } {
  set pkt_type   [lindex $longs 0]
  set burst_ra   [expr [lindex $longs 7]*0.0001]
  set burst_dec  [expr [lindex $longs 8]*0.0001]
  return [list $pkt_type $burst_ra $burst_dec]
}
vwait forever
```

Algorithm 6

```
<WhereWhen>
  <ObsDataLocation>
    <ObservationLocation>
      <AstroCoord coord_system_id="FK5-UTC-GEO">
        <Position2D unit="deg">
          <Name1>RA</Name1>
          <Name2>Dec</Name2>
          <Value2>
            <C1 pos_unit="deg">189.0740</C1>
            <C2 pos_unit="deg">32.9846</C2>
          </Value2>
        </Position2D>
      </AstroCoord>
    </ObservationLocation>
  <ObsDataLocation>
</WhereWhen>
```

Algorithm 7

`link` that indicates the URL of the last VOEvent file to download. In the case of GRBs, the ARO agent should to be informed with a frequency higher than 1 Hz. This is generally a frequency too high for an RSS flux (but RSS remains a good solution for low frequency events as gravitational lensing, e.g.).

A better solution comes from the eXtensible Messaging and Presence Protocol (XMPP). This is an open protocol used for instant messaging. This is not very different than for the popular SkyPE or MSN Messenger but XMPP is open source! An ARO agent opens a client connexion to an XMPP server. The XMPP client is identified and must subscribe to a feed. There is such a server at Caltech (see informations in http://voeventnet.caltech.edu/software/index.html). This server also proposes a Java client named `voeclient.jar` that can be included easily in any code. A configuration file allows to choose the GCN notices in a VOEvent format; (see Algorithm 8).

The XMPP client is run using a Java Runtime Environment (JRE) available for any operating systems. Before starting, one must create an account by:

```
java -jar voeclient.jar -n myname
 mypassword client.properties
```

And one must subscribe to the GCN VOEvents by

```
java -jar voeclient.jar -s home/moriori.
 cacr.caltech.edu/egcn \
  client.properties
```

Then, the following call starts the listener of VOEvents:

```
java -jar voeclient.jar -r
 client.properties
```

`voeclient.jar` generates the file feed.log where one can follow the arrival of new VOEvents. Unfortunately, the original code of `voeclient.jar` does not write the VOEvent itself on the disk but the code is open source and it is easy to write code just after the comment "// ADD CODE HERE TO PROCESS PACKETS" of the VOEventJabberClient.java code. Note that this code is also able to send a VOEvent.

6. RTML

Remote Telescope Markup Language (RTML, see (http://www.astro.physik.uni-goettingen.de/hessman/) [8, 9]) is usualy used to communicate between agents of a given ARO. RTML defines an XML syntax to describe observation parameters like instrument setup. RTML is different from VOEvent. VOEvents define the science case of objects to observe. Then, a VOEvent must be treated by an agent who parse the astronomical parameters to generate RTML messages knowing the instrument characteristics of the observatory. In the case of a telescope network, this agent must generate RTML messages for each telescope. The power

```
# Properties file for VOEventJabberClient
jabber.id=ARO Agent
jabber.server=moriori.cacr.caltech.edu
jabber.feed=home/moriori.cacr.caltech.edu/egcn
user.name=myname
user.pass=mypassword
packet.validate=false
schema.location=c:/d/voeventnet/VOEvent-v1.1.xsd

# Properties for logging
log4j.rootLogger = INFO, stdout, logfile
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
log4j.appender.logfile=org.apache.log4j.DailyRollingFileAppender
log4j.appender.logfile.File=feeds.log
log4j.appender.logfile.DatePattern='.'yyyy-ww
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
```

Algorithm 8

of the combination of VOEvents and RTML is that the agent can optimize observations knowing a preexisting observation strategy of an heterogeneous telescope network (HTN). To summarize, a VOEvent defines the astronomical characteristics of a target and RTML defines the telescope setup to observe this target. The protocol to send and to receive RTML messages can be whatever we want (e.g., TCP socket, and SOAP) but it must be adapted to the constraints of the ARO or of the HTN.

In Section 3, we took an example of an intelligent agent that sends an RTML message to the telescope agent:

```
<Coordinates>
  <RightAscension>41.25</RightAscension>
  <Declination>+22.5</Declination>
</Coordinates>
```

This is just an excerpt of the RMTL message focused on the tag `Coordinates`. RTML can be used also from a telescope agent to inform the intelligent agent that a task is done.

7. Conclusions

Protocols for robotic telescope networks just begin to be available for few years and their development is not fully completed. Today, the most important heterogeneous telescope network is eSTAR which is also a development platform for VOEvents and RTML [10]. In the next years, thousands of VOEvents are believed to be broadcasted by large digital survey projects (LSST (http://www.lsst.org/) and Pan-Starrs (http://pan-starrs.ifa.hawaii.edu/). It is important to incorporate these protocols into ARO agents as soon as possible, especially in small telescope networks, because they will thus benefit from the VOEvents generated by the large surveys, enabling them to participate actively in the astronomical science of the twenty-first century.

Acknowledgments

The author thanks Matthew Graham and Joshua Bloom for helpful discussions and the CNRS-INSU GDR PCHE for funds provided via the Figaro collaboration.

References

[1] R. R. White and A. Allan, "An overview of the heterogeneous telescope network system: concept, scalability and operation," *Astronomische Nachrichten*, vol. 329, pp. 232–236, 2008.

[2] C. J. Mottram and S. N. Fraser, "Robonet-1.0," *Astronomische Nachrichten*, vol. 329, no. 3, pp. 317–320, 2008.

[3] P. Kubánek, M. Jelínek, and J. French, "The RTS2 protocol," in *Advanced Software and Control for Astronomy II*, vol. 7019 of *Proceedings of SPIE*, p. 92, Marseille, France, June 2008.

[4] Y. Tsapras, R. Street, K. Horne, et al., "RoboNet-II: follow-up observations of microlensing events with a robotic network of telescopes," *Astronomische Nachrichten*, vol. 330, pp. 4–11, 2009.

[5] T. Granzer, "What makes an automated telescope robotic?" *Astronomische Nachrichten*, vol. 325, no. 6–8, pp. 513–518, 2004.

[6] J. van Paradijs, P. J. Groot, T. Galama, et al., "Transient optical emission from the error box of the γ-ray burst of 28 February 1997," *Nature*, vol. 386, pp. 686–689, 1997.

[7] S. D. Barthelmy, T. L. Cline, P. Butterworth, et al., "GRB Coordinates Network (GCN): A status report," in *Gamma-Ray Bursts*, C. Meegan and R. Preece, Eds., vol. 526 of *AIP Conference Proceedings*, pp. 731–735, Springer, Berlin, Germany, 2000.

[8] C. Pennypacker, M. Boer, R. Denny, et al., "RTML—a standard for use of remote telescope enabling ubiquitous use of remote telescopes," *Astronomy and Astrophysics*, vol. 395, pp. 727–731, 2002.

[9] F. V. Hessman, "Remote telescope markup language (RTML)," *Astronomische Nachrichten*, vol. 327, no. 8, pp. 751–757, 2006.

[10] A. Allan, T. Naylor, and E. S. Saunders, "Autonomous software: myth or magic?" *Astronomische Nachrichten*, vol. 329, pp. 266–268, 2008.